Faculté
des Sciences
Département des Sciences Mathématiques

UMONS
Université de Mons

Introduire des éléments d'algorithmique dans un cours de mathématiques: une expérience dans l'enseignement secondaire belge

Mémoire réalisé par Anaïs MEURIST
pour l'obtention du diplôme de Master en sciences mathématiques

Année académique 2013–2014

Directeur: Stéphanie Bridoux

Service: Cellule de didactique des mathématiques

Remerciements

La première personne que je tiens à remercier est Stéphanie Bridoux qui m'a permis de réaliser ce présent travail grâce à ses précieux conseils, ses relectures minutieuses et ses critiques constructives. Je la remercie également pour sa sympathie, son professionnalisme, sa disponibilité et le temps qu'elle m'a consacré.

Je tiens à remercier mes proches de m'avoir soutenue et supportée durant toute ma scolarité.

Je remercie Antonin pour sa patience et son soutien tout au long de cette année.

Mes remerciements s'adressent également à toutes les personnes qui ont contribué, de près ou de loin, à l'élaboration de ce travail : Carine Launois, Anne Tourpe et leurs classes, Hadrien Mélot, Bruno Quoitin, les étudiants et les doctorantes de Stéphanie Bridoux,...

Je remercie aussi Thomas Brihaye et Martine Devleeschouwer d'avoir accepté de rapporter ce travail.

Papy, mes derniers remerciements te sont destinés. De mes premières années à l'école primaire jusqu'à ces dernières années à l'université, tu as toujours été celui qui a le plus cru en moi. J'aurais tant aimé que tu lises ce travail, point d'orgue de mes études, et j'espère que, de là-haut, tu es fier de moi.

Table des matières

Introduction

Dans le cadre de ce mémoire en didactique des mathématiques, nous étudions la possibilité d'intégrer des éléments d'algorithmique dans un cours de mathématiques dans l'enseignement secondaire belge.
Cette problématique nous a semblé pertinente puisque, par exemple, nos voisins français ont adopté cette démarche au lycée, l'équivalent de notre enseignement secondaire supérieur, depuis septembre 2009.

L'objectif principal de ce travail consiste donc en l'élaboration d'une séquence d'enseignement sur l'algorithmique en vue d'une expérimentation dans quelques classes de l'enseignement secondaire belge.

Pour débuter ce travail, nous présentons, dans le chapitre I, le contexte dans lequel il s'inscrit. Nous expliquons, exemples à l'appui, pourquoi il serait légitime de s'intéresser à l'algorithmique dans le cadre d'un travail en didactique des mathématiques.

Comme nos voisins français intègrent déjà des éléments d'algorithmique dans leur enseignement secondaire, nous nous intéressons ensuite à ce qui se passe en France. Dans le chapitre II, nous décrivons les différents types de ressources mises à la disposition des enseignants français pour l'enseignement de l'algorithmique au lycée.
Afin d'obtenir l'avis des chercheurs sur cette situation et de disposer d'un cadre théorique sur lequel nous appuyer, nous nous intéressons à des travaux antérieurs de didacticiens français au chapitre III.

Au chapitre IV, nous précisons notre problématique de recherche et la méthodologie associée.

Avant de concevoir notre séquence d'enseignement, nous avons choisi de nous intéresser à la situation en Belgique en ce qui concerne l'apprentissage de l'algorithmique au travers de cours donnés au niveau universitaire. Nous

analysons donc, au chapitre V, un cours traitant de l'algorithmique afin de comparer ce cours à la situation française. Nous montrons en particulier comment nous retirons de ce cours des éléments à intégrer dans notre séquence d'enseignement.
Dans le chapitre VI, nous nous intéressons à des questionnaires travaillant des éléments d'algorithmique auxquels ont pris part des étudiants universitaires, dans le but de déterminer les difficultés qu'ils rencontrent avec cette matière.

Forte de ces analyses, nous présentons notre séquence d'enseignement au chapitre VII en nous focalisant sur les points de la matière qui posent des problèmes aux étudiants universitaires.

Dans les chapitres VIII et IX, nous décrivons les différentes expérimentations de notre séquence d'enseignement que nous avons menées dans l'enseignement secondaire belge ainsi que l'évaluation de cette séquence. Nous précisons également les difficultés éprouvées par les élèves.

Enfin, nous terminons par une conclusion générale en revenant sur le travail réalisé et en y intégrant un regard critique.

Chapitre I

Contexte du travail

Dans ce premier chapitre, nous expliquons les raisons qui nous ont poussée à nous intéresser à l'algorithmique dans le cadre d'un mémoire en didactique des mathématiques.

I.1 Bilan du projet de master I

Lors de l'année académique 2012-2013, dans le cadre du « projet intégré » dispensé durant la première année de master en sciences mathématiques à finalité didactique, nous nous sommes intéressée à la notion de fonction. À l'issue du second quadrimestre, notre travail final a consisté en une analyse didactique du chapitre « Généralités sur les fonctions » de différents manuels de mathématiques de l'enseignement secondaire supérieur belge.

Ces différents manuels ont été au nombre de trois : le « Mathématisons 57 » (1993), l'« Espace Math 4 » (2002) et l'« Actimath 4 » (2004).
L'Espace Math 4 et l'Actimath 4 sont deux manuels pour la quatrième secondaire conformes aux derniers programmes en vigueur à ce jour (programmes des années 2000). Le Mathématisons 57 est, quant à lui, un manuel antérieur aux derniers programmes. Il était destiné à des élèves de cinquième secondaire avec sept heures de mathématiques par semaine.
Nous avons volontairement travaillé avec un manuel plus ancien pour étudier l'évolution de ce chapitre dans les manuels, surtout en ce qui concerne l'approche par compétences, approche à la base des programmes des années 2000.

Notons également que nous avons décidé de travailler avec le Mathématisons destiné aux élèves de cinquième secondaire car le chapitre concernant les

généralités sur les fonctions se trouve dans ce manuel et non dans le manuel destiné aux élèves de quatrième, contrairement aux deux manuels actuels qui incluent, conformément aux programmes, ce chapitre en quatrième secondaire.

En analysant les trois manuels choisis, nous avons vite constaté que le Mathématisons se démarquait assez bien des deux autres manuels sur différents points. Tout d'abord, toutes les notions présentes dans le chapitre des généralités sur les fonctions dans les deux manuels actuels ne se retrouvent pas forcément dans le chapitre des généralités du Mathématisons. C'est notamment le cas des notions de maximum et de minimum. En effet, dans l'Espace Math et dans l'Actimath, ces notions sont vues en quatrième secondaire, dans le chapitre relatif aux généralités sur les fonctions. A contrario, dans le Mathématisons, ces notions sont vues dans un chapitre suivant les généralités, à savoir le chapitre concernant le rôle des dérivées. En effet, les notions de maximum et de minimum sont fortement liées à la dérivée et le manuel Mathématisons prend le parti de ne définir ces notions que lorsqu'elles auront un réel intérêt, car les deux seuls types de fonctions alors connus des élèves sont la fonction du premier degré et la fonction du second degré.
Nous pensons que l'idée du Mathématisons de ne définir les notions que lorsqu'elles ont un réel intérêt est une bonne idée. De prime abord, cela nous a semblé étrange de ne pas définir les notions de maximum et de minimum dans un chapitre consacré aux généralités sur les fonctions. En réalité, cela nous a semblé bizarre parce que, durant notre cursus secondaire, ces notions ont été vues dans ce chapitre et il nous a paru tout à fait normal qu'elles doivent y figurer. Ceci dit, après réflexion, nous trouvons qu'il n'est pas nécessairement utile de définir ces notions dans ce chapitre si ce n'est pas pour les exploiter et leur donner du sens. En effet, dans les deux manuels actuels, les quelques recherches d'extrema proposées dans les exercices sont des recherches d'extrema globaux et non locaux (même si l'Espace Math définit les notions de maximum et minimum locaux). De plus, ces recherches se font à l'aide d'un graphique ou d'un tableau de valeurs et jamais à l'aide des définitions. Les définitions formelles ne sont donc pas exploitées. Sur la base de ce constat, nous pensons donc qu'il vaut mieux suivre le Mathématisons et ne définir les notions de maximum et de minimum que lorsqu'elles pourront être convenablement exploitées. D'un autre côté, ne voir ces notions que dans le chapitre sur le rôle des dérivées peut induire chez les élèves la conception erronée que ces notions n'existent que pour des fonctions dérivables. Il faudrait donc veiller à étudier des fonctions admettant des extrema et n'étant pas dérivables, comme la

fonction $f : \mathbb{R} \to \mathbb{R}^+ : x \to f(x) = |x|$ qui admet un minimum en $x = 0$, mais qui n'est pas dérivable en ce point.

Ensuite, un autre point de rupture par rapport aux deux manuels actuels concerne les exercices. En effet, dans le Mathématisons, tous les exercices sont contextualisés, c'est-à-dire qu'aucun exercice n'est formulé dans un contexte autre que purement mathématique. Les questions sont donc toujours très explicites quant à ce qu'il faut faire et au(x) point(s) de théorie à utiliser. La FIGURE I.1 nous montre un exercice typique issu de ce manuel.

Pour chacune des fonctions f suivantes, déterminer dom f, Im f, les racines de f; construire point par point le graphe cartésien,

a) $f(x) = \sqrt{x}$ c) $f(x) = \frac{1}{x}$ e) $f(x) = x^4$

b) $f(x) = \sqrt[3]{x}$ d) $f(x) = x^3$ f) $f(x) = |x|$

FIGURE I.1 – Exercice du Mathématisons 57

Dans les deux manuels actuels, nous retrouvons une certaine volonté de décontextualiser les exercices. En effet, bon nombre d'exercices, aussi bien dans un manuel que dans l'autre, sont posés sous la forme d'un problème associé à une mise en situation. Ces exercices ont donc un contexte autre que mathématique et les points de matière à utiliser ne sont pas toujours clairement explicités. La FIGURE I.2 et la FIGURE I.3 nous montrent des exercices décontextualisés issus de l'Espace Math et de l'Actimath.

Enfin, un dernier point distinguant le Mathématisons de l'Espace Math et de l'Actimath est une caractéristique propre du manuel plus ancien par rapport aux manuels récents. Dans le Mathématisons se trouvent divers algorithmes que nous ne retrouvons pas dans les manuels plus récents.
Par exemple, pour trouver l'image d'une valeur par une fonction donnée, le Mathématisons donne la procédure à suivre (l'algorithme) pour y parvenir. Dans le manuel, l'exemple est donné pour la fonction $f : \mathbb{R} \to \mathbb{R} : x \to x - \frac{x^2}{10}$. L'algorithme permettant de trouver l'image d'une valeur x par cette fonction est donné sous la forme d'un ordinogramme à la FIGURE I.4.
Cet algorithme est immédiatement suivi par un autre algorithme permettant de l'exécuter pour des valeurs allant de 0 à 10 par pas de $0,5$. Cet algorithme est illustré à la FIGURE I.5.

Voici un extrait de l'explication accompagnant une déclaration de revenus aux contributions. Il s'agit de l'explication pour calculer forfaitairement les charges professionnelles :

Autres frais professionnels.

Ne complétez cette rubrique que si vous pouvez prouver que vos frais professionnels excèdent le forfait légal.

Ce forfait est calculé sur le total des revenus déclarés aux rubriques A, 4 à 8, a, diminué des montants mentionnés aux rubriques A, 8, b et A, 10 et est égal à :

- 20 p.c. de la première tranche de 4090 € ;
- 10 p.c. de la tranche de 4090 € à 8180 € ;
- 5 p.c. de la tranche de 8180 € à 13 634 € ;
- 3 p.c. de la tranche qui excède 13 634 € ; cependant, ce total ne peut excéder 2730 € (ce montant maximum est atteint avec un revenu de 54 536 €).

Le cas échéant, le résultat obtenu est encore majoré du forfait pour longs déplacements.

Si vous complétez la rubrique A, 11, vous devez fournir le détail de ces frais professionnels dans une annexe.

1) Réalise un graphe cartésien visualisant cette fonction en notant en abscisses les revenus professionnels (1 cm = 5000 €) et en ordonnées les charges déductibles (1 cm = 250 €).

2) À partir de ce graphe cartésien, détermine le montant des charges déductibles pour un revenu de 3000 € , de 7000 € , de 16 000 € , de 25 000 € .

   Effectue aussi le calcul à partir des données du tableau.

   Compare la rapidité et la précision des deux méthodes.

3) Si on veut utiliser la calculatrice, il faut écrire les formules mathématiques qui donnent les charges (y) en fonction des revenus (x).

   Ainsi, pour les premiers 4090 € :

   $y = \frac{20}{100} x$ ou $y = \frac{x}{5}$

   Détermine les formules à utiliser dans les autres cas.

   À l'aide de ces formules, vérifie les calculs réalisés en 2).

   Compare les deux méthodes.

4) Que penser de ce Monsieur qui déduit des charges de 2975 € ?

FIGURE I.2 – Exercice décontextualisé de l'Espace Math 4

Monsieur Gazon tond sa pelouse régulièrement.
À chaque tonte, il utilise 1,6 litre de carburant à 0,95 euro le litre.

a) Exprime le coût d'une tonte en fonction du nombre total x de tontes sur une année, sachant que l'entretien annuel de la tondeuse lui coûte 45 euros.

b) À partir de combien de tontes, le coût d'une tonte est-il inférieur à 6 euros ?

c) Résous graphiquement ce problème.

FIGURE I.3 – Exercice décontextualisé de l'Actimath 4

Il est légitime de se demander l'utilité d'introduire des algorithmes pour effectuer ces démarches. Le premier algorithme traduit la démarche que nous effectuons, parfois même sans plus y prêter attention, lorsque nous implémentons une fonction dans un logiciel graphique, par exemple. Même sans nous en rendre compte, nous effectuons donc des algorithmes assez régulièrement. Le second algorithme correspond à l'évaluation de cette fonction en 21 points. Imaginons que nous devions écrire, sur papier, l'ensemble des calculs à effectuer pour obtenir l'évaluation de cette fonction en ces

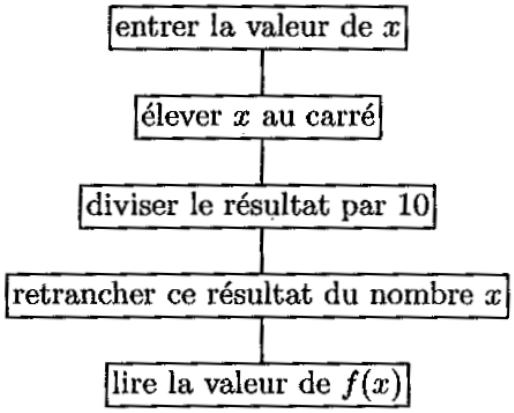


FIGURE I.4 – Algorithme pour le calcul d'une image

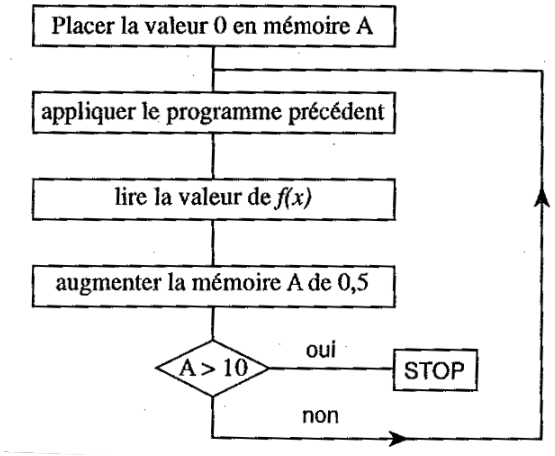


FIGURE I.5 – Algorithme pour le calcul de plusieurs images

21 valeurs. Cela deviendrait assez vite pénible et répétitif. L'algorithme permet, quant à lui, en effectuant la même démarche que nous, de résumer en quelques lignes l'ensemble des opérations à effectuer en se servant du fait que pour les 21 valeurs, il s'agit toujours du même groupe d'opérations à effectuer.
Bien entendu, à l'heure actuelle, il est possible d'obtenir rapidement, à l'aide d'une calculatrice, un tableau de valeurs pour évaluer cette fonction en ces 21 valeurs. Cependant, il est toujours intéressant de comprendre ce qui se passe derrière des programmes tout faits des calculatrices afin de mieux

appréhender les éventuelles erreurs qui pourraient se produire.
Les deux algorithmes présentés permettent donc de mettre des mots et une structure sur des démarches que nous avons l'habitude d'exécuter, parfois même instinctivement. C'est exactement ce genre de démarches qu'il faudrait montrer plus souvent à des élèves du secondaire afin qu'ils aient une meilleure maîtrise et une meilleure perception du fonctionnement de leurs calculatrices.

Face à cette caractéristique propre du Mathématisons, une question nous a interpellée : *pourquoi les algorithmes ont-ils disparu des manuels actuels belges et a fortiori des programmes actuels ?*

I.2 L'algorithmique dans l'enseignement actuel

Pourquoi donc s'intéresser à l'algorithmique dans l'enseignement secondaire supérieur puisqu'elle a disparu des programmes de mathématiques belges francophones actuels et, de surcroît, des manuels conformes à ces programmes ? Deux arguments peuvent être développés pour répondre à cette question.

D'une part, à quelques dizaines de kilomètres de chez nous, chez nos voisins français, l'algorithmique est présente au lycée (l'équivalent de la quatrième, de la cinquième et de la sixième année de l'enseignement secondaire belge).
En effet, depuis 2009, suite au rapport de la commission Kahane (2000), une part d'algorithmique a été introduite dans les programmes de mathématiques de la seconde (quatrième secondaire belge) à la terminale (sixième belge).
De là, certaines ressources à la disposition des enseignants ont dû être actualisées et d'autres encore ont vu le jour.
Nous y reviendrons plus en détail au chapitre suivant.

D'autre part, même s'il n'y a plus de cours d'algorithmique dans le secondaire, des cours d'algorithmique sont toujours dispensés dans des hautes écoles et les universités belges, du moins dans des filières telles que les sciences mathématiques, les sciences physiques, les sciences de l'ingénieur,... Par exemple, durant le bachelier en sciences mathématiques, au moins un cours uniquement consacré à l'algorithmique - et éventuellement à la programmation - est proposé aux étudiants dans les cinq universités belges francophones (UMONS, UNAMUR, UCL, ULB et ULg). Il semble donc évident que l'algorithmique entretient des liens particuliers avec les

mathématiques.
De plus, dans certains cours universitaires où les principaux enjeux sont purement mathématiques et non algorithmiques, le recours aux algorithmes est inévitable. C'est notamment le cas des cours *Introduction to optimization* (Strodiot, 2011-2012) et *Analyse numérique* (Sartenaer, 2011-2012), cours dispensés lors de la troisième année du bachelier en sciences mathématiques à l'Université de Namur et que nous avons eu l'occasion de suivre.
Nous allons donc nous appuyer ici sur notre propre expérience dans le but de montrer l'utilité et surtout la nécessité d'introduire des algorithmes dans certains cours de mathématiques.

Citons des algorithmes rencontrés dans ces deux cours afin d'illustrer nos propos. Dans le cours *Introduction to optimization*, nous sommes notamment en présence du *simplex algorithm* (algorithme du simplexe), algorithme permettant de résoudre des problèmes de programmation linéaire mis sous forme standard, c'est-à-dire des problèmes de la forme :

$$\begin{cases} \min\limits_{x} & z = c^T x, \\ \text{s.c.} & Ax = b, \\ & x \geqslant 0, \end{cases}$$

avec $A \in \mathbb{R}^{m \times n}$, $b \geqslant 0 \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ et $x \in \mathbb{R}^n$.

Supposons que nous ayons à résoudre le problème de programmation linéaire suivant (non mis sous forme standard) :

$$\begin{cases} \max & z = 10x_1 + 15x_2, \\ \text{s.c.} & x_1 + 4x_2 \leqslant 20, \\ & 4x_1 + x_2 \leqslant 35, \\ & x_1, x_2 \geqslant 0. \end{cases}$$

Dans ce cas, il est possible de résoudre ce problème géométriquement car, comme il contient deux variables (x_1 et x_2), il est très facile de représenter la situation dans un repère cartésien.
La résolution de cet exercice est présentée à la FIGURE I.6.

La surface en bleu correspond à l'ensemble des points satisfaisant aux quatre contraintes du problème. Ensuite, nous représentons les courbes de niveau de la fonction z, c'est-à-dire les droites d'équations $10x_1 + 15x_2 = c$,

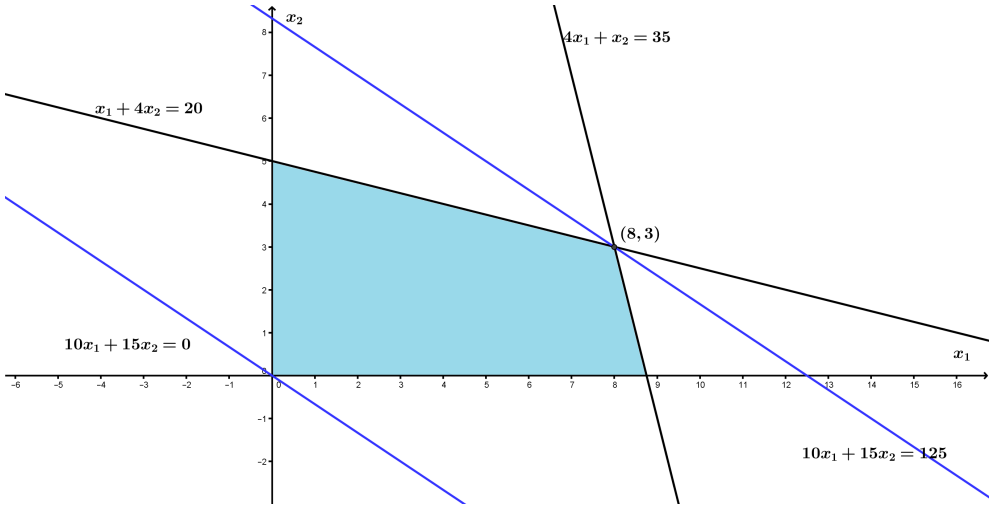


FIGURE I.6 – Résolution d'un problème de programmation linéaire

avec $c \in \mathbb{R}$. Nous avons $\nabla z = (10, 15)$, où ∇z est le gradient de la fonction z, et chaque droite est perpendiculaire à ∇z. Dès lors, en se déplaçant dans le sens de ∇z, le maximum recherché est atteint au dernier point de la région admissible qui appartient aussi à une courbe de niveau. Il s'agit du point $(x_1, x_2) = (8, 3)$, donc max $z = 125$.

Imaginons maintenant que nous ayons un problème avec trois variables ou plus. Dans ce cas, il devient difficile - voire impossible - de représenter la situation comme nous venons de le faire. Nous avons donc besoin d'une procédure permettant de résoudre n'importe quel problème de programmation linéaire. L'algorithmique vient alors au secours des mathématiques afin d'apporter une solution à ce problème. Avec l'algorithme du simplexe, il est possible de résoudre tout problème de programmation linéaire de n'importe quelle dimension. Dans ce cas précis, l'algorithme est donc un outil indispensable dans la résolution d'un problème mathématique. Sans lui, il ne serait pas possible d'y arriver.

Dans ce cours, l'algorithme du simplexe est donné afin d'être exécuté pour résoudre un problème de programmation linéaire, mais il est également étudié en tant que tel. En effet, il est prouvé que l'algorithme se termine en un nombre fini d'étapes afin de s'assurer qu'il ne boucle pas indéfiniment. Une étude du coût en temps est également proposée pour ce même algorithme.

Dans le cours d'*Analyse numérique*, nous retrouvons également un certain nombre d'algorithmes. Nous pouvons, entre autres, citer *the Bisection method* (méthode dichotomique). Cette méthode est en fait un algorithme permettant de trouver une racine d'une fonction f continue sur un intervalle donné $[a, b]$ et vérifiant $f(a).f(b) < 0$.
L'algorithme consiste à évaluer la fonction f en la valeur $\frac{a+b}{2}$ que nous notons z. Si $f(z)$ est du même signe que $f(a)$, l'intervalle $[a, b]$ devient alors $[z, b]$. De même, si $f(z)$ est du même signe que $f(b)$, l'intervalle de départ devient alors $[a, z]$. Ce processus est répété jusqu'à ce que l'erreur sur la valeur réelle de la racine soit suffisamment petite.
Cet algorithme est utilisé dans la démonstration du théorème de la valeur intermédiaire. Ce théorème s'énonce comme suit :

> Soit $f : [a, b] \to \mathbb{R}$ une fonction continue. Alors f prend au moins une fois toute valeur comprise entre $f(a)$ et $f(b)$.

Il est vu dans l'enseignement secondaire, mais il n'est pas démontré. La démonstration est donnée à l'université dans des cours d'analyse, par exemple dans le cours de *Calcul différentiel et intégral* (Strodiot, 2009-2010), cours que nous avons suivi durant notre première année en sciences mathématiques à l'Université de Namur.

La preuve de ce théorème consiste donc à montrer que pour toute valeur μ comprise entre $f(a)$ et $f(b)$, il existe une valeur z comprise entre a et b pour laquelle l'égalité $f(z) = \mu$ est vérifiée. Nous supposons, sans perte de généralité, que $f(a) < f(b)$ et nous devons alors prouver que $\forall \mu \in]f(a), f(b)[: \exists z \in [a, b] : f(z) = \mu$. Pour toute valeur μ de l'intervalle $]f(a), f(b)[$, nous définissons la fonction continue $F : [a, b] \to \mathbb{R} : F(x) = f(x) - \mu$. Elle vérifie les inégalités $F(a) < 0$ et $F(b) > 0$. Nous cherchons désormais à montrer que $\exists z \in [a, b]$ tel que $F(z) = 0$.
Pour cela, nous construisons, par récurrence et en utilisant la méthode dichotomique, une suite d'intervalles emboîtés $\big([a_k, b_k]\big)_{k\in\mathbb{N}}$ avec $[a_0, b_0] = [a, b]$. Ayant construit $[a_k, b_k]$ avec $F(a_k) \leqslant 0$ et $F(b_k) \geqslant 0$, nous calculons la valeur $z_k = \frac{a_k+b_k}{2}$ et nous construisons l'intervalle $[a_{k+1}, b_{k+1}]$ tel que :

$$[a_{k+1}, b_{k+1}] = \begin{cases} [a_k, z_k] & \text{si } F(z_k) \geqslant 0 \\ [z_k, b_k] & \text{si } F(z_k) \leqslant 0. \end{cases}$$

Par le théorème des intervalles emboîtés, comme la longueur des intervalles devient de plus en plus proche de 0, nous obtenons le résultat suivant : $\exists z \in [a, b]$ tel que $a_k \to z$ et $b_k \to z$. Comme la fonction F est continue,

nos limites deviennent $F(a_k) \to F(z)$ et $F(b_k) \to F(z)$, avec $F(a_k) \leqslant 0$ et $F(b_k) \geqslant 0$ quel que soit k. De là, nous en déduisons que $F(z) = 0$, ce qui conclut notre preuve.

L'algorithme de dichotomie est d'autant plus intéressant qu'il est déjà présent dans le manuel Mathématisons 57, dans le chapitre sur la continuité. Il se trouve à la suite de l'énoncé du théorème de la valeur intermédiaire. La FIGURE I.7 nous montre comment cet algorithme est exprimé dans ce manuel.

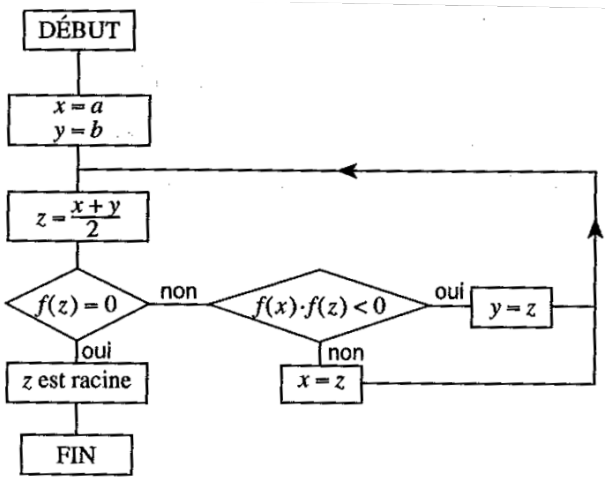


FIGURE I.7 – Algorithme de dichotomie du Mathématisons

Nous constatons que cet algorithme ne s'arrête que lorsque la racine est réellement atteinte et non lorsque nous nous situons suffisamment près de cette racine. Le danger d'un tel algorithme est qu'il est possible que certaines étapes soient répétées indéfiniment puisque nous n'avons pas la certitude que la racine va être exactement atteinte à un moment donné (notamment si la racine est un irrationnel).

Montrons l'utilité de l'algorithme de dichotomie sur un exemple. Nous cherchons à savoir si la fonction continue sur son domaine $f : \mathbb{R} \to \mathbb{R} : x \to e^x + x$ possède une racine. Trouver l'éventuelle racine de cette fonction revient à résoudre l'équation $e^x + x = 0$, ou encore $e^x = -x$. Il faudrait donc trouver un point appartenant aux deux courbes, ce qui n'est malheureusement pas possible, même en tâtonnant. Néanmoins, nous constatons que $f(-1) = e^{-1} + (-1) \simeq -0,63$ et $f(0) = e^0 + 0 = 1$. Nous avons choisi

ces valeurs car nous savons que $\frac{1}{e}$ est inférieur à 1 et qu'en 0, l'exponentielle est facile à calculer. Nous en déduisons donc qu'une racine se situe dans l'intervalle $[-1, 0]$. De là, nous pouvons appliquer l'algorithme de dichotomie afin de trouver une valeur approchée de la racine.
Nous constatons donc que sans cet algorithme ou un autre ayant le même but, il ne serait pas possible de trouver - même si ce n'est qu'une valeur approchée - la valeur de la racine de cette fonction. Une fois encore, nous remarquons qu'un algorithme vient en aide aux mathématiciens afin de résoudre un problème purement mathématique qu'il ne serait pas possible de résoudre sans le recours à un algorithme.

Bien évidemment, il existe d'autres cours universitaires de mathématiques pour lesquels le recours à l'algorithmique est désormais indispensable à la résolution de problèmes typiquement mathématiques. Néanmoins, nous n'allons pas nous étendre davantage sur le sujet.

Pour résumer la situation, les algorithmes ont disparu de l'enseignement des mathématiques dans le secondaire, mais ils sont toujours présents dans des cursus universitaires. Ils n'ont pas disparu des cursus universitaires et ils n'auraient aucune raison d'en disparaître car nous avons vu, au travers de différents exemples, que le recours à l'algorithmique est plus qu'indispensable dans la résolution de certains problèmes posés par les mathématiques. Sans le recours à l'algorithmique, il ne serait tout simplement pas envisageable de résoudre certains problèmes.

I.3 Conclusion

En regardant le manuel Mathématisons et des cours universitaires belges destinés à des étudiants de mathématiques, nous avons pu constater que les mathématiques et l'algorithmique entretenaient des liens particuliers et qu'il y avait moyen de travailler des algorithmes dans un cours de mathématiques. Se pose alors la question de savoir quel genre de mathématiques il serait possible de travailler à l'aide de l'algorithmique dans l'enseignement secondaire et, à plus forte raison, de savoir ce que pourrait apporter l'introduction d'éléments d'algorithmique dans un cours de mathématiques.

Afin d'avoir une idée plus précise sur le sujet, nous allons nous intéresser à ce qui se passe en France. C'est l'objet du chapitre II.

Chapitre II

La situation en France

Dans ce chapitre, nous expliquons pourquoi une part d'algorithmique a été introduite dans les cours de mathématiques de l'enseignement secondaire français. Nous résumons également les contenus des différentes ressources que nous pouvons trouver en France en ce qui concerne l'algorithmique au lycée.

II.1 Vers un enseignement de l'algorithmique au lycée

En France, une « Commission de réflexion sur l'enseignement des mathématiques » s'est créée en 1999 sur commande d'associations de mathématiciens. Cette commission est également dénommée « Commission Kahane », du nom de son président, le professeur Jean-Pierre Kahane. Kahane a eu la mission de réunir des chercheurs et des enseignants afin d'étudier, de façon globale, l'enseignement des mathématiques de l'école élémentaire (primaire belge) jusqu'à l'université.

En décembre 2000 paraît un rapport d'étape intitulé « Informatique et enseignement des mathématiques ». C'est suite à ce rapport que des notions d'algorithmique vont être introduites en 2009 dans les programmes français du lycée. Ci-après, nous résumons les principales idées de ce rapport.

La problématique de la commission concernant l'introduction d'éléments d'informatique au lycée est, d'un côté, de comprendre pourquoi il pourrait être intéressant d'introduire une part d'informatique dans l'enseignement des mathématiques et, d'un autre côté, d'étudier comment il serait

possible de faire évoluer les programmes afin d'y introduire de l'informatique.

II.1.1 Intérêt de l'introduction d'une part d'informatique dans l'enseignement des mathématiques

Introduire une part d'informatique au lycée n'a pas qu'un intérêt purement mathématique puisque des algorithmes et des programmes informatiques sont présents dans notre quotidien, même si nous n'en avons pas forcément conscience. S'y intéresser peut nous permettre de mieux appréhender certains objets du quotidien comme une carte bancaire ou un répondeur téléphonique. De même, des démarches typiques d'un programmeur sont des démarches que nous effectuons - parfois inconsciemment - dans notre vie de tous les jours : décomposer une tâche en tâches élémentaires, évaluer a priori la durée d'un processus, vérifier que le processus donne bien le résultat attendu,...

Concernant les mathématiques, à l'heure actuelle, il est indéniable que l'informatique contribue au développement des mathématiques et à simplifier le travail du mathématicien. De nombreux domaines des mathématiques n'échappent plus à l'expérimentation sur ordinateur. Citons, par exemple, les équations différentielles et les équations aux dérivées partielles. Ces deux types d'équations sont parfois assez complexes à résoudre « à la main » et l'ordinateur permet une simulation numérique afin d'obtenir une idée de l'allure de la solution.
L'ordinateur permet également de traiter des problèmes avec beaucoup de données ou de cas envisageables. C'est notamment le cas lors de la réalisation d'une enquête statistique à grande échelle : l'ordinateur permet d'alléger considérablement le temps de calcul en automatisant certaines tâches. Citons également le théorème des quatre couleurs qui a été prouvé au moyen d'un ordinateur. En effet, il n'était pas envisageable de le démontrer sans, en raison de la multitude de configurations possibles.
Cependant, l'algorithmique pose également de nouvelles questions aux mathématiciens, comme la question de la complexité d'un algorithme. La complexité sert à estimer l'efficacité d'un algorithme ou à comparer différents algorithmes résolvant le même problème. L'étude de la complexité des algorithmiques est même devenue une branche à part entière.

À travers ces différents exemples, nous constatons que l'ordinateur est donc devenu un outil plus que nécessaire pour le mathématicien. Face à cette

nécessité, comment faire évoluer les programmes français afin d'introduire une part d'informatique au lycée ? C'est le deuxième point abordé dans le rapport de la commission Kahane.

II.1.2 Propositions pour une évolution des programmes du lycée

Dans les introductions des programmes en vigueur lors du rapport de la commission, il est demandé que les élèves soient capables d'utiliser une calculatrice scientifique en seconde et en première, et une calculatrice programmable en terminale. Cependant, cette recommandation n'est plus prise en compte dans les différents contenus des programmes. Il y aurait donc déjà là un remaniement des programmes à effectuer afin d'y introduire explicitement cette utilisation des calculatrices scientifique et programmable. Il faut certes distinguer tout ce qui touche à l'utilisation de logiciels sur calculatrices ou ordinateurs de l'algorithmique et de la programmation en eux-mêmes. Tracer une courbe à l'aide d'un logiciel de géométrie ne permet pas forcément de travailler des concepts d'algorithmique.

Concernant la programmation et l'algorithmique, des notions importantes à étudier seraient les boucles, les branchements, la récursivité, mais également la complexité. En effet, traiter ces différents points permet de travailler la logique, en plus de la programmation et de l'algorithmique. En outre, cela permettrait également de mieux comprendre et appréhender les calculatrices et ordinateurs déjà utilisés par les élèves.

L'évolution des programmes peut donc s'effectuer en deux temps. Dans un premier temps, il faudrait intégrer au sein même des programmes les instructions ne figurant que dans leurs introductions et qui sont alors peu appliquées généralement, notamment en ce qui concerne l'utilisation des calculatrices programmables. Dans un second temps, il serait nécessaire de procéder à une évolution des programmes afin d'y insérer les notions et les objets relatifs à la programmation et à l'algorithmique.

Cependant, même si les programmes du lycée évoluent, les enseignants devraient être capables d'assurer ces cours d'algorithmique et de programmation. En effet, tous les enseignants de mathématiques n'ont pas reçu un enseignement d'algorithmique et de programmation durant leurs cursus scolaires. Pour pallier à ce manque, les IREM (Instituts de Recherche

sur l'Enseignement des Mathématiques) diffusent des documents pour la formation continue des professeurs de mathématiques.
À l'époque de la commission, certaines universités françaises proposaient des cours d'algorithmique et de programmation, mais ce n'était pas une règle générale. Il conviendrait donc que tous les futurs enseignants de mathématiques du lycée reçoivent un cours d'algorithmique et de programmation dans leurs formations universitaires.

II.1.3 Conclusion

À travers le rapport de la commission Kahane, nous remarquons, comme nous avons déjà pu le constater au chapitre précédent, que les mathématiques et l'algorithmique sont liées. La commission ne demande pas une révolution dans les programmes du lycée, mais bien une évolution, en partant des programmes existants.
Dès lors, regardons de plus près les programmes sortis après 2009 afin de voir dans quelle mesure le rapport de la commission a été exploité.

II.2 Les programmes du lycée depuis 2009

À partir de 2009, les programmes de mathématiques de la voie générale du lycée ont été modifiés. Les programmes en question sont donc les programmes de seconde (quatrième secondaire belge), première (cinquième année) et terminale (sixième année). Pour la première et la terminale, cette modification concerne les trois séries : économique et sociale (ES), littéraire (L) et scientifique (S). À partir de maintenant, nous désignons ces trois séries par leurs initiales.
Nous allons donc nous intéresser à six programmes de mathématiques :

- seconde (juillet 2009),
- première S (septembre 2010),
- premières ES et L (septembre 2010),
- terminale S (octobre 2011),
- terminales ES et L (octobre 2011),
- terminale ISN (septembre 2011).

Les programmes de premières ES et L ainsi que ceux de terminales ES et L sont groupés car le programme de mathématiques est commun aux deux séries. Nous expliquons par la suite en quoi consiste le programme de terminale ISN.

Dans les cinq premiers programmes, la grande nouveauté par rapport aux programmes antérieurs est l'apparition d'un thème transversal concernant l'algorithmique. Il s'agit donc d'intégrer de l'algorithmique dans les programmes de chaque année.
Dans chacun de ces cinq programmes, nous retrouvons le même paragraphe concernant les objectifs à atteindre en ce qui concerne l'algorithmique.

Dans ce paragraphe, il est relaté que la démarche algorithmique n'est en fait pas nouvelle pour les élèves puisqu'ils ont déjà rencontré des algorithmes durant les années antérieures, notamment avec les algorithmes opératoires et l'algorithme d'Euclide pour la recherche du plus grand commun diviseur de deux naturels. Il s'agit donc en fait de formaliser ce qui a été vu dans le but de comprendre les principes régissant un algorithme (entrée-sortie, affection,...).
La création d'algorithmes et de programmes informatiques demande aux élèves de la rigueur, ainsi que de la vérification et du contrôle. Pour les programmes informatiques, aucun langage n'est exigé par les programmes du lycée.
Concernant le contenu, il est demandé que les élèves soient capables d'écrire des algorithmes et/ou des programmes contenant des entrées et des sorties, des affectations (attribuer une valeur à une variable), des boucles avec un nombre d'itérations donné et des boucles avec une condition d'arrêt donnée.

Les cours d'algorithmique et de programmation ne doivent pas faire l'objet d'un cours spécifique, il faut intégrer des algorithmes et des programmes informatiques tout au long de l'année dans les chapitres purement mathématiques (fonctions,...).
Dans ces cinq programmes du lycée, lorsqu'il est possible de travailler de l'algorithmique et/ou de la programmation, cela est signalé par un symbole spécifique.
Regardons donc de plus près ces cinq programmes afin de donner des exemples d'applications des algorithmes dans le cours de mathématiques.

II.2.1 Le programme de seconde

Le programme de seconde est divisé en trois parties : fonctions, géométrie, statistiques et probabilités. Dans chaque partie, il est possible de travailler des algorithmes. Citons, pour chacune, des exemples d'applications de type algorithmique.

Dans la partie concernant les fonctions, deux activités algorithmiques sont proposées. La première concerne la représentation de courbes. Il est possible de faire écrire aux élèves un algorithme permettant de tracer une courbe pour des fonctions définies par morceaux.
La seconde concerne la recherche de racines. Dans le programme, il est proposé de chercher la racine d'une équation par la méthode dichotomique. C'est la méthode que nous avons déjà exposée dans le chapitre I.

Pour la géométrie, seule une activité est proposée pour l'algorithmique. Il s'agit de mettre sous la forme d'algorithmes des propriétés géométriques afin de résoudre certains problèmes.

Finalement, par rapport aux statistiques et aux probabilités, deux points concernent l'algorithmique. Il s'agit, d'une part, de créer des algorithmes avec des instructions conditionnelles et d'autre part, d'écrire des algorithmes permettant la répétition d'expériences aléatoires.

II.2.2 Les programmes de première

Comme les programmes de première S et de premières ES et L sont assez similaires en ce qui concerne l'algorithmique, nous choisissons de les traiter ensemble.

Le programme de première S est divisé en trois parties : analyse, géométrie, statistiques et probabilités. Celui de premières ES et L est quant à lui divisé en deux parties : algèbre et analyse, statistiques et probabilités. Le fait qu'il y ait un chapitre en plus dans le programme de première S ne pose aucun problème d'un point de vue algorithmique puisqu'aucune activité de type algorithmique n'est proposée dans cette partie.

Dans les parties relatives à l'algèbre et à l'analyse, nous retrouvons les mêmes propositions d'activités dans les deux programmes. Les algorithmes proposés concernent la résolution d'une équation du second degré et le calcul d'un terme d'une suite numérique. Il est également possible de créer des algorithmes pour comparer les évolutions de deux suites ainsi que des algorithmes pour déterminer à partir de quel moment les termes d'une suite vont être supérieurs à une valeur donnée.

Concernant la partie sur les statistiques et les probabilités, les algorithmes peuvent être utilisés pour simuler des lois de probabilités : la loi binomiale (dans les deux programmes) et la loi géométrique (en première S).

II.2.3 Les programmes de terminale

À nouveau, les programmes de terminale S et de terminales ES et L étant assez similaires concernant l'algorithmique, nous décidons de ne pas les traiter séparément.

Le programme de terminale S est à nouveau divisé en trois parties : analyse, géométrie, probabilités et statistique. Celui de terminales ES et L est divisé en deux parties : analyse, probabilités et statistique.
Une fois de plus, dans le programme de terminale S se trouve une partie relative à la géométrie non présente de le programme de terminales ES et L, mais à nouveau, aucun algorithme n'est proposé dans cette partie.

Dans la partie consacrée à l'analyse, nous retrouvons, comme en première, des algorithmes traitant des suites où il faut déterminer le rang à partir duquel une suite est supérieure ou inférieure à une valeur donnée. Dans le programme de terminale S, il est également demandé de mener des activités algorithmiques dans le cadre des suites et de la résolution d'équations du type $f(x) = k$.

En probabilités et statistique, aucun algorithme n'est proposé en terminales ES et L. En ce qui concerne la terminale S, seul un algorithme est proposé : la simulation d'une marche aléatoire.

II.2.4 Le programme de terminale ISN

Le cours d'ISN (Informatique et Sciences du Numérique) est un cours de spécialité (non obligatoire) dispensé aux élèves de terminale S. Le programme est divisé en quatre parties : représentation de l'information, algorithmique, langages et programmation et architectures matérielles.
Regardons donc de plus près la partie concernant l'algorithmique, étant donné que c'est celle qui nous intéresse dans le cadre de ce travail.

Dans l'introduction de la partie consacrée à l'algorithmique, il est bien précisé que ce cours se base sur les notions d'algorithmique vues en seconde et en première. Les algorithmes proposés sont des algorithmes de tri et de théorie des graphes : recherche d'un élément dans un tableau trié, tri par sélection, tri par fusion, recherche du plus court chemin,...
Les objectifs quant à l'algorithmique sont spécifiés : il s'agit de comprendre un algorithme, d'en concevoir un, d'écrire un programme à partir d'un algorithme, mais également de se poser la question de l'efficacité d'un algorithme.

Ce dernier objectif n'était exigé dans aucun des cinq programmes précédents.

II.2.5 Conclusion

Après avoir regardé les programmes de mathématiques de la voie générale du lycée à partir de 2009, nous constatons que l'algorithmique est donc bien entrée dans les programmes. Cependant, nous avons remarqué que dans certaines parties des programmes, peu ou pas d'algorithmes sont suggérés. De plus, certaines suggestions sont assez vagues, comme par exemple, *des activités algorithmiques seront menées dans ce cadre.*
Afin d'aider les enseignants à se familiariser avec l'algorithmique et avec les demandes des programmes, un document *Ressources pour la classe de seconde - Algorithmique* - a vu le jour en juin 2009.
Nous en détaillons son contenu au paragraphe suivant.

II.3 Le document « ressources » pour l'algorithmique

Lors de la mise en place de ces nouveaux programmes, le ministère de l'éducation nationale a également élaboré un dossier d'accompagnement pour les enseignants, intitulé *Ressources pour la classe de seconde - Algorithmique* -. Celui-ci est paru la même année que le nouveau programme de seconde.
Ce document destiné aux enseignants du lycée a pour but de les familiariser avec les notions d'algorithmique et de programmation. En effet, comme déjà dit dans le paragraphe concernant le rapport de la commission Kahane, tous les enseignants de mathématiques n'ont pas forcément eu de cours d'algorithmique et de programmation dans leurs cursus. De plus, ce document « ressources » propose quelques situations d'algorithmique pour la classe de seconde. Il est divisé en sept parties.

La première partie est intitulée *Présentation générale*. Comme dans le rapport de la commission Kahane, il est précisé que nous rencontrons des algorithmes au quotidien et que les élèves en ont déjà rencontré au cours de leur scolarité (mise au même dénominateur de fractions, résolution d'une équation du second degré,...).
De plus, l'introduction de nouveaux éléments d'algorithmique devrait être motivée. Par exemple, pour introduire les boucles, il faudrait en faire sentir la nécessité lors d'un grand nombre de répétitions de la même opération.

Dans cette introduction du document, il est également précisé que l'introduction d'éléments d'algorithmique et de programmation n'a pas pour but de former des élèves programmeurs : la programmation ne doit pas être trop compliquée. D'ailleurs, il est spécifié que travailler de l'algorithmique ne se résume pas à programmer.
Les activités possibles des élèves face à l'algorithmique sont multiples : comprendre le but d'un algorithme donné afin d'éventuellement devoir le modifier, construire un algorithme résolvant un problème donné, analyser un algorithme,...

La deuxième partie du document, intitulée *Une initiation à l'algorithmique*, résume les éléments constitutifs d'un algorithme. Afin d'illustrer ces différents éléments, un exemple est donné. Il s'agit de lancer deux dés et d'observer la somme des points obtenus par ces deux dés. Si cette somme vaut 8, alors le joueur gagne 10€, sinon, il perd 1€.
À partir de cet exemple, l'affectation est utilisée. En effet, il faut attribuer une valeur à chaque dé. Il y a donc deux affectations à effectuer. La valeur pour chaque dé doit être comprise entre 1 et 6.
Ensuite, dans le problème de départ, il y a un « si...alors...sinon ». Il s'agit de la structure alternative. Si la condition est vérifiée (la somme vaut 8), il faut exécuter ce qui se trouve après le « alors » (le joueur gagne 10€). Si la condition n'est pas vérifiée (la somme est différente de 8), c'est ce qui suit le « sinon » qui est exécuté (le joueur perd 1€).
Une première variante de ce problème est proposée : le joueur joue dix fois à ce jeu et cumule ses gains. Dans ce cas, il faut introduire une boucle (structure répétitive) afin de répéter l'opération du lancer de dés dix fois. Ce type de boucle est noté « pour I allant de 1 à N faire ». Il s'agit donc de répéter ce qui se trouve après le « faire » jusqu'à ce que I atteigne la valeur N (10 dans notre exemple).
La deuxième variante du jeu initial consiste à jouer jusqu'à ce que le gain du joueur atteigne 5€. La structure à ajouter est encore une fois une boucle puisque le lancer de dés va à nouveau être répété. La seule différence par rapport à avant, c'est que nous ne connaissons pas, avant le début du jeu, le nombre de fois que les dés vont devoir être lancés. Le type de boucle utilisé dans ce cas est la boucle « tant que ». Cette boucle s'exprime de la sorte : « tant que *condition* faire ». Si la condition est vraie, il faut exécuter ce qui se trouve après le « faire ». Ensuite, la condition est à nouveau testée. Ce processus est répété jusqu'à ce que la condition devienne fausse.
La boucle « répète » est également citée. Cette boucle s'appuie sur le même principe que la précédente sauf que la condition se trouve après le corps de la boucle et non avant. Un problème majeur de ces types de boucles est

que rien ne nous dit que la condition va à un moment devenir fausse et que l'algorithme ne va pas boucler indéfiniment.
Les notions d'entrée et de sortie sont également définies comme étant respectivement la lecture des données et l'écriture des données.

Après cette initiation à l'algorithmique arrive la troisième partie avec des *Exemples de dispositifs de classe*. Dans cette partie, nous trouvons d'abord des jeux à proposer aux élèves dans le but de les faire réfléchir à des algorithmes sans pour autant introduire de formalisme. Il y a, par exemple, le *jeu du cartable* consistant à réfléchir à une méthode optimale pour préparer son cartable pour le lendemain de façon à avoir tout le nécessaire, mais rien de plus.
Ensuite se trouvent des exemples d'algorithmes de la vie quotidienne que les élèves doivent comprendre et reproduire eux-mêmes. Par exemple, les élèves ont à réfléchir à l'utilisation d'une carte repas à l'école. En effet, il faut voir si l'élève a réservé son repas dans la matinée, s'il a encore de l'argent sur sa carte,...
La dernière section de cette troisième partie consiste en la lecture d'algorithmes. Des algorithmes sont donnés et les élèves doivent répondre à des questions les concernant. Nous trouvons notamment l'algorithme nommé « un peu d'épargne » dont les opérations sont décrites à la FIGURE II.1.

```
Mettre 5000 dans S
Mettre 0 dans N
Effacer l'écran
Tant que S est strictement inférieur à 8000
        Remplacer S par S*1,02
        Augmenter N de 1
        Afficher N et S
Fin du Tant Que
```

FIGURE II.1 – Algorithme à comprendre

Dans cet algorithme, S désigne la somme détenue par la personne et N, le nombre de semestres du placement. Les élèves doivent, à partir de cet algorithme, écrire tout ce qui va apparaître à l'écran et trouver un problème pour lequel cet algorithme est solution.

Les trois parties suivantes du document, dénommées *Algorithmes et géométrie*, *Algorithmes et fonctions* et *Algorithmes et probabilités* donnent des exemples d'algorithmes en rapport avec chacune des trois parties du programme de seconde. Ces algorithmes sont d'abord donnés en pseudo-code puis dans un langage de programmation (calculatrice Casio, calculatrice TI,

Xcas, Python, Scratch,...).
Certains algorithmes présents dans ces parties sont ceux demandés par le programme de seconde, mais nous trouvons également d'autres algorithmes. Nous en présentons quelques uns.

Dans la partie consacrée à la géométrie, un algorithme permettant de déterminer le milieu d'un segment $[A, B]$ est donné. Le code pour l'implémenter sur une calculatrice Casio est également noté. L'algorithme et le code sont illustrés à la FIGURE II.2.

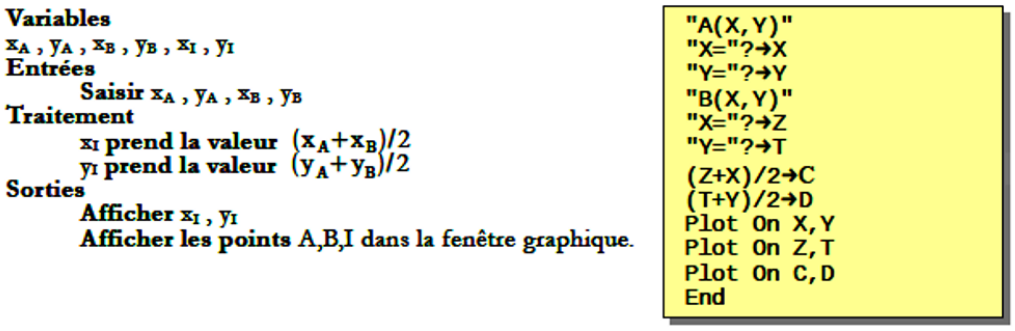


FIGURE II.2 – Algorithme pour déterminer le milieu d'un segment

Dans la partie relative aux fonctions, nous trouvons un algorithme permettant d'essayer de trouver le maximum et le minimum d'une fonction en choisissant, au hasard, un certain nombre d'antécédents. Bien entendu, au plus le nombre d'antécédents sera grand, au plus nous aurons de chance de ne commettre qu'une erreur minime sur les vraies valeurs du maximum et du minimum. L'algorithme est donné en pseudo-code et il est également traduit dans le langage Python pour la fonction $f : [-5, 5] \to \mathbb{R} : x \mapsto 3x - x^2$. La FIGURE II.3 illustre cet exemple.

Dans la partie sur les probabilités, dernière partie consacrée à la matière de seconde, se trouve un algorithme permettant de tester si au moins deux élèves d'une classe de trente partagent la même date d'anniversaire. Il faut donc tirer au sort les trente dates puis ensuite les comparer pour déterminer si deux élèves sont nés le même jour. Dans ce cas-là, les trente dates sont mémorisées dans un tableau. L'algorithme est à nouveau donné en pseudo-code puis traduit dans le langage Scilab. Cet exemple est illustré à la FIGURE II.4.

Variables
a, b les bornes de l'intervalle
f la fonction à étudier
N le nombre d'images à calculer.
Initialisation
min **prend la valeur** $f(a)$
max **prend la valeur** $f(a)$
Traitement
Pour k **variant de** 1 **à** N
x **prend une valeur aléatoire entre** a **et** b.
Si $f(x) >$ max **alors**
max **prend la valeur** $f(x)$
Si $f(x) <$ min **alors**
min **prend la valeur** $f(x)$
Sortie
Afficher min **et** max

```
def f(x):
 return 3.0*x-x**2
import random
a=-5.0
b=5.0
N=50
minimum=f(a)
maximum=f(b)
for k in range(N):
 x=random.uniform(a,b)
 if f(x)>maximum:
     maximum=f(x)
 if f(x)<minimum:
     minimum=f(x)
print maximum
print minimum
```

FIGURE II.3 – Algorithme pour la recherche d'extrema

Variables
dates : tableau des trente jours d'anniversaire
trouvé : un booléen qui indique si deux dates coïncident.
k, p: deux compteurs de boucles.
Initialisation
Pour k **de** 0 **à** 29
dates[k] **prend une valeur** entière aléatoire
comprise entre 1 et 365 inclus
trouvé **prend la valeur** faux
Traitement
Pour k **de** 0 **à** 28
Pour p **de** k+1 **à** 29
Si dates[k] = dates[p] **alors**
trouvé **prend la valeur** vrai
Sortie
Affiche trouvé

```
dates=floor(rand(30,1)*365+1);
trouve=%F;
for k=1:29
  for p=k+1:30
    if (dates(p)==dates(k))
      trouve=%T;
    end;
  end;
end;
if (trouve)
  disp("Deux personnes ont même
    anniversaire");
else
  disp("Pas deux anniversaires
    communs");
end;
```

FIGURE II.4 – Algorithme pour les anniversaires

Finalement, la septième et dernière partie du document « ressources » est un tableau récapitulatif des différents langages de programmation rencontrés dans le document. Toutes les structures sont données dans les différents langages (affectation, boucles,...).
Rappelons qu'aucun langage de programmation n'est imposé par les programmes du lycée.

II.4 Le manuel « Math Repères »

Face à l'évolution des programmes, les manuels de mathématiques destinés au lycée ont dû s'adapter afin d'intégrer la part d'algorithmique exigée par les programmes.
Regardons donc de plus près un manuel français destiné à des élèves de seconde afin de voir comment cette part y a été introduite. Le manuel choisi est le manuel « Math Repères ».
Nous avons choisi de ne présenter qu'un seul manuel pour le lycée car notre but n'est pas d'effectuer une description exhaustive de l'ensemble des ressources disponibles en France. Nous souhaitons simplement présenter les différents types de ressources mises à la disposition des enseignants.

II.4.1 Introduction

Dans ce manuel, avant la matière de seconde se trouve une partie intitulée *À la découverte des algorithmes et de la calculatrice*. Cette partie est divisée en deux sous-parties : une concernant les algorithmes et l'autre concernant les calculatrices TI et Casio. Nous allons donc seulement nous intéresser à la partie relative aux algorithmes.

Dans cette partie, le manuel commence par définir ce qu'est un algorithme : *un algorithme est une méthode, une « recette » qui va permettre d'obtenir un résultat*. Des exemples, tantôt de la vie quotidienne, tantôt mathématiques, sont alors donnés pour illustrer ce concept : règle de l'accord d'un participe passé avec l'auxiliaire « avoir », recette de cuisine, théorème de Pythagore,... Il est également précisé que le nombre d'étapes dans un algorithme doit être fini.
Différents langages de programmation sont ensuite présentés : Algobox, Scratch et les deux calculatrices.

La suite de cette partie concerne l'écriture et la compréhension des algorithmes. Nous trouvons les notions d'entrée et de sortie, la structure « si...alors...sinon », la boucle « pour », la boucle « tant que » et la boucle « répète ».
Chaque notion est d'abord illustrée sur un exemple issu de la vie quotidienne. Par exemple, la boucle « pour » est introduite avec le cas du gsm. Pour envoyer un message, il faut d'abord créer le message, puis écrire le message, ensuite sélectionner un contact et finalement envoyer le message. Si nous souhaitons envoyer plusieurs messages à la suite, nous allons répéter la

même démarche plusieurs fois. Nous allons donc utiliser une boucle « pour ».

Suite aux exemples du quotidien, ces différents concepts sont traduits formellement à l'aide des structures algorithmiques. Dans le cas de la boucle « tant que », la structure d'une telle boucle est donnée comme suit :

```
Tant que condition est vraie
| Instructions
Fin tant que
```

Ensuite, nous trouvons un autre exemple d'algorithme utilisant chaque notion. Ces algorithmes sont analysés pour bien comprendre comment ils fonctionnent. Ils sont ensuite traduits dans les quatre langages de programmation cités plus haut.

Finalement, un exercice résolu et des exercices sont donnés pour chaque notion. Les exercices résolus partent d'un problème en français qu'il faut résoudre au moyen d'un algorithme puis qu'il faut programmer en vue de tester l'algorithme et le code. Les exercices sont aussi bien des exercices d'exécution d'un code donné que des exercices de conception d'algorithmes et de programmes informatiques. Les exercices concernant la boucle « pour » sont mélangés avec ceux relatifs à la boucle « tant que ».

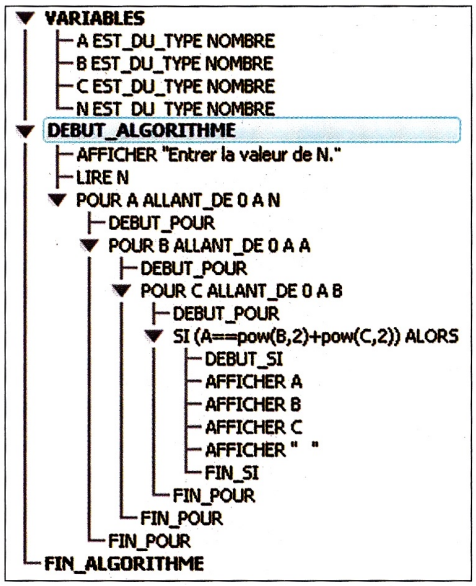
L'avant-dernière page de cette partie - la dernière étant un glossaire reprenant toutes les notions - propose des exercices mélangeant toutes les différentes notions. À nouveau, il faut autant comprendre un code que savoir écrire un algorithme et/ou un code. Plusieurs notions se retrouvent parfois au sein d'un même exercice. C'est le cas de l'exercice donné à la FIGURE II.5 où un « si...alors...sinon » se trouve au milieu d'une boucle « pour ».

II.4.2 La matière de seconde

Comme demandé dans le programme de seconde, des algorithmes se trouvent dans les trois parties de la matière (fonctions, statistiques et probabilités, géométrie). Ceux-ci sont notifiés par un symbole « Algo ». Les activités de type algorithmique proposées par le manuel étant assez nombreuses, nous n'en citerons que quelques unes. Le but de ces activités est en général de savoir exécuter un algorithme ou un code donné. Quelques activités proposent cependant aux élèves d'écrire un algorithme et/ou un code.

5. Trop « For » avec Algobox

Voici la copie d'écran d'un algorithme écrit sous le logiciel Algobox :



1. Si possible, reproduire cet algorithme sous Algobox.

2 a. Traduire de l'anglais au français le mot « power ».

b. Rechercher, dans les calculs autorisés sur les fonctions numériques, à quoi correspondent : « pow(B,2) » et « pow(C,2) ».

3. Exécuter l'algorithme avec des valeurs entières de N (N = {0 ; 1 ; 2 ; 5 ; 10 ; 25 ; 35 ; 78 ; 100}).

4. Traduire alors cet algorithme en langage naturel et expliquer sa fonction.

5. Le tester alors pour N = 200.
Quel problème dit « de sécurité » rencontre Algobox ?

6. a. Traduire de l'anglais au français les mots « square » et « root ».

b. Quelle opération mathématique est traduite par « sqrt(A) » ?

7. **Modifier** alors la **ligne 11** de l'algorithme, en remplaçant « A » par « sqrt(A) ».
Tester de nouveau l'algorithme avec N = 200, N = 500 et N = 1 000.

8. Montrer que l'on a nécessairement $0 \leqslant B \leqslant \sqrt{A}$.
Expliquer alors pourquoi cet algorithme est plus performant que le précédent.

FIGURE II.5 – Algorithme avec un « pour » et un « si...alors...sinon »

Dans la partie sur les fonctions, nous trouvons un exercice consistant à associer une expression algébrique à un algorithme. Il s'agit du même type d'algorithme que celui du Mathématisons 57 présenté dans le chapitre I.
En outre, dans le document « ressources », nous avons présenté un algorithme permettant de trouver (ou de s'approcher) du maximum et du minimum d'une fonction. Dans le manuel Math Repères, un algorithme similaire est donné aux élèves dans le but de trouver le maximum et le minimum de la fonction $f(x) = x^3 - 3x - 1$.
Ces deux exercices sont donnés à la FIGURE II.6.

Dans les chapitres relatifs aux statistiques et aux probabilités, beaucoup d'algorithmes ont recours au hasard. C'est notamment le cas de l'exercice présenté à la FIGURE II.7.

11. **Une question d'ordre**

Associer à chaque algorithme de calcul son expression algébrique.

A	B
Choisir un nombre x. Élever au carré. Prendre l'opposé. Ajouter 1.	Choisir un nombre x. Prendre l'opposé. Élever au carré. Ajouter 1.
C	**D**
Choisir un nombre x. Ajouter 1. Élever au carré. Prendre l'opposé.	Choisir un nombre x. Prendre l'opposé. Ajouter 1. Élever au carré.

1	$(-x+1)^2$	**2**	$(-x)^2+1$
3	$-x^2+1$	**4**	$-(x+1)^2$

65. **Être à son maximum**

Soit f la fonction définie sur $[-2\,;2]$ par :

$$f(x) = x^3 - 3x - 1.$$

On donne l'algorithme suivant :

Variables : x, y, deux nombres réels.
m et M, deux nombres entiers.
Début
Affecter à m la valeur de $(-2)^3 - 3(-2) - 1$.
Affecter à M la valeur de $(-2)^3 - 3(-2) - 1$.
Pour x allant de -1 à 2 de 1 en 1 :
  Affecter à y la valeur $x^3 - 3x - 1$.
  Si $y > M$
    alors affecter à M la valeur de y.
  Fin Si
  Si $y < m$
    alors affecter à m la valeur de y.
  Fin Si
Fin Pour
Afficher m et M.
Fin

1. Quel est l'affichage obtenu avec cet algorithme ?
2. En utilisant les fonctions « Minimum » et « Maximum » d'une calculatrice graphique, vérifier que les résultats obtenus sont bien les minimum et maximum de la fonction f.

FIGURE II.6 – Exercices sur les fonctions

Pour chaque algorithme présenté dans les exercices 63 *à* 66, *préciser sa fonction, puis le programmer sur sa calculatrice pour le tester.*

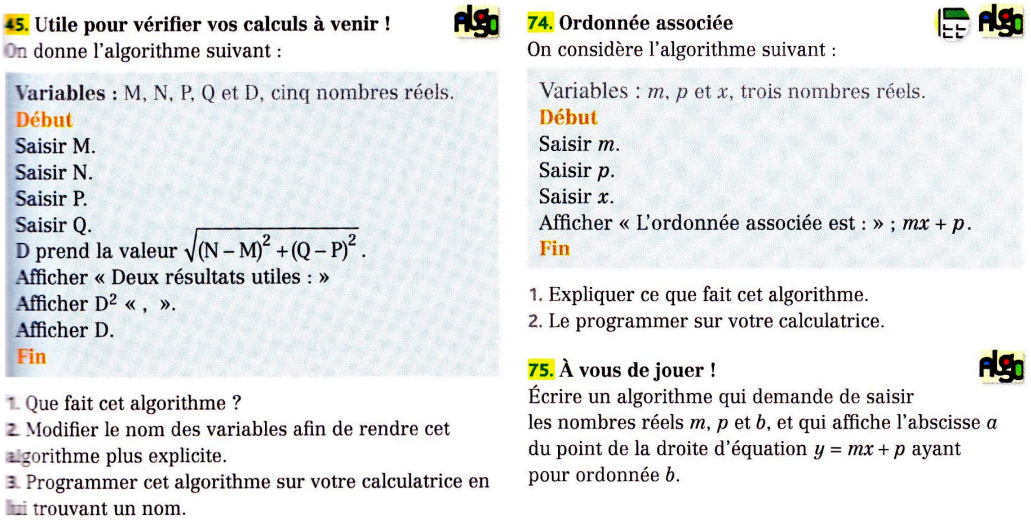
63. **Simuler une expérience**

Variables :
N, L, I, trois nombres entiers naturels non nuls.
Début
Saisir N.
Pour I allant de 1 à N :
  Affecter à L un entier aléatoire entre 1 et 6.
  Afficher L.
Fin Pour
Fin

FIGURE II.7 – Exercice sur les statistiques et les probabilités

En géométrie, nous trouvons des propriétés géométriques mises sous la forme d'algorithmes, comme demandé dans le programme de seconde. C'est notamment le cas pour le calcul de la distance entre deux points pour lequel l'algorithme est donné aux élèves.

Deux exercices sont illustrés à la FIGURE II.8 : celui pour le calcul de la distance et un où les élèves doivent écrire un algorithme.
Pour l'exercice 75 (écriture d'un algorithme), nous avons aussi mis l'exercice 74 car l'algorithme présent dans cet exercice est assez similaire à celui à créer dans l'exercice 75.



45. Utile pour vérifier vos calculs à venir !
On donne l'algorithme suivant :

Variables : M, N, P, Q et D, cinq nombres réels.
Début
Saisir M.
Saisir N.
Saisir P.
Saisir Q.
D prend la valeur $\sqrt{(N-M)^2+(Q-P)^2}$.
Afficher « Deux résultats utiles : »
Afficher D^2 « , ».
Afficher D.
Fin

1. Que fait cet algorithme ?
2. Modifier le nom des variables afin de rendre cet algorithme plus explicite.
3. Programmer cet algorithme sur votre calculatrice en lui trouvant un nom.

74. Ordonnée associée
On considère l'algorithme suivant :

Variables : m, p et x, trois nombres réels.
Début
Saisir m.
Saisir p.
Saisir x.
Afficher « L'ordonnée associée est : » ; $mx + p$.
Fin

1. Expliquer ce que fait cet algorithme.
2. Le programmer sur votre calculatrice.

75. À vous de jouer !
Écrire un algorithme qui demande de saisir les nombres réels m, p et b, et qui affiche l'abscisse a du point de la droite d'équation $y = mx + p$ ayant pour ordonnée b.

FIGURE II.8 – Exercices de géométrie

II.5 Les IREM

Suite à l'évolution des programmes pour le lycée, nous avons vu qu'un document « ressources » pour l'algorithmique avait été mis à la disposition des enseignants et que les manuels français avaient également adapté leurs contenus. Outre ces ressources pour les enseignants, des groupes « algorithmique » se sont créés au sein des IREM (Instituts de Recherche sur l'Enseignement des Mathématiques) dès 2009. Ces groupes ont produit des documents en vue de la formation des enseignants à l'algorithmique, mais également des documents destinés à des activités pouvant être proposées en classe.

À titre d'exemple, le groupe « algorithmique » de l'IREM de Grenoble a rédigé un document pour la formation à l'algorithmique. Nous y retrouvons des situations plutôt destinées aux enseignants et d'autres destinées à des

activités à proposer en classe.
Citons notamment l'activité « Recherche dichotomique ». Le but est d'écrire un algorithme de calcul itératif dans le cadre de la recherche de racines d'une fonction (pour la méthode de dichotomie, voir le chapitre I).
Avant d'étudier cet algorithme, le groupe de travail propose un autre exemple plus parlant qui utilise la dichotomie : la recherche d'un nombre entre 1 et 1000. Il est demandé aux élèves d'imaginer une stratégie permettant de trouver le nombre en moins de 10 propositions. Dès qu'une valeur est proposée, il faut dire si cette valeur est supérieure ou inférieure au nombre recherché. Ensuite, il faut programmer et tester l'algorithme découlant de cette stratégie en Python. Finalement, une étude de cet algorithme est demandée afin de montrer qu'avec la méthode de dichotomie, 10 essais seront toujours suffisants pour trouver n'importe quelle valeur entre 1 et 1000.
Cet extrait du document de l'IREM de Grenoble se trouve dans l'annexe A.

II.6 Bilan

Dans ce chapitre, nous avons vu que le point de départ du dernier changement des programmes de mathématiques au lycée en France était le rapport de la commission Kahane. En 2000, cette commission de réflexion sur l'enseignement des mathématiques a fait paraître un rapport sur l'introduction d'éléments d'informatique dans le cours de mathématiques. La principale raison pour intégrer de l'informatique est son omniprésence dans les mathématiques à l'heure actuelle. En effet, pour les expérimentations mathématiques, le recours à l'informatique est parfois plus que nécessaire.
Cependant, avant d'envisager un changement dans les programmes, il faut d'abord veiller à la formation des enseignants. La commission suggère donc d'intégrer des cours d'algorithmique et d'informatique dans les cursus des futurs enseignants de mathématiques.

Suite au rapport de la commission, les programmes de mathématiques de la seconde à la terminale ont été modifiés à partir de 2009. Les programmes de ces trois années ont donc intégré un thème transversal consacré à l'algorithmique. Les éléments d'algorithmique ne doivent pas être isolés : ils doivent s'intégrer dans les différentes matières des programmes (analyse, géométrie et statistiques et probabilités).

Étant donné que tous les enseignants de mathématiques du lycée n'ont pas reçu une formation à l'algorithmique durant leurs études, un document *Ressources pour la classe de seconde - Algorithmique* - est paru en 2009. Ce document a pour but de familiariser les enseignants avec l'algorithmique, ainsi que de leur suggérer des situations algorithmiques à proposer en classe dans le cadre des différentes matières du programme.

Face à l'évolution des programmes, les manuels scolaires ont également dû adapter leurs contenus afin d'intégrer au mieux la part d'algorithmique requise par les programmes. Dans le manuel consulté, le « Math Repères » pour la seconde, une introduction à l'algorithmique est d'abord proposée afin d'étudier tous les concepts utiles dans l'écriture et la compréhension d'un algorithme. Ensuite, dans les parties relatives aux matières de la seconde, des exercices mettant en jeu des algorithmes sont proposés.

Finalement, toujours dans le but d'aider les enseignants avec cette nouvelle matière, des groupes « algorithmique » se sont formés dans les différents IREM de France. Ces groupes proposent des ressources pour la formation des enseignants à l'algorithmique, mais également des documents à proposer en classe.

Nous venons donc de dresser, au travers de ce chapitre, la situation concernant l'algorithmique au lycée en France. Pour compléter ce travail, nous nous sommes demandée quel était l'avis des didacticiens des mathématiques sur l'introduction de l'algorithmique dans l'enseignement français.

Chapitre III

Travaux antérieurs

Dans ce chapitre, nous nous intéressons à des travaux de didacticiens français sur l'introduction de l'algorithmique au lycée afin d'obtenir leur avis sur la situation en France.

III.1 Une définition d'« algorithme »

L'introduction de l'algorithmique dans l'enseignement des mathématiques au lycée étant assez récente, peu de travaux en didactique des mathématiques existent sur le sujet à ce jour.

Modeste, didacticien des mathématiques français, propose d'étudier, en collaboration avec Ouvrier-Buffet et Gravier (2010) puis seul dans sa thèse (2012), cette introduction de l'algorithmique dans les cours de mathématiques au lycée. Travailler sur l'algorithmique du point de vue de la didactique des mathématiques est assez nouveau puisqu'avant, les chercheurs s'intéressaient plutôt à l'algorithmique en tant que telle. L'intérêt d'étudier l'algorithmique d'un point de vue mathématique, c'est que l'algorithme, en plus d'être un outil pour la résolution de problèmes, est également un objet des mathématiques. Nous y reviendrons.

Avant d'aller plus loin, il convient de définir formellement la notion d'« algorithme ». Après avoir comparé diverses définitions de ce terme, Modeste en donne la définition suivante :

> Un **algorithme** est une procédure de résolution de problème, s'appliquant à une famille d'instances du problème et produisant, en un nombre fini d'étapes constructives, effectives, non-ambigües et organisées, la réponse au problème pour toute instance de cette famille.

Il convient d'expliquer certains termes contenus dans cette définition. Nous avons déjà constaté, au travers des exemples contenus dans les chapitres précédents, qu'un algorithme est *une procédure de résolution de problème*. Une *famille d'instances du problème* représente l'ensemble des configurations possibles du problème, alors qu'une *instance* est un cas particulier d'une configuration du problème. Ensuite, il est dit que les étapes doivent être en nombre fini, constructives, effectives, non-ambigües et organisées. *Effectives* signifie que les étapes doivent être suffisamment basiques pour être réalisables en un temps fini par *un homme utilisant du papier et un crayon*. Enfin, nous trouvons le fait qu'un algorithme doit être général puisqu'il doit être d'application pour toute instance d'une famille d'instances. Dès lors, toute procédure de résolution de problème n'étant pas générale ne peut pas être considérée comme étant un algorithme. Modeste appelle ces procédures des *algorithmes-instanciés*. C'est notamment le cas de la recette de cuisine qui est souvent citée comme exemple pour illustrer la notion d'algorithme, alors qu'elle n'est valable que pour un plat particulier. En outre, un algorithme doit comporter des entrées et des sorties. En conséquence, si une procédure n'en comporte pas, elle ne peut pas porter le titre d'algorithme. C'est notamment le cas du problème des anniversaires rencontré en probabilités dans le document « ressources ». En effet, pour cet exemple, il n'y a aucune entrée puisque les anniversaires sont générés au hasard au sein de l'algorithme.

Par la définition d'algorithme donnée, nous constatons également qu'aucune référence explicite n'est faite à la programmation. Un algorithme a donc une existence propre par rapport aux différents langages de programmation. Il est donc tout à fait envisageable de travailler sur l'algorithmique sans avoir recours à l'informatique. D'ailleurs, dans ses travaux, Modeste ne s'intéresse en aucun cas aux différents langages de programmation existants. Il n'est pas le seul chercheur à négliger cet aspect informatique. Il cite Lovász (1988) qui reproche l'utilisation trop rapide de l'ordinateur lorsqu'il est question d'étudier des algorithmes.

III.2 Quelques aspects de l'algorithme

Toujours en comparant différentes définitions de la notion d'algorithme ainsi que divers exemples, Modeste (2012) relève des points essentiels concernant les algorithmes. Il regroupe ces points essentiels en cinq groupes qu'il nomme les *aspects de l'algorithme*. Ces cinq aspects sont les aspects *problème*, *effectivité*, *preuve*, *complexité* et *modèles théoriques*.

1. **L'aspect problème :** un algorithme apporte la réponse à une question, à un problème. Il est général car il doit fonctionner pour toute instance d'une famille d'instances. Dans cet aspect, nous trouvons également le fait qu'un algorithme doit comporter des entrées et des sorties.

2. **L'aspect effectivité :** un algorithme s'exécute en un nombre fini d'étapes sur des données finies. De plus, il doit pouvoir être mis sous la forme d'un programme dans le but d'être mis en œuvre par un ordinateur ou une machine.

3. **L'aspect preuve :** un algorithme doit fournir le résultat attendu au problème pour lequel il a été conçu et cela, quelle que soit l'instance qui y a été entrée. Le fait de prouver que l'algorithme fournit le bon résultat s'appelle la *correction*.
   En outre, ce résultat doit être atteint en un nombre fini d'étapes. Prouver qu'un algorithme se termine en un nombre fini d'étapes, c'est en prouver la *terminaison*.
   Bien évidemment, il ne s'agit en aucun cas de tester l'algorithme sur quelques exemples pour voir s'il fournit bien le résultat escompté en un nombre fini d'étapes.

4. **L'aspect complexité :** la notion de complexité est une notion mathématique spécifique à l'algorithmique. Elle se calcule en fonction de l'ampleur des données entrées dans l'algorithme. De plus, elle peut être mesurée en temps ou en taille de mémoire.
   Le but d'étudier la complexité des algorithmes est de pouvoir classer différents algorithmes répondant au même problème, dans le but de trouver l'algorithme qui est le plus efficace. Par ce point, nous en arrivons à devoir chercher un algorithme optimal pour un problème donné.

5. **L'aspect modèles théoriques :** cet aspect se base sur des modèles théoriques tels que les machines de Turing ou les fonctions récursives de Kleene. Ces différents modèles peuvent être considérés comme des définitions plus théoriques d'« algorithme ». Grâce à eux, il est possible de constater que tous les problèmes ne sont pas résolubles algorithmiquement. Pour ceux qui le sont, ces modèles permettent de les classer en fonction de leurs complexités respectives.

Comme nous l'avons souligné plus haut, l'algorithme n'est pas simplement un outil de résolution de problèmes, il est également un objet des mathématiques. Certains aspects relevés par Modeste font référence à l'algorithme en tant qu'outil, tandis que d'autres y font référence en tant qu'objet. Tout ce qui touche au bon fonctionnement d'un algorithme et à son étude se rattache au côté objet. Ce sont les aspects « preuve », « complexité » et « modèles théoriques » qui sont concernés. De l'autre côté, tout ce qui concerne l'utilisation d'un algorithme pour résoudre un problème se rattache au côté outil. Il s'agit là des aspects « problème » et « effectivité ».
Le tableau ci-dessous résume les différents aspects de l'algorithme classés selon la dualité outil-objet.

Outil	Objet
Problème	Preuve
Effectivité	Complexité
	Modèles théoriques

Afin d'illustrer ces différents aspects, nous allons nous servir d'un exemple. L'exemple que nous avons choisi est celui de la dichotomie proposé par l'IREM de Grenoble. Nous avons déjà présenté cet exemple dans le chapitre II. La partie du document de cet IREM relative au problème de la dichotomie se trouve à l'annexe A. Dans cet exemple, quatre des cinq aspects sont travaillés : les aspects « effectivité », « problème », « preuve » et« complexité ». Modeste décrit où ces différents aspects interviennent dans ce problème. Sa description est présentée à la FIGURE III.1. Les aspects sont classés en deux catégories, suivant qu'ils se réfèrent à l'algorithme en tant qu'outil ou qu'ils s'y réfèrent en tant qu'objet.

OUTIL	
Effectivité	- programmation : présence des algorithmes sous forme très proche d'un langage de programmation et de programmes Python - ordinateur/machine : idem, questions d'erreurs d'arrondis - opérateur : recherche de la stratégie « optimum » pour la recherche du nombre - finitude : Justification de la terminaison.
Problème	- entrée/sortie : explicitement précisées pour chaque algorithme « Données » et « Résultat » mais un amalgame est fait entre données d'un programme et entrées d'un algorithme - instance : recherches des instances les plus "coûteuses" pour la recherche dichotomique - Résout un problème pour toute instance : présence de la preuve de correction de l'algorithme
OBJET	
Preuve	- Preuve de terminaison : proposée pour la recherche de zéro d'une fonction - Preuve de correction : proposée pour la recherche de zéro d'une fonction - Invariant : invariant de la recherche de zéro d'une fonction
Complexité	- Au pire : étude de la complexité de la recherche dichotomique
Modèles théoriques	×

FIGURE III.1 – Aspects présents dans le problème de la dichotomie

III.3 Analyse de la situation au lycée

Sur la base des différents aspects développés au paragraphe précédent, Modeste analyse les programmes de mathématiques du lycée, le document « ressources » en algorithmique pour la seconde, des ressources mises en ligne par les IREM et des manuels du lycée, dans le but d'évaluer quels aspects sont travaillés et dans quelle proportion ils le sont. Il s'intéresse également aux différents types d'algorithmes en jeu dans ces documents. Il n'analyse en aucun cas les pratiques enseignantes.
Nous présentons des synthèses de ses analyses.

III.3.1 Programmes et document « ressources »

Dans les programmes, il est recommandé d'étudier l'algorithme en tant qu'outil et non en tant qu'objet. En effet, les objectifs du lycée consistent essentiellement à programmer sur ordinateur ou sur papier. D'ailleurs, un amalgame entre les termes « algorithme » et « programme » est présent dans les programmes du lycée car ces deux termes ne sont pas définis et ne sont pas clairement distingués.

Intéressons-nous à la présence des différents aspects. Les aspects « preuve » et « complexité » ne font jamais l'objet de tâches spécifiques dans les programmes et le document « ressources ». L'aspect « modèles théoriques » n'est jamais abordé non plus, mais Modeste juge qu'il n'est pas nécessaire de l'aborder dans des programmes pour le lycée, contrairement aux aspects « preuve » et « complexité » qui apportent du sens à la notion d'algorithme. L'aspect « effectivité » est assez présent. Enfin, l'aspect « problème » l'est aussi, mais nettement moins.
En outre, l'aspect « problème » n'est pas bien représenté car généralement, ce ne sont pas des classes de problèmes qui sont considérées. Dans ces cas-là, il faut travailler sur des algorithmes s'appliquant sur un exemple donné et non sur une famille d'instances. Modeste nomme ce type d'algorithmes des *algorithmes-instanciés*. Un exemple d'algorithme-instancié est celui décrit à la FIGURE III.2. Nous l'avons déjà rencontré dans le chapitre II, lors de la présentation du document « ressources » en algorithmique.

```
Mettre 5000 dans S
Mettre 0 dans N
Effacer l'écran
Tant que S est strictement inférieur à 8000
        Remplacer S par S*1,02
        Augmenter N de 1
        Afficher N et S
Fin du Tant Que
```

FIGURE III.2 – Un algorithme-instancié

De plus, certains algorithmes rencontrés dans ces documents n'en sont pas en réalité. C'est le cas des programmes de *modélisation-simulation* qui permettent de simuler un phénomène. En effet, ceux-ci ne résolvent pas de problèmes et n'ont en général aucune entrée.
Un exemple de ce type d'« algorithmes » est celui présenté à la FIGURE III.3. Il s'agit de répéter le lancer d'un dé jusqu'à ce qu'une certaine condition soit vérifiée.

La seule exception notable au sujet des programmes concerne le programme de la spécialité ISN (Informatique et Sciences du Numérique) que nous avons présenté au chapitre précédent.
Dans ce programme, les notions d'« algorithme » et de « programme » sont clairement distinguées. D'ailleurs, il est demandé de commencer par présenter simultanément ces deux concepts pour ensuite les distinguer. De

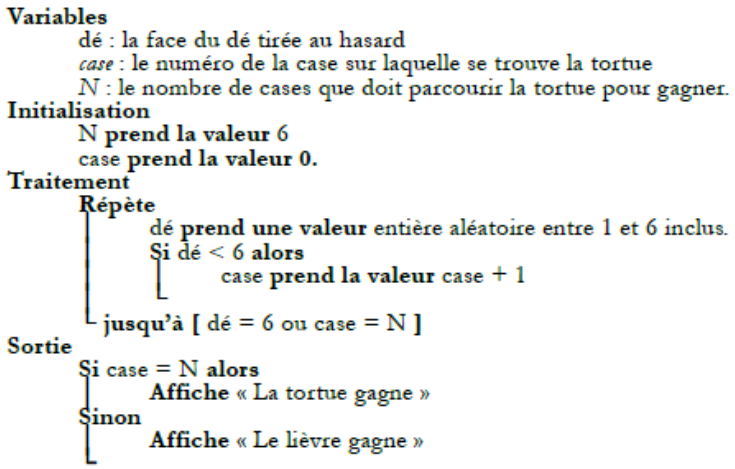


FIGURE III.3 – Un algorithme qui ne résout pas un problème

plus, une définition acceptable d'« algorithme » est donnée, évitant ainsi tout amalgame avec le terme « programme ».
En outre, il est clairement demandé de discuter de l'efficacité d'un algorithme, ce qui n'était pas demandé dans les autres programmes. Il y est également précisé qu'il n'est pas nécessaire de programmer les algorithmes créés.

En résumé, dans ce programme, nous trouvons, en plus des aspects « effectivité » et « problème » trouvés dans les autres programmes, l'aspect « complexité » qui apparaît très clairement lorsqu'il est demandé de comparer différents algorithmes entre eux pour discuter de leurs efficacités.

III.3.2 Ressources des IREM

Dans sa thèse, Modeste décide de s'intéresser à une trentaine de ressources disponibles sur le site des IREM. Ces documents sont de types variés : documents pour la formation des enseignants, documents pour la classe,...
Le bilan de cette analyse est assez similaire à celui effectué pour l'analyse des programmes du lycée. En effet, une fois encore, plus d'intérêt est accordé à la programmation qu'à l'algorithmique à proprement parler. Nous retrouvons également, à plusieurs reprises, le même amalgame entre « programme » et « algorithme ».

Concernant les aspects, ce sont les aspects relatifs au côté outil de l'algorithme qui prédominent une fois de plus. Trois types de documents sont d'ailleurs distingués :

1. ceux où l'algorithme est uniquement outil,
2. ceux où l'algorithme est à la fois outil et objet,
3. ceux qui ne traitent pas réellement d'algorithmes (écriture de programmes,...).

Les documents dans lesquels l'algorithme est vu à la fois en tant qu'objet et qu'outil sont tous, à une exception près, à destination des enseignants.

Modeste ne présentant pas d'exemples pour l'analyse de l'ensemble des documents, nous nous contentons donc de résumer ses principales constatations en ce qui concerne les différents aspects de l'algorithme.
Comme dans les programmes et le document « ressources », ce sont les aspects « effectivité » et « problème » qui sont principalement travaillés dans l'ensemble des ressources étudiées.
L'aspect « effectivité » est notamment travaillé, même implicitement, dans des tâches consistant à écrire un programme informatique ou à exécuter un programme ou un algorithme.
Concernant l'aspect « problème », nous trouvons à nouveau des programmes de modélisation-simulation et des algorithmes-instanciés. En outre, les entrées et les sorties ne sont pas toujours clairement explicitées.
Seuls sept documents traitent de l'aspect « preuve » et tous sont destinés aux enseignants et non à la classe. Dans deux documents, la preuve d'un algorithme est la question centrale en jeu, tandis que dans les autres, le lien avec la preuve est évoqué, mais la preuve d'un algorithme ne constitue pas forcément un objectif principal du document.
L'aspect « complexité » n'est présent que dans cinq documents dont quatre sont destinés aux enseignants. Il s'agit, par exemple, de comparer plusieurs algorithmes entre eux. Le seul document destiné à la classe qui comporte des éléments relatifs à la complexité est un problème consistant à discuter de la complexité du tri à bulles et à comparer le tri à bulle avec le tri rapide. Cependant, il s'agit simplement de compter, sur la base d'un exemple, le nombre d'étapes effectuées par l'algorithme.
L'aspect « modèle théorique » n'est abordé dans aucun document.

III.3.3 Manuels du lycée

Les manuels étudiés sont les manuels de seconde, premières ES-L, terminales ES-L, première S et terminale S des collections *Indices* et *Transmath.* Rappelons que les programmes de mathématiques de première et de terminale des séries ES et L sont les mêmes.

Dans les manuels des deux collections, la grande majorité des algorithmes sont des algorithmes peu adaptés pour permettre de les travailler en tant qu'objets. Ce sont notamment des algorithmes basés sur des formules mathématiques (résolution d'équations du second degré, formules de géométrie, formules de sommes,...) et des simulations aléatoires.

Les principales questions posées aux élèves, toutes années confondues, sont des questions d'écriture d'algorithmes, d'écriture de programmes à partir d'algorithmes, de modifications et de compréhension d'algorithmes et d'exécution d'algorithmes et de programmes. L'activité souvent demandée aux élèves est du type : problème $\rightarrow$ écriture d'un algorithme $\rightarrow$ écriture d'un programme $\rightarrow$ exécution.

En ce qui concerne les aspects présents dans les deux collections, seuls les aspects « effectivité » et « problème » sont présents, comme le laissaient présumer les principales questions posées aux élèves. Les trois autres aspects n'apparaissent tout simplement pas dans les différents manuels. L'algorithme est donc toujours vu en tant qu'outil et n'est alors jamais étudié en tant qu'objet des mathématiques.
En outre, à nouveau, en ce qui concerne l'aspect « problème », nous retrouvons des programmes de modélisation-simulation et des algorithmes-instanciés.
Pour vérifier qu'un algorithme fonctionne, les manuels se contentent de quelques exemples à titre de preuve, mais cela ne constitue pas une preuve valide.

III.3.4 Conclusion

Après avoir synthétisé les analyses de Modeste en ce qui concerne les différents programmes de mathématiques du lycée, le document « ressources » en algorithmique, les ressources en algorithmique mises en ligne par les IREM et certains manuels du lycée, nous pouvons tirer quelques conclusions.

Tout d'abord, la notion d'« algorithme » n'est pas correctement définie

dans ces documents. De même, un amalgame existe entre les notions d'« algorithme » et de « programme ».

Ensuite, les différents aspects de l'algorithme ne sont pas travaillés de manière équitable. En effet, ce sont essentiellement les aspects « effectivité » et « problème » qui sont travaillés. L'aspect « effectivité » est parfois sous-jacent à des tâches dont le but est d'écrire ou d'exécuter un programme informatique. Les algorithmes ne sont pas toujours réellement la solution à un problème général : nous trouvons des programmes de modélisation-simulation et des algorithmes-instanciés.
Les aspects « preuve » et « complexité » ne sont qu'occasionnellement évoqués. Dans des exercices, souvent, il ne s'agit pas du but principal de l'exercice.
L'aspect « modèles théoriques » n'est jamais évoqué, mais ce n'était pas l'aspect le plus attendu dans un enseignement de mathématiques au lycée.

L'algorithme au lycée est donc essentiellement vu en tant qu'outil pour la résolution de problèmes et non en tant qu'objet des mathématiques.

III.4 Des situations en vue d'une amélioration

Nous l'avons constaté, dans l'enseignement de l'algorithmique au lycée, seuls deux aspects sont travaillés, au dépend des trois autres. Il s'agit des aspects « effectivité » et « problème ». Les aspects « preuve », « complexité » et « modèles théoriques » sont totalement négligés. Or, ce sont ces aspects qui permettent d'étudier l'algorithme en tant qu'objet des mathématiques.
Sur la base de ce constat, Modeste (et al. 2010, 2012) et Aldon et al. (2012), didacticiens des mathématiques français, proposent de travailler sur des situations mettant en jeu des algorithmes pour lesquels il est envisageable d'étudier leurs preuves et leurs complexités dans un enseignement des mathématiques au lycée. Désormais, nous ne discutons plus de l'aspect « modèles théoriques » car il semble plus délicat à intégrer au lycée.

III.4.1 Un problème de fausses pièces

Énoncé du problème

Un premier problème pour discuter des aspects « preuve » et « complexité » d'un algorithme est le *problème des pesées* proposé par Modeste. La FIGURE III.4 en précise son énoncé.

Dans un ensemble de pièces indiscernables à la vue et au toucher, se trouvent des fausses pièces. Les vraies pièces pèsent toutes le même poids, les fausses aussi mais leur poids est différent de celui des vraies. À l'aide d'une balance Roberval à deux plateaux et sans poids, peut-on retrouver les fausses pièces ? Si oui, quelle est la méthode qui permet de les retrouver en effectuant le moins de pesées possible ?

FIGURE III.4 – Le problème des pesées

Plusieurs variantes de ce problème existent. La FIGURE III.5 illustre les différentes variables sur lesquelles il est possible de jouer. À ce jour, toutes les variantes de ce problème n'ont pas encore été résolues.

- **Poids des fausses pièces :**
  - **On sait au départ que les fausses pièces sont plus lourdes que les vraies (ou plus légères).**
  - **On ne sait rien.**
- **Le nombre de fausses pièces :**
  - **Fixé à l'avance.**
  - **Compris dans un intervalle donné.**
  - **Non connu à l'avance.**

FIGURE III.5 – Variables du problème des pesées

Ce problème relève bien de l'algorithmique car le but est d'élaborer un algorithme permettant de trouver les fausses pièces. L'objectif par la suite est double : il s'agit, d'une part, de montrer que l'algorithme va toujours trouver les fausses pièces (preuve de l'algorithme) et, d'autre part, de montrer que l'algorithme est optimal, c'est-à-dire que le nombre de pesées est minimum. Le critère retenu pour l'optimalité est la complexité dans le pire des cas. Il s'agit alors de minimiser le nombre d'étapes (le nombre de pesées dans ce cas) qu'effectue l'algorithme pour des instances d'une taille de donnée fixée (un nombre de pièces fixé).
Le but de cette situation est donc d'étudier l'algorithme en tant qu'objet des mathématiques et non plus en tant que simple outil de résolution de problèmes.

Comme les variantes du problème sont nombreuses, Modeste se limite à l'étude de quelques cas, à savoir :
- 1 fausse pièce, plus lourde ($P_{1,+}$),
- 1 fausse pièce, plus lourde ou plus légère ($P_{1,+/-}$),

- 0 ou 1 fausse pièce, plus lourde ($P_{0/1,+}$),
- 0 ou 1 fausse pièce, plus lourde ou plus légère ($P_{0/1,+/-}$).

Le cas $P_{1,+}$

Envisageons uniquement le cas le plus simple, à savoir $P_{1,+}$. Il s'agit bien du cas le plus simple car le nombre de fausses pièces est connu ainsi que leur poids par rapport aux pièces normales. Notons que le problème dans lequel il n'y aurait qu'une fausse pièce plus légère que les vraies est similaire à $P_{1,+}$ et n'est donc pas abordé.
Pour le problème $P_{1,+}$, plusieurs stratégies sont envisageables.

1. La stratégie expérimentale.
   Pour un tel problème, il est parfois intéressant de travailler sur des petits cas afin de voir un peu ce qui se produit pour ensuite formuler une conjecture et trouver une stratégie optimale pour ces cas.

2. La stratégie de dichotomie.
   En partant de la stratégie précédente, il est possible d'arriver à la constatation qu'une pesée consiste à comparer deux ensembles de pièces de même effectif pour savoir lequel est le plus lourd.
   Cela permet ainsi de garder $\lfloor \frac{n}{2} \rfloor$ pièces, si n est le nombre de pièces avant la pesée. Nous avons pris $\lfloor \frac{n}{2} \rfloor$ car si le nombre de pièces avant la pesée est pair, nous aurons $\frac{n}{2}$ pièces et s'il est impair, nous laissons une pièce de côté et donc le nombre de pièces après la pesée est $\frac{n-1}{2}$.
   Si, dans le cas où le nombre de pièces est impair, la balance est en équilibre lors de la pesée, c'est que la pièce mise de côté est la fausse pièce.
   Cette méthode nous donne un algorithme en $\lfloor \log_2 n \rfloor$ pesées dans le pire des cas.

3. La stratégie de trichotomie.
   En confrontant les stratégies expérimentale et dichotomique, nous nous apercevons que la stratégie de dichotomie n'est pas optimale lorsque le nombre de pièces est un multiple de 3. Par exemple, pour 9 pièces, la stratégie de dichotomie va amener à dire qu'il faut 3 pesées, alors qu'avec la stratégie expérimentale, nous constatons que 2 pesées suffisent. En réalité, lorsque nous effectuons une pesée, nous obtenons aussi de l'information sur les pièces non pesées.

La stratégie de trichotomie consiste alors à diviser le nombre de pièces en trois parties : deux tas contenant autant de pièces que nous comparons à l'aide de la balance et le dernier tas que nous ne pesons pas. Le dernier tas ne doit différer en nombre de pièces que d'une unité par rapport à chaque tas sur la balance, pour garder des tas de pièces équitables en nombres.
Cet algorithme est en $\lceil \log_3 n \rceil$ pesées dans le pire des cas. Il est possible de montrer qu'il est optimal pour la complexité dans le pire des cas. Le lecteur intéressé pourra se référer à Modeste (2012) pour la démonstration.

4. La stratégie par étalon.
   Il s'agit de comparer un ensemble de pièces inconnues avec un ensemble de pièces reconnues vraies. Après comparaison, si ces pièces sont reconnues vraies, nous les ajoutons au tas de vraies.

5. La stratégie de passage par un autre problème.
   Nous nous ramenons à des problèmes qui ont déjà été traités précédemment ou à des problèmes plus simples.

Expérimentation du problème

Modeste a proposé cette activité en tant que situation de recherche en classe (SiRC) dans deux classes : un cours de mathématiques de la formation des professeurs des écoles en deuxième année (PE_2) de l'Institut Universitaire de Formation des Maîtres (Iufm) de Créteil et une option proposée à l'université Joseph Fourier aux étudiants de première et deuxième années des licences scientifiques.

Les étudiants ont utilisé majoritairement la stratégie de dichotomie pour résoudre ce problème. La stratégie expérimentale a également été beaucoup utilisée dans le but de trouver la méthode optimale, mais également pour évaluer la complexité des différentes méthodes. La stratégie de trichotomie est un peu moins présente. La stratégie de passage par un autre problème a été utilisée par des étudiants qui ne se sont pas limités à $P_{1,+}$. La stratégie par étalon a été très peu utilisée.

Les étudiants se sont assez peu intéressés à la preuve des différents algorithmes. Ils ont cependant étudié tous les cas envisageables pour la fausse pièce ou ont essayé de se ramener à des cas déjà étudiés. Lors

d'échanges oraux, il s'est avéré que la preuve était présente, notamment quand les étudiants expliquaient la manière dont ils avaient conçu leurs algorithmes.
En ce qui concerne l'optimalité des algorithmes, les élèves ont comparé les différentes stratégies sur des exemples et des contre-exemples.
Pour la complexité (le nombre de pesées), des étudiants ont tenté de formuler des hypothèses sur base de la stratégie expérimentale. Certains ont remarqué que le nombre de pesées dépendait du nombre de pièces.

III.4.2 Des situations pour la preuve et la complexité

Aldon et al. (2012) proposent quelques algorithmes intéressants à travailler en classe afin de les étudier dans un cadre mathématique. Les deux types d'activités proposés sont la preuve et la complexité d'algorithmes. Nous nous situons donc du côté des aspects qui permettent de travailler l'algorithme en tant qu'objet. Présentons donc un exemple permettant de travailler la preuve d'un algorithme et un autre permettant d'en travailler la complexité.

Preuve d'un algorithme

Pour la preuve d'un algorithme, Aldon propose un algorithme de recherche de fractions égyptiennes. Il s'agit d'écrire un rationnel de l'intervalle $]0,1[$ sous la forme d'une somme d'inverses de naturels tous distincts. En termes de mathématiques, il faut trouver une suite finie de naturels strictement croissante r_1, r_2,..., r_s telle que $\frac{a}{b} = \frac{1}{r_1} + \frac{1}{r_2} + ... + \frac{1}{r_s}$, avec $0 < a < b$. Un algorithme pour y parvenir est donné à la FIGURE III.6.

La preuve de cet algorithme s'effectue en deux étapes : nous devons vérifier que l'algorithme se termine en un nombre fini d'étapes (terminaison) et qu'il produit le résultat escompté (correction). Pour prouver la terminaison de cet algorithme, nous montrons que la valeur de la variable u diminue strictement à chaque nouveau passage dans la boucle tout en restant positive. Comme r représente la partie entière supérieure (*ceil*) de $\frac{v}{u}$, nous disposons de la relation suivante :

$$r - 1 < \frac{v}{u} \leqslant r.$$

```
Algorithme egypte
Entrée : a entier, b entier tels que 0 < a < b
Sortie : une liste [r_1,...,r_s] telle que
a/b = 1/r_1+...+1/r_s
Corps :
  initialiser u := a, v := b, L := []
                    // ici, la notation [] désigne
                    la liste vide
  tant que u ≠ 0 faire
    r := ceil(v/u)
    u := u * r – v
                    // ici, l'étoile * représente la
                     multiplication...
    v := v * r
    ajouter r à la liste L
  fin du tant que
  renvoyer L
```

FIGURE III.6 – Algorithme de recherche des fractions égyptiennes

En supprimant le dénominateur (positif) et en simplifiant l'expression, nous obtenons :

$$0 \leqslant ur - v < u.$$

Or, $ur - v$ est la nouvelle valeur de u après un passage dans la boucle. La valeur de la variable u est donc strictement décroissante et positive. Ainsi, nous créons une suite de naturels strictement décroissante. La variable u va donc atteindre la valeur 0 et la condition du **tant que** ne sera plus vérifiée. Nous sortons donc de la boucle à un moment donné. L'algorithme se termine donc bien.

Pour prouver la correction de cet algorithme, nous devons montrer que l'algorithme donne le résultat attendu. Pour cela, nous montrons qu'à chaque passage dans la boucle, l'égalité suivante est vérifiée :

$$\frac{a}{b} = \frac{u}{v} + \sum_{s \in L} \frac{1}{s}, \tag{III.1}$$

où L représente la liste des naturels à obtenir par l'algorithme.
Nous montrons cela par récurrence. Avant d'entrer dans la boucle, la liste L est vide et le couple (u, v) est égal au couple (a, b). L'égalité III.1 est donc vérifiée.
Supposons que cette égalité soit vraie au début d'un passage dans la boucle

et montrons qu'elle l'est encore après le passage dans la boucle. Lors d'un passage dans la boucle, les variables sont remplacées comme suit :

- u devient $u' = ur - v$,
- v devient $v' = vr$,
- L devient M, liste obtenue en ajoutant l'élément r à la liste L.

Nous obtenons donc les égalités suivantes :

$$\frac{u'}{v'} + \sum_{s \in M} \frac{1}{s} = \frac{ur - v}{vr} + \frac{1}{r} + \sum_{s \in L} \frac{1}{s} = \frac{u}{v} + \sum_{s \in L} \frac{1}{s} = \frac{a}{b}.$$

À la sortie de la boucle, comme u est nul, l'expression III.1 devient $\frac{a}{b} = \sum_{s \in L} \frac{1}{s}$.

Pour en finir avec cette preuve, il reste à montrer que la suite de naturels de la liste L est bien strictement croissante. Cela se montre facilement en comparant les valeurs des variables lors d'un nouveau passage dans la boucle.

Complexité d'un algorithme

Aldon et al. (2012) exposent deux algorithmes permettant de calculer les nombres de Fibonacci. Le premier algorithme est présenté à la FIGURE III.7.

```
Algorithme Fib_rec
Entrée : n entier naturel
Sortie : f(n) entier naturel, le n^e nombre de Fibonacci
Corps :
  si n < 2 renvoyer 1
  sinon renvoyer Fib_rec(n–1) + Fib_rec(n–2)
  fin du si
```

FIGURE III.7 – Algorithme de calcul des nombres de Fibonacci (1)

Si nous implémentons cet algorithme et que nous le testons pour $n > 30$, l'ordinateur ne donne pas de réponse. Une étude de la complexité de cet algorithme permet d'expliquer pourquoi l'ordinateur met un temps non négligeable pour calculer les nombres de Fibonacci pour $n > 30$.

Les seules opérations effectuées par l'algorithme sont des additions et des appels récursifs à lui-même. Notons $S(n)$, le nombre de sommes pour calculer $f(n)$ et $A(n)$, le nombre d'appels récursifs pour son calcul.
Nous obtenons les égalités suivantes :

$$\begin{aligned} S(n) &= S(n-1) + S(n-2) + 1, \\ A(n) &= A(n-1) + A(n-2) + 2. \end{aligned}$$

À partir de là, il est possible de montrer que le nombre d'additions et le nombre d'appels récursifs augmentent de façon exponentielle avec la valeur de n. Nous comprenons donc mieux pourquoi l'exécution prenait du temps pour $n > 30$.
Cependant, il existe un algorithme résolvant le même problème mais ayant une meilleure complexité. Cet algorithme est donné à la FIGURE III.8.

```
Algorithme Fib_iter
Entrée : n entier naturel
Sortie : f(n) entier naturel, le n^e nombre de
Fibonacci
Variables : k, a, b, c entiers
Corps :
  a := 1
  b := 1
  pour k de 1 à n – 1 faire
        c := a
        a := b
        b := a + c
  renvoyer b
```

FIGURE III.8 – Algorithme de calcul des nombres de Fibonacci (2)

À nouveau, déterminons la complexité de cet algorithme. Ce dernier ne comporte que des affectations $A(n)$ et des sommes $S(n)$. Pour $n > 1$, nous avons les égalités suivantes :

$$\begin{aligned} A(n) &= 2 + 3(n-1), \\ S(n) &= n - 1. \end{aligned}$$

Nous constatons donc que les nombres d'affectations et de sommes sont, cette fois-ci, linéaires en n. La complexité de cet algorithme est donc de

l'ordre de n et il est alors possible de l'exécuter pour de grandes valeurs de n, contrairement au précédent.

III.4.3 Conclusion

Au travers de ces différentes situations, nous avons remarqué qu'il était envisageable de travailler des éléments et des compétences des mathématiques à l'aide d'algorithmes pourtant simples. En effet, nous avons vu qu'il était possible de travailler sur les logarithmes, les exponentielles, les suites de naturels,... Cependant, ce n'est pas tout : il est également possible de travailler des compétences transversales des mathématiques à l'aide des algorithmes. Nous avons notamment vu que les raisonnements logiques et l'argumentation pouvaient être travaillés, mais aussi la preuve. La preuve est un aspect primordial de l'activité du mathématicien. Dans nos exemples, nous avons constaté que la preuve pouvait être travaillée à l'aide d'exemples et de contre-exemples, d'énumérations de cas, et même en ayant recours à la preuve par récurrence.
Nous avons donc illustré des exemples accessibles à des élèves du secondaire qui permettent de travailler l'algorithme non plus seulement en tant qu'outil, mais également en tant qu'objet des mathématiques.

III.5 Bilan

Dans ce chapitre, nous avons effectué la synthèse de travaux de didacticiens des mathématiques français.
Modeste, en comparant des définitions et des exemples d'algorithmes a repéré des points essentiels en ce qui concerne l'algorithme qu'il a regroupés en cinq catégories, nommées les aspects de l'algorithme. Ces aspects sont l'effectivité, la résolution de problèmes, la preuve, la complexité et les modèles théoriques. Il a également constaté que certains de ces aspects utilisaient l'algorithme en tant qu'outil pour résoudre un problème (effectivité et problème) tandis que d'autres étudiaient l'algorithme en tant qu'objet des mathématiques (preuve, complexité et modèles théoriques).

En analysant sur la base de ces aspects les différents documents produits en France sur l'algorithmique, Modeste constate que les algorithmes sont exclusivement vus comme des outils pour la résolution de problèmes au lycée. En effet, seuls les aspects « effectivité » et « problème » sont travaillés. L'as-

pect « effectivité » est sous-jacent à tout ce qui touche à la programmation. L'aspect « problème » n'est, quant à lui, pas toujours correctement travaillé car certains algorithmes ne sont pas généraux et d'autres encore ne résolvent pas nécessairement un problème ou ne comportent pas d'entrées ou de sorties.

Face à ce constat et convaincus du fait qu'il est possible de travailler des éléments de mathématiques grâce aux algorithmes, Modeste et Aldon proposent tous deux des situations autour de l'algorithme pour lesquelles il est possible de travailler l'algorithme en tant qu'objet des mathématiques. Ce sont donc les aspects « preuve » et « complexité » qui sont mis en avant dans leurs propositions de situations.
Les algorithmes qu'ils proposent ne sont pas forcément compliqués, mais il est possible d'effectuer un travail mathématique dessus, travail accessible - du moins en partie - à des élèves du secondaire supérieur.
Les différentes situations qu'ils proposent permettent de travailler des points spécifiques des mathématiques ainsi que des compétences transversales. Comme points spécifiques, nous trouvons un travail sur les logarithmes, les exponentielles, les suites de naturels,... Les compétences transversales qu'il est possible de travailler sont des compétences telles que le raisonnement logique, l'argumentation ainsi que la preuve.
En ne travaillant pas l'algorithme en tant qu'objet, nous négligerions donc des facettes de l'algorithme qu'il serait regrettable de négliger.

Notre cadre théorique posé, nous expliquons, au chapitre suivant, notre problématique de recherche ainsi que la méthodologie associée.

Chapitre IV

Problématique de recherche et méthodologie associée

Dans ce chapitre, nous prenons appui sur les analyses menées aux chapitres II et III pour préciser notre problématique de recherche et nous présentons la méthodologie associée.

IV.1 Problématique de recherche

L'algorithmique n'est pas présente dans les programmes actuels de mathématiques de l'enseignement secondaire en Belgique. Il n'y aurait donc aucun intérêt à l'étudier dans le cadre d'un mémoire en didactique des mathématiques. Cependant, il suffit de quelques exemples pour se convaincre du contraire, comme nous avons pu le constater dans les chapitres précédents.

Dans notre premier chapitre, en regardant un ancien manuel du secondaire belge et des cours universitaires de mathématiques, nous avons constaté que les mathématiques entretenaient des liens particuliers avec l'algorithmique. Effectivement, certains problèmes de mathématiques ne sont tout simplement pas résolubles « à la main » et le recours aux algorithmes est donc plus que nécessaire pour apporter une méthode de résolution à ces différents problèmes. Nous en avons donc tiré la conclusion qu'il était tout à fait envisageable de travailler des algorithmes dans un cours de mathématiques.
De ce constat est née la première version de notre problématique qui consistait à se poser la question de savoir, d'une part, quel genre de mathématiques il serait possible de travailler à l'aide de l'algorithmique

dans l'enseignement secondaire, et d'autre part, ce que pourrait apporter l'introduction d'éléments d'algorithmique dans un cours de mathématiques.

Notre problématique était assez vague et nous avons donc décidé de nous intéresser à la situation en France, ainsi qu'aux travaux de didacticiens français afin de la préciser. L'intérêt de s'intéresser à ce qui se passe en France est que des éléments d'algorithmique ont été introduits dans les programmes de mathématiques du lycée (secondaire supérieur belge) à partir de 2009.

En France, la commission Kahane propose, en 2000, d'introduire une part d'informatique dans les cours de mathématiques, notamment parce que le recours à l'informatique est actuellement inévitable dans la recherche mathématique et il peut alléger considérablement le travail du mathématicien. Une fois de plus, nous avons donc eu la confirmation que les mathématiques et l'informatique/algorithmique étaient liés.
Nous avons également regardé les programmes de mathématiques du lycée ainsi que d'autres ressources dans lesquelles une part d'algorithmique a été introduite dans le but de voir quels éléments d'algorithmique ont été intégrés et comment ils l'ont été.
Au final, nous avons constaté que les algorithmes étaient utilisés dans des contextes mathématiques pour automatiser certains calculs (calcul du milieu d'un segment, recherche d'extrema,...), mais que bien souvent, les seules questions posées concernaient l'exécution d'un algorithme ou d'un code, ou encore l'écriture d'un algorithme. En dehors de cette automatisation des calculs mathématiques et de la simulation d'événements aléatoires, nous n'avons donc pas réellement perçu, au travers de ces documents, ce que l'introduction d'éléments d'algorithmique pourrait apporter dans un cours de mathématiques.

Finalement, nous nous sommes intéressée à des travaux de didacticiens des mathématiques français. De ces travaux, nous avons extrait une première définition de la notion d'« algorithme ». Elle est donnée à la page 36.
En analysant ces travaux, nous nous sommes aperçue qu'il y avait des manquements concernant l'enseignement de l'algorithmique au lycée : en effet, l'algorithme n'est en général travaillé qu'en tant qu'outil pour la résolution de problèmes, alors qu'il est tout à fait envisageable de le travailler en tant qu'objet des mathématiques.

En considérant l'algorithme comme objet des mathématiques, nous avons vu qu'il était possible de travailler des compétences transversales telles que

la preuve, l'argumentation et la logique, avec un contexte initial autre que purement mathématique. Rappelons que l'enseignement belge préconise justement de travailler par compétences.

À partir de ce dernier constat, nous énonçons une nouvelle version de notre problématique : *comment travailler des compétences transversales telles que la preuve, l'argumentation et la logique avec des éléments d'algorithmique au sein d'un cours de mathématiques ?*
Pour mettre en évidence la possibilité de faire de l'algorithmique dans un cours de mathématiques, nous nous sommes donnée comme objectif de concevoir une séquence d'enseignement sur l'algorithmique ayant pour but de travailler des compétences transversales en mathématiques.
Nous expliquons au point suivant notre méthodologie pour y parvenir.

IV.2 Méthodologie associée

La méthodologie que nous avons envisagée est découpée en cinq points. Dans un premier temps, nous nous sommes intéressée à un cours universitaire belge traitant de l'algorithmique. Le cours que nous avons choisi est le cours de « Programmation et Algorithmique I », cours dispensé aux étudiants de première année en sciences mathématiques et informatiques et aux étudiants de deuxième année en sciences physiques à l'Université de Mons. Le titulaire de ce cours est Hadrien Mélot. Nous avons choisi ce cours pour plusieurs raisons. Tout d'abord, il contient une part non négligeable d'algorithmique comme l'indique l'intitulé du cours. De plus, nous avons choisi un cours dispensé à l'Université de Mons car de cette manière, nous avons pu avoir facilement accès aux documents mis à la disposition des étudiants ainsi qu'à leurs évaluations. Enfin, il s'agit du premier cours d'algorithmique que ces étudiants ont suivi et donc ce cours assied les bases de l'algorithmique, ce qui nous a d'autant plus intéressée puisque notre séquence de cours a dû contenir ces bases.
À partir de ce cours, nous avons regardé minutieusement chaque chapitre afin de voir quels aspects étaient travaillés dans les différents chapitres et dans quelle proportion ils l'étaient. Ensuite, nous avons analysé l'agencement des différents chapitres afin d'obtenir une idée de structure pour notre séquence de cours, notamment pour les différentes notions déjà rencontrées au chapitre II (affectation, structure conditionnelle, structure itérative,...).

Le deuxième point de notre méthodologie a consisté à nous intéresser à des évaluations proposées aux étudiants de première année en sciences mathématiques et informatiques. Les évaluations en question ont été l'écriture d'une démarche pour la résolution d'un problème, la preuve par récurrence d'un algorithme (uniquement pour les étudiants en informatique) et l'examen du cours de Programmation et Algorithmique I de janvier 2014. Notre but, avec ces évaluations, a été d'analyser les copies des étudiants afin de déterminer quelles ont été les difficultés qu'ils ont rencontrées vis-à-vis de l'algorithmique. Le fait de savoir où se situent les difficultés des étudiants nous a été utile lors de la conception de notre séquence de cours car ainsi, nous avons pu déterminer les points sur lesquels nous devions davantage nous attarder et que nous devions détailler plus en profondeur.
Finalement, nous avons récolté l'avis des étudiants de première année en mathématiques et en informatique quant au cours de Programmation et Algorithmique I pour avoir leur ressenti après l'examen, notamment sur les difficultés éprouvées lors de l'enseignement de ce cours et de l'examen.

Le troisième point de cette méthodologie a concerné l'élaboration de notre séquence de cours sur l'algorithmique. Nous l'avons conçue sur la base des deux points précédents de notre méthodologie. Le cours de Programmation et Algorithmique I nous a servi pour établir la structure de la séquence et pour définir les notions à étudier. Les difficultés des élèves nous ont été utiles, comme déjà précisé, pour savoir a priori ce qui était susceptible de poser problème aux élèves du secondaire.
Nous avons également conçu cette séquence de cours en prenant en considération les propos de Modeste et Aldon afin de ne pas tomber dans les « travers » de la France et de ne proposer que des activités mettant en jeu l'algorithme en tant qu'outil ou des activités liées à la programmation. Notre but a donc été d'étudier l'algorithme en tant qu'objet des mathématiques. Les deux manières proposées par ces deux didacticiens sont d'étudier la preuve et la complexité d'un algorithme. Cependant, étant donné que les élèves du secondaire belge n'ont aucune notion d'algorithmique, nous n'avons pas pu directement commencer notre séquence de cours par ces deux aspects. Il nous a fallu d'abord commencer par étudier les différentes structures composant un algorithme (affectation,...).

Le quatrième et avant-dernier point de cette méthodologie a tourné autour de l'expérimentation de notre séquence de cours conçue au troisième point. Notre séquence de cours rédigée, nous l'avons expérimentée dans différentes classes de différents niveaux du secondaire supérieur et de différents professeurs afin que les expérimentations ne soient pas toutes

identiques. Nous avons prévu la matière à étudier durant chaque heure de cours ainsi que notre analyse a priori avec les compétences visées. Ensuite, nous avons confronté notre analyse a priori avec les expérimentations. Nous avons aussi déterminé les différents points de cette séquence de cours qui ont semblé être problématiques pour les élèves.

Enfin, le dernier point de notre méthodologie a consisté en l'évaluation de la séquence de cours. Durant la dernière heure de notre expérimentation, nous avons proposé aux élèves deux questionnaires. Le premier consistant en une évaluation de la matière pour voir après l'expérimentation ce que les élèves en ont retenu et pour déterminer les points problématiques, pour voir s'il s'agit des mêmes points qui ont posé problème aux étudiants de l'université. Nous nous sommes également demandée si les compétences visées dans notre analyse a priori ont semblé être en voie d'acquisition chez les élèves. Le deuxième questionnaire a récolté l'avis des élèves sur la séquence de cours en ce qui concerne les points positifs et négatifs et les points à améliorer.

Chapitre V

Analyse d'un cours universitaire sur l'algorithmique

Dans ce chapitre, nous analysons un cours universitaire belge traitant de l'algorithmique en vue de la conception de notre séquence d'enseignement. Le but de cette analyse est double : d'une part, analyser les différents chapitres le composant afin de déterminer quels aspects sont travaillés et dans quelle proportion ils le sont, et d'autre part, établir la structure de ce cours.

V.1 Analyse des différents chapitres

Le cours que nous allons analyser est donc le cours de « Programmation et Algorithmique I » pour les raisons que nous avons évoquées au chapitre précédent. Pour rappel, ce cours est dispensé aux étudiants de première année en sciences mathématiques et informatiques et aux étudiants de deuxième année en sciences physiques à l'Université de Mons. Ce cours est divisé en 14 chapitres qui sont vus en 30 heures de cours.
Dans le cadre de ce travail, nous n'analysons que le syllabus mis à la disposition des étudiants par le titulaire du cours.

Dans un premier temps, nous allons analyser ces 14 chapitres en observant quels aspects sont travaillés dans chaque chapitre et dans quelle proportion ils le sont. Le but d'une telle analyse est que nous souhaitons travailler l'algorithme en tant qu'objet dans notre séquence de cours et non pas seulement en tant qu'outil. Nous allons donc regarder si l'algorithme est travaillé en tant qu'objet dans le cours de Programmation et

Algorithmique I. Comme il s'agit du premier cours de programmation et d'algorithmique que ces étudiants suivent, toutes les notions sont nouvelles pour eux et nous supposons donc que les premiers exemples pour chaque notion sont assez simples à comprendre, ce qui nous intéresse d'autant plus dans le cadre de notre séquence de cours pour des élèves du secondaire supérieur.

Nous détaillons ci-dessous notre analyse chapitre par chapitre, mais si le lecteur préfère d'emblée une analyse globale, il peut se reporter à la page 69 directement.

Chapitre 1 : introduction

Dans ce premier chapitre sont d'abord explicités les objectifs du cours : comprendre un algorithme, en concevoir un et écrire un programme. Les objectifs se situent donc plutôt du côté outil de l'algorithme puisque l'analyse d'un algorithme ne fait pas partie des objectifs. L'écriture d'un programme fait également partie intégrante de ces objectifs, ce qui nous semble tout à fait normal étant donné que l'intitulé du cours comporte le terme « programmation ».
Ensuite, les notions d'« algorithme » et de « programme » sont définies séparément. Les deux termes sont donc distingués. Pour la définition d'« algorithme », un schéma est d'abord donné afin d'en montrer le but. Il est illustré à la FIGURE V.1.

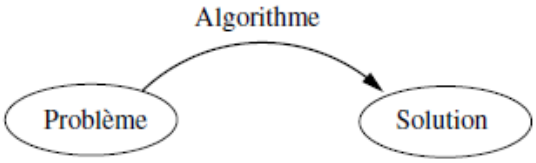


FIGURE V.1 – Schéma pour illustrer le but d'un algorithme

Ensuite, la définition d'« algorithme » est donnée : « un algorithme est une séquence d'étapes précises et non ambiguës pouvant être exécutées de façon automatique ». Dans cette définition, nous ne retrouvons pas le fait qu'un algorithme résout un problème, fait pourtant illustré avec le schéma. Le fait qu'un algorithme soit général et valable pour une famille d'instances du problème ne figure pas non plus dans cette définition. Il en est de même

pour le nombre fini d'étapes. Cette définition relève en réalité plutôt de l'aspect « effectivité » de l'algorithme.
Les notions d'entrée et de sortie sont également définies juste après la définition d'« algorithme ».
La définition de « programme » est la suivante : « un programme est une séquence d'instructions qui spécifie comment réaliser un calcul ou une tâche. Ils sont décrits à l'aide de langages de programmation ».
Des exemples d'algorithmes et de programmes sont également donnés. Les algorithmes donnés ne sont pas les exemples typiques souvent rencontrés, comme la recette de cuisine. Nous trouvons un algorithme pour afficher un message à l'utilisateur en fonction de l'heure et des algorithmes pour la recherche d'un mot dans un dictionnaire. Une méthode de dichotomie est déjà présentée. Ces algorithmes sont écrits en pseudo-code dans le but d'être implémentés assez facilement.
Les exemples de programmes sont donnés dans différents langages : C, Java et Python. Dans les chapitres ultérieurs, tous les programmes seront écrits en Python car c'est le langage choisi pour ce cours.

En résumé, l'aspect dominant de ce chapitre est l'aspect « effectivité » avec les définitions d'« algorithme » et de « programme ». Rappelons que l'aspect « effectivité » est sous-jacent dans tout ce qui traite de la programmation.
L'aspect « problème » est également un peu présent avec le schéma montré plus haut et avec les notions d'entrée et de sortie.

Chapitre 2 : types, variables et expressions

Dans ce chapitre sont vus, dans un premier temps, les différents types existants en Python (string, integer, float-point,...). La notion de variable est ensuite définie avec la notion d'assignation qui lui est liée. L'assignation permet de créer une nouvelle variable et de lui attribuer une valeur ou de mettre à jour la valeur d'une variable. Les opérations possibles en Python, ainsi que des exemples d'expressions sont finalement donnés.

En résumé, ce chapitre ne nous intéresse pas réellement puisqu'il s'agit avant tout d'un chapitre permettant de se familiariser avec les fonctionnalités du langage Python.

Chapitre 3 : fonctions

La notion de fonction en informatique est définie dans ce chapitre. Il

s'agit d'une « séquence d'instructions à laquelle on donne un nom ». Le chapitre détaille tout ce qui concerne les fonctions : en-tête, corps, intérêt, création de nouvelles fonctions,...

À nouveau, ce chapitre est uniquement relatif à la programmation puisqu'il s'agit de voir comment écrire des programmes lisibles, réutilisables,...

Chapitre 4 : instructions conditionnelles

Ce chapitre commence avec les expressions booléennes, expressions retournant soit vrai, soit faux. Les différents opérateurs sont alors vus : opérateurs de comparaisons (égalité de deux valeurs, valeur supérieure à une autre,...), opérateurs logiques (and, or, not),...
Ensuite, ce sont les instructions conditionnelles qui sont étudiées. Il s'agit de la structure « si...alors...sinon » déjà rencontrée dans nos précédents chapitres. Dans ce chapitre, la structure générale algorithmique n'est pas donnée, seule la structure en Python l'est. Cette structure est ensuite amplement discutée, mais toujours dans le cadre du langage Python. Nous trouvons, par exemple, une explication pour le cas où une condition doit être imbriquée dans une autre, ainsi que l'utilisation de fonctions booléennes. À chaque fois, des exemples sont donnés, mais toujours écrits en Python.
Finalement, un paragraphe est consacré aux erreurs d'approximation et à la représentation binaire.

En résumé, nous nous trouvons encore dans la programmation avec tout ce qui concerne la création de structures conditionnelles dans des programmes écrits en Python. Cependant, quelques aspects mathématiques sont présents à la fin du chapitre avec les erreurs d'approximation et la représentation binaire. Néanmoins, cela n'est pas directement en lien avec l'algorithmique, mais plutôt avec les ordinateurs.

Chapitre 5 : récursivité

Ce chapitre commence avec un rappel sur la preuve par récurrence, preuve déjà rencontrée par les étudiants dans le cours de *Mathématiques élémentaires*. Le cours de *Mathématiques élémentaires* est un cours obligatoire en première année que les étudiants suivent durant les premières semaines de l'année. Un exemple de preuve par récurrence est aussi donné afin d'illustrer le principe théorique.
Sur la base de ce principe mathématique, la récursivité est expliquée en informatique avec des exemples à l'appui. La validation d'une fonction

récursive est également discutée avec une référence à la terminaison d'un algorithme : il faut « s'assurer que chaque appel récursif réduise la taille du problème jusqu'à atteindre un cas de base ». Cependant, malgré cette référence à l'aspect « preuve » avec la terminaison, aucune preuve d'arrêt d'un algorithme n'est faite dans ce chapitre. Pour vérifier qu'un programme donne le résultat pour lequel il a été conçu, il est seulement exécuté pour différentes valeurs. Les programmes, et donc les algorithmes sous-jacents, ne sont donc jamais prouvés dans ce chapitre.

Pour résumer ce chapitre, nous nous situons encore assez fortement dans la réalisation de programmes informatiques. Les problèmes posés et pour lesquels il faut écrire un programme n'en sont pas vraiment car il s'agit surtout de définitions ou de formules (factorielle d'un naturel, suite de Fibonacci,...). Nous ne pouvons donc pas vraiment parler de l'aspect « problème ». L'aspect « preuve », quant à lui, manque dans ce chapitre alors qu'il aurait facilement pu être introduit.

Chapitre 6 : itérations et chaînes de caractères

Dans ce chapitre sont vus les différents types de boucles qui existent en Python. La syntaxe de ces boucles est une fois encore directement exprimée en Python. La boucle « while » (tant que) est la première à être vue. Il est relaté que la condition doit à un moment donné être évaluée à « false », de sorte que la boucle se termine. C'est ce qui s'appelle la « preuve d'arrêt » d'une boucle. Un exemple d'une telle preuve est donné. Nous trouvons donc une référence à la terminaison d'un algorithme. La boucle « for » (pour) est vue, mais sans référence à la terminaison.
Les chaînes de caractères sont également étudiées dans ce chapitre, mais elles concernent plutôt la programmation en Python que l'algorithmique.
Les problèmes proposés dans ce chapitre sont des petits problèmes qui permettent d'utiliser les structures du Python rencontrées dans ce chapitre. Les exemples ne sont pas forcément mathématiques : il faut, par exemple, écrire une fonction permettant de trier trois mots, quelle que soit la casse des mots.

En résumé, dans ce chapitre, c'est la programmation qui domine avec les structures itératives vues en Python et les chaînes de caractères. Nous trouvons néanmoins l'aspect « preuve » au début du chapitre dans le cadre de la terminaison d'un programme avec une boucle « while ». Nous retrouvons également quelques problèmes à résoudre au moyen d'un programme.
Trois aspects sont donc présents au sein de ce chapitre : l'aspect « ef-

fectivité », l'aspect « problème » et l'aspect « preuve », mais avec une prédominance de l'aspect « effectivité ».

Chapitre 7 : listes

Tout ce qui concerne les listes, « séquences d'éléments », est étudié dans ce chapitre. Les programmes à écrire sont en général très courts puisqu'ils n'utilisent que des fonctionnalités des listes prévues par le langage Python. De plus, les exemples ne sont pas mathématiques.

Pour résumer, ce chapitre traite encore une fois uniquement de programmation. Des méthodes liées au langage sont vues afin d'alléger les programmes.

Chapitre 8 : prouver les propriétés des algorithmes

Le but de ce chapitre est double : il faut montrer, d'une part, que les algorithmes s'arrêtent et, d'autre part, qu'ils sont exacts, c'est-à-dire qu'ils donnent les résultats attendus. Dans ce chapitre, des algorithmes sont donnés à la fois en pseudo-code et en Python et d'autres sont donnés uniquement en Python. Les preuves sont effectuées à partir du code en Python.

La terminaison est prouvée pour un algorithme comportant une boucle « while » ou une boucle « for » ou une structure récursive ; pour des algorithmes ne comportant aucune de ces structures, la terminaison est évidente car toutes les opérations sont effectuées au plus une fois chacune. La preuve de la terminaison d'un algorithme itératif avec une boucle « while » ou d'un algorithme récursif peut s'effectuer au moyen d'une preuve par récurrence. En ce qui concerne la boucle « for », la preuve de la terminaison est quasiment immédiate.

Pour prouver la correction d'un algorithme itératif, il faut trouver, dans la mesure du possible, une assertion vraie à chaque nouveau passage dans la boucle (un invariant de boucle). La preuve s'effectue ensuite par récurrence sur le nombre de passages dans la boucle. La preuve de la correction d'un algorithme récursif peut également s'effectuer au moyen d'une preuve par récurrence.

En résumé, dans ce chapitre, nous retrouvons presque exclusivement l'aspect « preuve ». Des algorithmes avec diverses structures sont prouvés,

même si au final, les exemples sont peu nombreux (1 ou 2 exemples pour chaque structure). Cependant, contrairement aux chapitres précédents, les exemples sont mathématiques (division euclidienne, nombres de Fibonacci,...) dans ce chapitre.
Néanmoins, les algorithmes sont prouvés à partir du code en Python alors que normalement, il n'est pas nécessaire d'y avoir recours pour discuter de la preuve d'un algorithme.

Chapitre 9 : complexité : évaluer l'efficacité des algorithmes

Ce chapitre est consacré à la complexité d'un algorithme. La complexité sert à estimer l'efficacité d'un algorithme ou à comparer deux algorithmes résolvant le même problème. Deux méthodes sont développées dans ce chapitre en vue d'atteindre ces deux objectifs.
La première consiste à implémenter l'algorithme en Python et à utiliser une fonctionnalité de ce langage permettant d'estimer le temps d'exécution. Cette méthode relève donc de la programmation et n'est pas intéressante d'un point de vue algorithmique.

La deuxième méthode est celle permettant d'évaluer théoriquement la complexité d'un algorithme dans le pire des cas en déterminant le nombre d'opérations élémentaires qu'effectue l'algorithme. Ensuite, la notation grand-O est abordée avec quelques propriétés qui sont démontrées. Ces propriétés concernent notamment le fait que, dans une étude de la complexité dans le pire des cas, les facteurs constants ne sont pas importants, de même que les termes d'ordres inférieurs sont négligeables.
Les exemples pour lesquels la complexité est étudiée sont des exemples mathématiques (algorithmes de tri, calcul de l'exposant,...). L'exemple du calcul des nombres de Fibonacci rencontré chez Aldon est également présent dans ce chapitre. Pour rappel, il s'agit d'étudier les algorithmes récursif et itératif résolvant ce problème afin de comprendre pourquoi la version récursive ne fonctionne pas pour de grandes valeurs, contrairement à la version itérative.

En résumé, dans ce chapitre, l'aspect dominant est l'aspect « complexité ». La complexité est étudiée sur des algorithmes mathématiques qui sont souvent utilisés dans des cours lorsqu'il s'agit d'étudier la complexité (algorithmes de tri, notamment). Elle est déterminée directement sur les codes en Python.

Chapitre 10 : dictionnaires

Le chapitre 10 est consacré aux dictionnaires. Les dictionnaires sont semblables aux listes, mais ils sont plus généraux. La manière de les coder et les possibilités qu'ils offrent sont expliquées.
La complexité des opérations sur les dictionnaires est également donnée, mais sans explication. L'estimation du temps d'exécution d'un algorithme comportant un dictionnaire est donnée grâce à la fonction en Python, mais aucun calcul de la complexité n'est effectué.

Pour résumer ce chapitre, nous sommes en plein dans l'apprentissage du langage Python et de ses possibilités. L'utilité des dictionnaires est montrée à l'aide de petits problèmes à résoudre. L'aspect « complexité » est également abordé, mais sans être travaillé.

Chapitre 11 : tuples

Ce chapitre traite des tuples, des séquences de valeurs. Nous trouvons la manière de créer un tuple et les opérations possibles.

Une fois encore, nous sommes pleinement dans l'apprentissage du langage de programmation au travers de ce chapitre.

Chapitre 12 : fichiers et exceptions

Les fichiers et les exceptions sont les sujets de ce chapitre. Les fichiers servent à conserver des données hors d'un programme. Tout ce qui concerne l'écriture et l'ouverture d'un fichier est détaillé.
Une exception est une erreur à l'exécution du programme ; il est expliqué comment les gérer et les éviter.

L'objectif de ce chapitre est donc l'apprentissage du langage Python et des fonctionnalités qu'il propose.

Chapitre 13 : introduction aux objets

Ce chapitre concerne la définition de nouveaux objets en Python (instances d'un certain type), ainsi que les opérations possibles sur ces objets.

En résumé, dans ce chapitre, c'est à nouveau uniquement de la programmation qui est vue. Il s'agit de définir de nouveaux objets afin de résoudre

plus facilement certains problèmes.

Chapitre 14 : introduction aux structures de données

Dans ce dernier chapitre sont vues les listes (voir chapitre 7), les piles (le dernier élément ajouté est le premier à devoir être retiré) et les files (le premier élément ajouté est le premier à devoir être retiré). Ces trois structures nécessitent l'utilisation de listes chaînées. Les listes chaînées permettent le stockage de données pour un nombre de données non connu à l'avance.
À partir des définitions de liste, pile et file, des opérations sont créées dans le but d'écrire un programme permettant de travailler sur une liste, une pile ou une file. La complexité des différentes opérations sur les piles et les files est donnée. Un exemple plus concret est aussi présenté pour utiliser les files : la simulation d'une file d'attente dans un bureau administratif. Rappelons que pour Modeste, de tels programmes ne sont que des programmes de modélisation-simulation et ne peuvent pas être considérés comme des algorithmes car ils ne résolvent aucun problème.

Pour résumer ce chapitre, au travers des listes, des piles et des files, c'est encore une fois la programmation qui est travaillée puisqu'il s'agit d'écrire des opérations en Python permettant de simuler une liste, une pile ou une file. Le problème de la simulation d'une file d'attente est un problème de grande envergure, contrairement aux problèmes rencontrés dans les chapitres précédents et pour lesquels les codes ne comportaient que quelques lignes.
L'aspect « complexité » est présent dans ce chapitre car il est demandé que les opérations sur les piles et les files aient la meilleure complexité possible. Cependant, la complexité n'est pas travaillée : la complexité des opérations est simplement donnée.

V.2 Conclusion de l'analyse

Comme nous avons pu le constater tout au long des différents chapitres, le but principal de ce cours est l'apprentissage d'un langage de programmation et de toutes les fonctionnalités qu'il propose. La conception et la compréhension d'algorithmes et de programmes sont omniprésentes dans ce cours, comme les objectifs du premier chapitre pouvaient le laisser présumer. Résumons maintenant nos constatations vis-à-vis des différents aspects rencontrés.

L'aspect « effectivité » est celui le plus souvent rencontré dans ce cours. En effet, l'effectivité est sous-jacente à tout ce qui touche à l'écriture de programmes informatiques et le but de plusieurs chapitres est d'étudier des structures du langage Python permettant d'alléger les programmes.

L'aspect « problème » est également assez présent. Déjà, dans l'introduction, il est précisé que l'utilité de créer des algorithmes est de résoudre des problèmes.
Un point négatif par rapport à notre objectif qui reste la conception d'une séquence de cours à intégrer dans un cours de mathématiques est que bon nombre de problèmes rencontrés ne s'inscrivent pas dans un contexte mathématique (recherche d'un mot dans un dictionnaire, simulation d'une file d'attente,...).
Cependant, dire que tous les algorithmes n'ont pas un contexte mathématique serait bien évidemment exagéré puisque nous avons rencontré des algorithmes résolvant des problèmes mathématiques comme le calcul des nombres de Fibonacci, le calcul de l'exposant, les algorithmes de tri,... Les chapitres comportant le plus d'algorithmes en lien avec les mathématiques sont les chapitres sur la preuve et sur la complexité, soit les deux chapitres dans lesquels il est possible de travailler les algorithmes en tant qu'objets des mathématiques.

L'aspect « preuve » est également présent dans ce cours. Au chapitre 5 (récursivité), il n'est pas présent alors qu'il aurait facilement pu être ajouté. Dans ce chapitre, nous sentons que l'enseignant veut attirer l'attention des étudiants sur la terminaison d'un algorithme récursif, mais la preuve n'est pas donnée. Au chapitre 6 (itérations), des algorithmes sont prouvés uniquement sur des exemples, ce qui ne constitue pas une preuve valide. Il faut certes admettre qu'il n'aurait peut-être pas été évident d'introduire les techniques de preuves d'un algorithme en plein milieu d'un chapitre dont le but est tout autre.
Il faut attendre le chapitre 8 (preuve) pour prouver les algorithmes. Dans ce chapitre, nous trouvons des preuves pour des algorithmes similaires à ceux trouvés dans les chapitres 5 et 6.
Dans les chapitres ultérieurs au chapitre 8, l'aspect « preuve » n'apparaît plus. Pour voir s'il donne le résultat attendu, un programme est testé pour différentes valeurs. L'aspect « preuve » semble donc être cloisonné dans le chapitre 8.

Le dernier aspect qui nous intéresse est l'aspect « complexité ». Avant le chapitre 9, chapitre traitant de la complexité, aucune référence n'y est faite. Après le chapitre 9, quelques références sont présentes, même si plus aucune étude de la complexité globale d'un algorithme n'est effectuée. C'est notamment le cas du chapitre 14 où il faut créer des opérations sur les piles et les files avec une complexité optimale. Cependant, la complexité de ces opérations est juste donnée et il est dit qu'il ne serait pas possible de faire mieux, mais sans explication.
Enfin, la complexité d'un algorithme n'est pas seulement étudiée mathématiquement en comptant le nombre d'opérations élémentaires, elle est également illustrée à l'aide de graphiques obtenus en exécutant un programme conçu à partir de l'algorithme. Cela permet de « visualiser » la complexité qui, il faut l'avouer, est une notion assez abstraite.

En conclusion, les quatre aspects mis en évidence par Modeste et attendus dans l'enseignement de l'algorithmique au lycée en France sont développés dans ce cours de Programmation et Algorithmique I, mais ils ne le sont pas en égales proportions. En effet, l'aspect « effectivité » est nettement dominant : le but est d'apprendre un langage de programmation en profondeur. Ce dernier point distingue assez bien ce cours du manuel français résumé au chapitre II puisque, dans ce manuel, plusieurs langages de programmation étaient étudiés suite à la volonté des programmes français de n'imposer aucun langage.
Les autres aspects sont développés, mais en plus faible proportion. Peut-être qu'ils le sont davantage dans les exercices relatifs à ce cours ou dans des cours ultérieurs à celui-ci, car rappelons qu'il s'agit du premier cours universitaire sur le sujet que rencontrent les étudiants. Quant aux aspects « preuve » et « complexité », ils semblent réellement être cloisonnés au sein des chapitres y étant consacrés.

V.3 Élaboration de la structure de la séquence d'enseignement

Souvenons-nous que notre objectif est de concevoir une séquence de cours sur l'algorithmique à intégrer dans un cours de mathématiques dans le secondaire supérieur. Nous extrayons alors de notre analyse du cours les éléments que nous jugeons pertinents d'intégrer dans notre séquence. Dès lors, tout ce qui a trait directement à la programmation nous intéresse dans

une moindre mesure.
Tout comme dans le cours de Programmation et Algorithmique I, il serait intéressant de commencer par une introduction pour définir la notion d'« algorithme », notion que les élèves ne connaissent pas forcément. Plusieurs exemples seront également nécessaires pour d'illustrer cette notion. Ces premiers exemples ne devront pas nécessairement être mathématiques, afin de montrer aux élèves que les algorithmes ne se retrouvent pas seulement dans des contextes mathématiques. Cependant, en plus de ces exemples non mathématiques, il y aura lieu d'étudier des exemples mathématiques accessibles aux élèves du secondaire.

Ensuite, dans le cours étudié, les deux chapitres suivants concernent les types, variables et expressions et les fonctions. Comme relaté dans l'analyse effectuée plus haut, ces deux chapitres sont consacrés à l'apprentissage du langage Python et ne nous intéressent donc pas dans le cadre de notre travail.

Le quatrième chapitre concerne les instructions conditionnelles (« si...alors...sinon »). Nous jugeons que cette structure est importante car en mathématiques, il arrive que nous ayons à discuter de l'état de vérité d'une proposition. C'est notamment le cas lors de la recherche de racines d'une équation du second degré dans $\mathbb{R}$ pour laquelle il faut discuter de l'existence et de l'unicité des solutions à partir de la valeur du discriminant : s'il est strictement positif, alors il existe deux racines distinctes ; s'il est nul, alors il n'existe qu'une racine ; sinon, il n'y a pas de racines dans les réels.
Le sixième chapitre, consacré aux itérations, nous semble également important pour la construction de notre séquence de cours. En effet, en mathématiques, nous avons parfois besoin de répéter certaines opérations. Par exemple, si nous devons calculer a^n pour $n \geqslant 2$, nous effectuons les calculs $a.a$ puis $a.a.a$,..., jusqu'à ce que les $n-1$ multiplications aient été effectuées.
La deuxième partie de notre séquence de cours sera donc consacrée à l'apprentissage des différentes structures qui composent un algorithme car sans leur apprentissage, il ne serait pas envisageable d'étudier des algorithmes intéressants d'un point de vue mathématique. Cependant, ces différentes structures devront être données en pseudo-code et non pas dans un langage de programmation.

Le chapitre 5 sur la récursivité nous semble très intéressant d'un point de vue mathématique puisque la récursivité est en lien direct avec la preuve par récurrence. Cependant, en général, il est possible de transformer un algorithme récursif en un algorithme itératif. Il n'est donc pas nécessaire

d'étudier la récursivité dans le cadre d'une introduction à l'algorithmique, surtout s'il est possible de s'en passer, car nous trouvons que c'est une notion assez compliquée et qu'il peut être difficile de comprendre qu'un algorithme fasse appel à lui-même.
Le septième chapitre est à nouveau un chapitre de programmation.

Enfin, nous arrivons aux chapitres 8 et 9 consacrés à la preuve des algorithmes et à la complexité. Ce sont ces chapitres qui permettent de travailler l'algorithme en tant qu'objet des mathématiques, ce qui nous intéresse fortement, étant donné que notre séquence d'enseignement doit s'inscrire dans le cadre d'un cours de mathématiques. Bien qu'étant tous deux des chapitres plutôt mathématiques, nous pensons qu'il vaut mieux les séparer, comme cela est fait dans le cours universitaire, car la preuve et la complexité sont quand même des notions distinctes. Dès lors, dans notre séquence d'enseignement, une partie sera consacrée à la preuve et une autre à la complexité. Nous décidons, comme cela est fait dans le cours, de commencer par la preuve car elle semble plus facile à aborder. En effet, nous avons vu que dans le cours, la preuve d'algorithmes était déjà évoquée dans des chapitres antérieurs au chapitre y étant consacré, alors qu'aucune référence à la complexité n'était faite.
Nous verrons donc la preuve et la complexité dans des parties spécifiques, comme cela est fait dans le cours de Programmation et Algorithmique I. Nous tâcherons cependant d'omettre les références à ces aspects dans les deux premières parties afin de ne pas tout mélanger et d'avoir une structure bien claire et bien délimitée. Effectivement, nous avons vu que dans le cours universitaire, l'aspect « preuve » était évoqué avant le chapitre sur le sujet, mais sans plus de détails, laissant le lecteur sur sa fin.

Finalement, les cinq derniers chapitres du cours de Programmation et Algorithmique I sont plutôt tournés vers la programmation et ne nous intéressent donc pas dans le cadre de notre séquence d'enseignement.

V.4 Bilan

Dans ce chapitre, nous avons dans un premier temps analysé le cours de Programmation et Algorithmique I et nous avons constaté que tous les aspects étaient présents dans ce cours, même ceux qui ne figuraient pas dans les ressources françaises pour le lycée (preuve et complexité). L'algorithme est donc aussi bien vu en tant qu'outil qu'en tant qu'objet dans ce cours.

Cependant, l'aspect « effectivité » est davantage travaillé par rapport aux autres aspects car un des buts de ce cours est l'apprentissage du langage Python.

Suite à cette analyse des différents chapitres, nous avons établi la structure pour notre séquence d'enseignement. Cette structure offre la possibilité de travailler l'algorithme en tant qu'outil pour la résolution de problèmes, mais également en tant qu'objet des mathématiques.

Maintenant que nous avons regardé de plus près ce cours, nous allons nous intéresser aux difficultés des étudiants vis-à-vis de celui-ci afin d'avoir une idée de ce qui pose problème, dans le but de concevoir notre séquence d'enseignement en conséquence. C'est l'objet du prochain chapitre.

Chapitre VI

Difficultés des étudiants face à l'algorithmique

Dans ce chapitre, nous analysons différents questionnaires proposés à des étudiants universitaires afin de déterminer leurs difficultés face à l'algorithmique.

VI.1 But de l'analyse et questionnaires choisis

Au travers des réponses des étudiants de première année en sciences mathématiques et informatiques à différentes évaluations de première session en lien avec l'algorithmique, nous souhaitons déterminer les difficultés rencontrées par ces étudiants en ce qui concerne l'algorithmique. Déterminer leurs difficultés nous sera utile lors de la conception de notre séquence d'enseignement car de cette façon, nous saurons quels points de matière seront susceptibles d'être problématiques pour des élèves du secondaire et donc sur quels points nous devrons davantage nous attarder. Dans le cadre de notre travail, il est intéressant de s'attarder sur les points de matière qui posent des problèmes aux étudiants universitaires car nous souhaitons que notre séquence de cours permette une transition secondaire-université pour l'algorithmique.

Concrètement, les questionnaires choisis sont au nombre de quatre. Le premier est issu du *Test introductif* de septembre 2012. Il s'agit d'un test proposé aux étudiants le jour de la rentrée académique afin d'évaluer leurs connaissances sur certains points de matières de mathématiques du secondaire. Dans ce test, la dernière question concerne l'élaboration d'une démarche pour résoudre un problème donné. Cette question n'est pas

directement en lien avec les matières du secondaire, mais elle nécessite de la logique, de la réflexion, de l'argumentation,..., compétences essentielles en mathématiques. Nous analysons les copies des étudiants en sciences mathématiques et en sciences informatiques.

Le deuxième questionnaire retenu est un test de *Mathématiques pour l'Informatique I* de décembre 2013. Ce cours est un cours de mathématiques suivi par les étudiants de première année en sciences informatiques.
Une question de ce test est dédiée à l'exécution d'un algorithme et à sa preuve. Cette question est donc particulièrement intéressante dans le sens où nous avons décidé de consacrer une partie de notre séquence d'enseignement à la preuve d'un algorithme.

Le troisième questionnaire ayant retenu notre attention est l'examen de janvier 2014 du cours de *Programmation et Algorithmique I*. Nous avons choisi cet examen car nous avons analysé ce cours au chapitre précédent et nous savons donc ce qui y est travaillé. Nous analysons uniquement les copies des étudiants en sciences mathématiques et informatiques pour deux raisons. La première est que nous pensons que les mathématiciens auront davantage de facilités avec les aspects plus mathématiques de l'algorithmique, tandis que les informaticiens trouveront plus faciles les questions liées à l'écriture d'un programme. La seconde est qu'il est plus facile pour nous d'avoir un retour de la part des étudiants de ces deux sections.

Enfin, le dernier questionnaire pris en compte est un questionnaire que nous avons proposé aux étudiants de mathématiques et d'informatique afin qu'ils donnent leur avis vis-à-vis du cours de Programmation et Algorithmique I. Celui-ci a été complété par les étudiants après l'examen de janvier 2014.

VI.2 Test introductif

L'énoncé de la question proposée aux étudiants est présenté ci-dessous.
Ce problème nous intéresse car il peut donner lieu à un algorithme. En effet, il est général car il est valable pour une famille d'instances : l'ensemble des suites de cinquante nombres. L'aspect travaillé dans cette question est donc l'aspect « problème ». Les autres aspects ne sont pas présents puisqu'il ne s'agit pas d'écrire un algorithme en pseudo-code ou un programme ni d'étudier la procédure décrite. Même si la preuve n'est pas exigée, il

faut quand même justifier la démarche pour convaincre le lecteur que la procédure proposée va bien donner le résultat attendu.

Énoncé du problème

Considérons une suite de cinquante nombres numérotés de 1 à 50. Ces nombres sont notés a_i avec i prenant des valeurs comprises entre 1 et 50. Un *pic* est défini comme un élément plus grand que les deux éléments qui l'entourent ($a_{i-1} < a_i > a_{i+1}$). On souhaiterait savoir si cette suite de 50 nombres comporte un pic et un seul.
On vous demande d'expliquer en français comment effectuer ce test. La personne à laquelle vous expliquez est censée être capable de réaliser les opérations arithmétiques élémentaires ($+$, -, $\times$, $\div$), ainsi que des tests basés sur des comparaisons.

Question préliminaire

À travers cette question, nous nous demandons si des étudiants fraichement sortis du secondaire sont capables de trouver une démarche dans le but de résoudre un problème donné et de l'expliquer en français. Trouver une démarche pour résoudre un problème leur sera utile pour le cours de Programmation et Algorithmique I qu'ils sont amenés à suivre. En effet, dans ce cours, il leur sera demandé d'écrire un programme ou un algorithme permettant de résoudre un problème, mais avant d'écrire ce programme ou cet algorithme, ils devront réfléchir à une démarche pour la résolution.
Nous ne pensons pas trouver énormément de différences entre les copies des mathématiciens et celles des informaticiens car ce test est donné à la rentrée, avant tout autre cours. Les étudiants n'ont donc pas encore eu l'occasion de se spécialiser.

Analyse des copies

Les copies des étudiants sont au nombre de 91 : 36 copies d'étudiants en mathématiques et 55 d'étudiants en informatique. Nous pouvons, dans un premier temps, effectuer un tri de ces copies en retirant les copies blanches. Nous en arrivons alors à 21 copies en mathématiques et à 24 en informatique, soit 45 copies en tout. Il y a donc un taux d'abstention de 42% en mathématique et de 56% en informatique pour cette question.

Nous pensons que plusieurs facteurs sont à l'origine de ce nombre élevé de copies blanches (plus de la moitié des copies). Tout d'abord, il s'agissait de la dernière question du test comportant 9 questions. Les étudiants ont peut-être manqué de temps pour répondre à cette question.
Ensuite, les autres questions du test consistaient en des questions auxquelles les étudiants ont été confrontés durant leurs années dans le secondaire (réduction de fractions, résolution d'équations,...). Les points de matière en jeu dans chaque question étaient donc assez visibles aux étudiants. Or, cette dernière question ne se ramenait pas à une matière vue dans le secondaire. D'ailleurs, le problème proposé n'est en fait pas si compliqué et n'exige pas une matière spécifique des mathématiques. L'énoncé est certes assez long et comporte beaucoup d'informations.
Les étudiants ont donc peut-être passé cette question car ils ne voyaient pas à quel point de matière elle se référait et ne savaient donc pas comment l'entreprendre. De plus, la longueur de l'énoncé en français par rapport aux autres énoncés du test a peut-être déstabilisé certains étudiants.

Nous pouvons classer les 45 copies restantes en différentes catégories : celles où l'énoncé est recopié en partie et qui n'apportent donc aucune réponse ; celles où les étudiants ont travaillé sur une suite avec des valeurs connues ou ont confondu l'indice et la valeur de la variable ; celles où les raisonnements sont totalement erronés et celles où le raisonnement est correct, même s'il est parfois incomplet.

Citons quelques exemples de copies d'étudiants qui ont travaillé sur un exemple ou qui ont confondu l'indice et la valeur de la variable.

E_1 : $a_{i-1} < a_i > a_{i+1}$. Prenons, par exemple, $i = 3$. $a_2 < a_3 > a_4$. Le pic ne se trouve pas à la position 3. Il n'y a pas de pic à cette suite.

E_2 : Nous remarquons que $a_i > a_{i-1}$ est toujours vrai, mais que $a_i > a_{i+1}$ ne se vérifie pas. Il n'y a donc pas de pic.

E_3 : $1 = a_4$, $2 = a_6$, $3 = a_1$, $4 = a_2$, $5 = a_3$, $6 = a_5$. Donc $a_5(6) > a_4(1)$ et $a_5(6) > a_6(2)$.

Dans les raisonnements erronés, nous trouvons des éléments tels que des références aux suites croissantes, décroissantes, arithmétiques, géométriques,... Les étudiants essaient donc de se ramener à des points de matière qu'ils connaissent. Cependant, nous trouvons également des raisonnements plus farfelus. Dans ces différentes copies, les étudiants n'ont pas pris la peine de tester leurs algorithmes sur des cas particuliers. Citons quelques exemples de raisonnements erronés afin d'illustrer nos dires.

E_4 : Il suffit d'additionner la suite de nombres allant de 1 à 50. Et en fonction du résultat il sera possible de déceler un pic.

E_5 : Pour avoir un pic dans une suite de nombres, il faut que la suite soit géométrique.

E_6 : Pour savoir si une suite comporte un pic, il faut savoir le type de suite à laquelle on a affaire et en calculer la raison. En fonction de cela, on aura tous les éléments composant cette suite et on pourra en déduire s'il y a un pic.

En mathématiques, les 21 copies restantes se répartissent comme suit :
- 6 copies où l'énoncé est recopié,
- 5 copies où le travail est effectué sur un exemple ou avec une confusion indice/valeur,
- 6 copies avec un raisonnement erroné,
- 4 copies avec un raisonnement correct.

En informatique, les 24 copies se classent de la sorte :
- 8 copies où l'énoncé est recopié,
- 3 copies où le travail est effectué sur un exemple ou avec une confusion indice/valeur,
- 3 copies avec un raisonnement erroné,
- 10 copies avec un raisonnement correct.

Au total, nous retrouvons donc un raisonnement correct dans seulement 14 copies d'étudiants. Cependant, chez certains étudiants, ce raisonnement n'est pas toujours abouti. Par exemple, deux d'entre eux expliquent comment déterminer s'il y a un pic en une position donnée, mais pas pour toutes les positions. Nous relatons leurs écrits.

E_7 : On choisit un élément pour le tester, par exemple a_{12}. Pour savoir si cet élément est un pic, on le soustrait par son prédécesseur (a_{11}). Si on obtient un nombre positif, on continue, sinon, on peut passer au suivant. Si le résultat est positif, on reprend l'élément a_{12} et on le soustrait cette fois par son successeur (a_{13}). Si à nouveau, le résultat est positif, on a bel et bien un pic, sinon ce n'est pas le cas.

E_8 : On commence par prendre a_2 et on lui retire a_3. Si $a_2 - a_3 > 0$, on continue le test et on reprend a_2 en lui retirant a_1. Si c'est toujours > 0, nous avons un pic. Si on a un pic, on continue la vérification à a_{i+2}, ici a_4. Si $a_2 - a_3 < 0$, on arrête et on passe à a_3.

Dire qu'il faut effectuer le test pour toutes les positions n'est pas correct puisque les valeurs aux positions 1 et 50 ne peuvent pas être des pics étant donné qu'il n'y a aucune valeur avant la première et aucune après la cinquantième. Les étudiants devaient alors prendre en compte ces cas extrêmes et le faire remarquer dans leurs copies. Ils sont 8 étudiants sur les 14 à le signaler.

Enfin, l'énoncé demandait de tester l'unicité d'un pic pour une suite de 50 valeurs. Seuls 7 étudiants sur les 14 (soit la moitié) n'ont pas oublié d'expliquer comment vérifier l'unicité d'un pic et répondent alors réellement à la question de départ. Chez bon nombre d'étudiants, il n'y a donc pas de retour sur l'énoncé une fois l'exercice terminé.

Finalement, nous pensons qu'au moins 4 étudiants parmi les 14 ont suivi un cours d'algorithmique et/ou de programmation dans leurs cursus scolaires (1 en mathématiques et 3 en informatique). En effet, dans leurs copies, nous trouvons des termes tels que « le compteur est initialisé », « le compteur est incrémenté », « le bloc instructions », « on affecte la valeur », « for...if...and », « pic++ »,... Ce sont quatre étudiants qui ont bien réussi la question puisqu'ils ont tous pensé à vérifier l'unicité du pic.

Conclusion

Au travers de ce test, nous avons remarqué que l'écriture d'une démarche pour résoudre un problème donné était une compétence loin d'être acquise à l'entrée à l'université puisque peu d'étudiants ont été capables de concevoir un raisonnement correct pour résoudre le problème. Il s'agit donc d'une compétence primordiale à développer et à travailler au sein d'un cours d'algorithmique afin de pouvoir écrire, par la suite, un programme ou un algorithme permettant de résoudre un problème donné.
Nous ne nous attendions pas à constater de différences notables entre les copies des mathématiciens et celles des informaticiens et effectivement, ils commettent les mêmes types d'erreurs. Cependant, chez les étudiants n'ayant pas rendu une copie blanche, proportionnellement, deux fois plus d'informaticiens que de mathématiciens avaient un raisonnement correct à la base.

VI.3 Test de Mathématiques pour l'Informatique I

Le cours de Mathématiques pour l'Informatique I est destiné aux étudiants en première année d'informatique. Le problème proposé aux étudiants qui nous intéresse est présenté à l'annexe B. Il s'agit d'un algorithme permettant de trouver le reste de la division euclidienne de deux naturels. Cet algorithme a déjà été rencontré par ces étudiants dans le cours de Programmation et Algorithmique I avec d'autres notations et d'autres noms de variables.
Les buts de cet exercice sont l'exécution d'un algorithme au point (a), la généralisation au point (b), la preuve par récurrence au point (c) et la déduction au point (d).

Questions préliminaires

Face à cet énoncé, nous nous posons plusieurs questions pour lesquelles nous donnons notre avis avant d'analyser les copies des étudiants.

1. Les étudiants arrivent-ils à exécuter cet algorithme pour des valeurs données ?
   → D'après nous, cette question ne devrait pas poser de problèmes. En effet, les étudiants ont déjà rencontré cet algorithme et il n'est pas forcément dur à exécuter. La difficulté peut venir du fait que la boucle est empruntée un grand nombre de fois, mais normalement, après quelques passages dedans, il est aisé de s'apercevoir du but de l'algorithme. La condition « **si** $(d = 0)$ **alors** $b := b^2$ » risque également de troubler quelques étudiants.
2. Parviennent-ils à généraliser sur la base d'un seul exemple ?
   → Une fois encore, nous pensons que ce point sera relativement bien réussi car cet algorithme a déjà été rencontré.
3. Les deux étapes de la preuve par récurrence sont-elles présentes (cas de base et pas de récurrence) ?
   → Étant donné qu'il est clairement demandé d'effectuer une telle preuve, nous nous attendons à ce qu'une majorité d'étudiants fassent figurer les deux étapes de la preuve.
4. Ces deux étapes sont-elles prouvées correctement (utilisation de l'hypothèse de récurrence, par exemple) ?
   → Nous pensons que le cas de base sera prouvé correctement par bon nombre d'étudiants car c'est en général la partie de la preuve qui est

la plus facile. Pour le pas de récurrence, nous nous attendons à ce que ce soit un peu moins bon puisqu'il faut faire intervenir plus d'éléments (hypothèse de récurrence, actualisation des valeurs des variables après un passage dans la boucle,...). Cependant, les étudiants ont déjà prouvé par récurrence de tels algorithmes dans le cours de Programmation et Algorithmique I.

5. Arrivent-ils à exploiter une propriété afin d'en déduire un résultat ?
   → D'après nous, cette question devrait être moyennement réussie. En effet, normalement, les étudiants devraient savoir y répondre même sans avoir réussi à prouver le point précédent. Il s'agit simplement de comprendre ce que deviennent les variables à la fin de l'algorithme. Cependant, nous pensons que bon nombre d'étudiants n'y répondront pas car, comme il s'agit de la dernière question, ils n'auront peut-être pas le temps d'y arriver. De même, certains risquent de penser que comme les points (c) et (d) sont liés, ils ne peuvent pas répondre au point (d) sans avoir une réponse correcte au point (c).

Analyse des copies

Nous disposons de 51 copies d'étudiants. Nous répondons donc aux questions du paragraphe précédent grâce à ces copies.

Les étudiants arrivent-ils à exécuter cet algorithme pour des valeurs données ?

Sur les 51 étudiants, 41 parviennent à exécuter cet algorithme correctement. Nous nous attendions à ce que cette question soit bien réussie, mais nous constatons que 10 étudiants (20%) ne sont pas parvenus à trouver la bonne réponse, ce qui n'est pas négligeable. Parmi ces 10 erreurs, nous trouvons, entre autres, deux étudiants qui donnent le quotient plutôt que le reste, deux qui ont oublié de retirer une fois 4 et qui donnent donc 7 comme réponse au lieu de 3, un qui ne répond rien,...

Parviennent-ils à généraliser sur la base d'un seul exemple ?

Parmi les 41 étudiants qui sont parvenus à trouver la réponse correcte au point (a), 39 réussissent à généraliser la sortie. Nous pensions, en effet, que ce point ne devait poser aucun problème. Nous relatons les réponses des deux étudiants n'ayant pas réussi à généraliser à partir de l'exemple.

E_1 : L'algorithme détermine quel est le plus grand multiple de b qui divise a.

E_2 : Le nombre d'opérations qu'il faut effectuer pour trouver un nombre inférieur à un entier b en le soustrayant à un entier a.

Dix étudiants n'étaient pas parvenus à trouver la réponse correcte au point (a), mais parmi eux, trois ont quand même trouvé la réponse correcte au point (b). Ils n'ont donc pas confronté ces deux points. Deux étudiants ne répondent rien.

Les deux étapes de la preuve par récurrence sont-elles présentes (cas de base et pas de récurrence) ?

Nous nous attendions à trouver une majorité de copies comportant les deux parties de la preuve comme il était clairement demandé d'effectuer une telle preuve. Cependant, la réalité est toute autre puisque seulement 26 étudiants sur les 51 font référence aux deux étapes (51% des étudiants). Les 25 copies ne comportant pas les deux étapes sont soit des copies blanches (8), soit des copies où il manque au moins une étape (17). Dans ces différentes copies où au moins une étape est manquante, certaines comportent une explication qui ne constitue pas une preuve par récurrence. Montrons quelques exemples.

E_1 : Le nombre de passages dans la boucle est référencé par $c = \frac{a-d}{b}$. Donc $a = bc + d = b.\frac{a-d}{b} + d = a$.
(Cet étudiant utilise deux fois la même égalité donc il ne prouve rien au final.)

E_2 : Avant la boucle : $a = b.c + d = 0 + d = 0 + a$. Donc $a = a$, ok. Après la boucle : $a = bc + d = 4.12 + 3 = 53$.

E_3 : c représente le nombre de passages dans la boucle. À chaque passage, on a retiré b à a. On a arrêté la boucle au moment où on ne pouvait plus retirer b à a. Cela veut dire que c représente la division entière de a par b et d représente le reste après division entière. Dès lors, on peut dire que $a = b.c + d$ correspond à dividende = diviseur.quotient + reste, égalité qui n'est plus à démontrer.

Dans ces démonstrations où au moins une étape est manquante, nous avons constaté que souvent, les étudiants utilisaient deux fois la même égalité pour prouver un résultat, ce qui fait qu'ils « tournent en rond » et qu'ils ne montrent rien au final.

Ces deux étapes sont-elles prouvées correctement (utilisation de l'hypothèse de récurrence, par exemple) ?

Commençons par le cas de base. Sur les 41 étudiants qui ont répondu à cette question, le cas de base figure chez 31 d'entre eux. Cependant, seuls 15 étudiants parmi ceux qui y ont pensé l'ont prouvé correctement (la moitié !). Nous pensions que le cas de base aurait été bien réussi. Pour 2 étudiants, le cas de base est $d < b$ (la négation de la condition de la boucle). Chez les 14 autres qui ont mauvais, la preuve du cas de base est effectuée avec les valeurs du point (a) ou avec d'autres valeurs. Il n'est donc pas prouvé en général. Nous montrons quelques exemples de ces preuves sur des cas particuliers.

E_1 : Cas de base : $c = 0$; $a = b.c + d$; $a = 131$ et $b.c + d = 4.0 + 131 = 131$. Le cas de base est vérifié.

E_2 : Cas de base : $a = 0$, $b = 1$, $c = 0$ et $d = a = 0$. Montrons que $a = b.c + d$ est vraie, c'est-à-dire $0 = 1.0 + 0$. Vrai.

En ce qui concerne le pas de récurrence, aucun étudiant ne l'a démontré sans d'abord démontrer le cas de base. Il est vrai que dans certaines copies, nous trouvons un semblant de pas de récurrence sans cas de base, mais il n'est jamais fait mention que c'est le pas de récurrence qui est tenté d'être démontré. Il y a donc 26 étudiants - les 26 qui ont les deux étapes - qui ont démontré un pas de récurrence. Dans ces différentes copies, nous trouvons des pas de récurrence qui n'en sont pas car les étudiants n'exploitent pas l'hypothèse de récurrence, ou bien ils les prouvent sur des cas particuliers, ou encore, ils arrivent à des incohérences.

E_3 : On sait que $a = bc + d$, $\forall c \leqslant k$. Prouvons-le pour $c = k + 1$. Montrons que $a = b(k + 1) + d$, où $d = a - b(k + 1)$. On a donc $a = b(k + 1) + (a - b(k + 1)) \Leftrightarrow a = a + b(k + 1) - b(k + 1)$. On a bien $a = a$.
(Cet étudiant utilise à deux reprises l'égalité qu'il doit prouver, il ne prouve donc rien.)

E_4 : Supposons la propriété vraie à k ; $134 = 4k + (134 - 4k)$. Montrons pour $k+1$: $134 = 4(k+1) + (134 - 4(k+1))$; $134 = (4k+4) - (4k+4+134)$; $134 = 134$. Vrai pour $k + 1$ donc la proposition est prouvée.

E_5 : Supposons que l'égalité est vérifiée pour tout $0 \leqslant c \leqslant k$, $k \in \mathbb{N}_0$, c'est-à-dire $a = bk + d$. Montrons que c'est aussi vrai pour $c = k + 1$, c'est-à-dire $a = b(k + 1) + d$; $a = bk + b + d$. Par hypothèse de récurrence, on sait que $a = bk + d$, donc $a = a + b$.

Au final, des 26 copies comportant les deux étapes de la preuve par récurrence, seules 3 copies sont à peu près correctes (cas de base prouvé en général, hypothèse de récurrence présente et exploitée,...), ce qui est assez peu étant donné que la preuve par récurrence d'un algorithme a déjà été étudiée dans le cours de Programmation et Algorithmique I.

Arrivent-ils à exploiter une propriété afin d'en déduire un résultat ?

Comme nous pouvions nous y attendre, un peu plus de la moitié des copies sont sans réponse pour cette question (27/51). Dans les 24 copies des étudiants qui y ont répondu, 16 réponses ne font pas explicitement référence au point (c). Citons quelques unes de ces copies.

E_1 : On sait que $\forall a, b \in \mathbb{Z}, \forall m \in \mathbb{N}_0$, $(a.b) \mod m = ((a \mod m).(b \mod m)) \mod m \Rightarrow \forall b, c \in \mathbb{Z}, \forall d \in \mathbb{N}_0 : (b.c) \mod d = ((b \mod d).(c \mod d)) \mod d \Rightarrow ((4 \mod 131).(0 \mod 131)) \mod 131 = 4 \mod 131$.

E_2 : On voit ici que c s'incrémente à chaque passage et que d diminue de b. On cherche donc d comme étant $a \mod b$, d est donc le reste, c est le quotient et b le diviseur.

E_3 : $a = b.j + a \mod b$ où $0 \leqslant j < b$ et $a \mod b < b$ qui est la condition d'entrée dans la boucle.

Les 8 copies qui exploitent le point (c) sont assez bonnes, avec parfois des manques de précision.

Conclusion

Au travers de ce test, nous avons constaté que l'exécution d'un algorithme ne posait en général aucun problème. Ces étudiants étaient, en effet, déjà familiarisés avec l'exécution d'algorithmes suite au cours de Programmation et Algorithmique I. Les quelques problèmes survenus sont, d'après nous, des problèmes liés au nombre conséquent de passages dans la boucle et à la condition « **si** $(d = 0)$ ». Nous pensons donc, pour notre séquence d'enseignement, que les exécutions d'algorithmes devront être assez courtes afin de ne pas introduire d'éventuelles fautes de calcul.
En ce qui concerne la généralisation, les étudiants n'ont eu aucun problème à généraliser sur la seule base d'une exécution. Néanmoins, il faut reconnaître qu'il est assez aisé, pour cet algorithme, de comprendre ce qu'il retourne. Nous pensons que travailler la généralisation dans notre séquence de cours peut être une bonne idée car « formuler des généralisations » fait partie

des compétences terminales à maîtriser à la fin du secondaire. Cependant, en fonction des algorithmes, plusieurs exécutions seront parfois nécessaires pour en déterminer les sorties.
La preuve par récurrence ne semble pas du tout être acquise. Chez bon nombre d'étudiants, au moins une des deux parties de la preuve est manquante. Pourtant, les étudiants l'ont rencontrée dans les cours de Mathématiques élémentaires (cours donné au début du premier quadrimestre de la première année en sciences mathématiques, informatiques et physiques), Programmation et Algorithmique I, Mathématiques pour l'Informatique I et éventuellement dans le secondaire. Même si cette preuve semble difficile à maîtriser, nous jugeons qu'elle est intéressante et nous pensons l'intégrer dans notre séquence d'enseignement en insistant fortement sur la nécessité des deux étapes.
La déduction n'est en général pas maîtrisée. Beaucoup d'étudiants n'exploitent pas la proposition qui est considérée comme vraie.
Enfin, beaucoup de réponses incorrectes des étudiants viennent du fait qu'ils ne respectent pas les consignes. Par exemple, il leur est demandé explicitement d'effectuer une preuve par récurrence et certains tentent de démontrer la proposition autrement. De même, lorsqu'ils doivent utiliser le point (c), certains ne s'en servent pas.

VI.4 Examen de Programmation et Algorithmique I

Suite à notre analyse de ce cours dans le chapitre précédent, nous pensons que les principaux axes de difficultés sont les suivants :

1. l'écriture d'algorithmes car nous avons constaté, dans le Test introductif, que l'élaboration d'une démarche pour résoudre un problème était une compétence loin d'être acquise lors de l'entrée à l'université,
2. les chapitres sur la preuve et la complexité car ils sont plutôt mathématiques et nous avons remarqué qu'ils se distinguaient assez nettement des autres chapitres,
3. l'écriture d'un programme en Python car nous avons vu que ce langage proposait bon nombre de fonctionnalités à connaître et à maîtriser.

Intéressons-nous maintenant à l'examen écrit de janvier 2014. Les quatre questions de cet examen sont présentées à l'annexe B. Les différents axes de difficultés que nous venons de signaler sont mis en jeu dans ces questions.

La première question est une question de compréhension de programmes Python. Il s'agit d'interroger les étudiants sur les subtilités de ce langage. Cette question ne nous intéresse donc pas dans le cadre de notre travail car nous désirons nous concentrer sur l'algorithmique.
Les deuxième et troisième questions consistent à écrire des programmes en Python en vue d'apporter des solutions aux problèmes proposés. Ces questions nous intéressent car derrière les programmes Python se trouvent des algorithmes. Nous désirons voir si la conception d'un algorithme pour apporter la solution à un problème semble être acquise par les étudiants après qu'ils aient suivi ce premier cours de programmation et d'algorithmique.
Enfin, pour la quatrième question, un programme est donné dans le but d'étudier sa preuve et sa complexité. Il s'agit donc d'une question qui nous intéresse fortement puisque ce genre de questions faisait défaut dans l'enseignement de l'algorithmique au lycée français.
Nous travaillons séparément les trois questions que nous avons retenues. Les cotations dont nous parlons par la suite sont les cotes attribuées aux étudiants par les correcteurs de l'examen.

VI.4.1 Deuxième question

Question préliminaire

Face au problème posé dans la deuxième question, nous nous demandons si les erreurs commises par les étudiants relèvent davantage de l'algorithme sous-jacent au programme à trouver ou davantage de la programmation avec toutes les subtilités du langage Python.
Nous pensons que cette question peut être moyennement réussie par les étudiants, avec un taux de réussite plus élevé en informatique car le problème n'est pas forcément compliqué à comprendre, mais il nécessite une bonne connaissance des listes en Python.

Analyse des copies

Les étudiants à avoir présenté cet examen sont au nombre de 48 en informatique et de 29 en mathématiques. Pour cette question, 20 étudiants en informatique (42%) et 11 en mathématiques (38%) ont eu 0. Nous tentons d'expliquer pourquoi autant d'étudiants ont eu une telle cote à cette question. Tout d'abord, certains d'entre eux n'ont rien répondu : ils sont 6 en informatique et 6 en mathématiques. Parmi ceux qui ont 0 et qui ont répondu, certains ont juste écrit le nom des fonctions et d'autres encore ont mal compris les deux sous-questions. Par exemple, pour le point a), des étu-

diants considèrent qu'une star est une personne qui connaît la moitié des gens. Or, dans l'énoncé, il est dit qu'une star est une personne connue par au moins la moitié des personnes, ce qui n'est sensiblement pas la même chose. Pour le point b), un étudiant regarde si le nombre de personnes que connaît quelqu'un correspond au nombre de stars, ce qui n'est pas non plus correct puisque nous pouvons connaître autant de personnes qu'il y a de stars sans forcément connaître la moindre star. Citons encore la copie d'un étudiant n'ayant rien compris au problème :

E_1 : a) Selon le nombre de people donné, la fonction retournera le double de stars, puisqu'une star est connue par la moitié des people.

if stars $>$ 0
return people $\times$ 2.

b) Selon le nombre de people donné, la fonction retournera exactement le même nombre de fans puisque les stars sont connues par la moitié des people, ce qui implique qu'un people connaît la moitié des stars et qu'un fan connaît toutes les stars.

fan = people
  if people $>$ 0
  return fan.

Les étudiants ayant eu 0 à cette question sont donc des étudiants n'ayant pas réussi à trouver le bon algorithme pour résoudre le problème.

Les autres étudiants ont trouvé un algorithme correct pour au moins un des deux points. Ils sont donc 28 en informatique (58%) et 18 en mathématiques (62%). Les principales causes de leurs pertes de points sont de deux types : soit un des deux algorithmes est mauvais et nous retrouvons les mêmes erreurs que chez ceux qui ont eu 0, soit les erreurs proviennent de la programmation. Pour les fautes de programmation, nous trouvons dans plusieurs copies l'utilisation de variables non déclarées, de mauvais appels de fonctions en Python (« count » ou « size » au lien de « len »), la présence de verbes conjugués (« if i **is** in j.knows »), un oubli du « return » à la fin de la fonction,... Une autre erreur, mais cette fois plutôt liée au problème, est que les étudiants veulent retirer une personne de la liste puisqu'une personne ne peut pas se connaître elle-même. Cependant, ils expriment mal cette idée. Nous trouvons notamment des étudiants qui retirent la dernière personne de la liste people, sauf qu'ils ne retirent alors pas nécessairement la bonne personne.
Enfin, quelques étudiants ont décidé de faciliter le travail des correcteurs en commentant leurs codes.

Terminons par une note positive : 21 étudiants en informatique (44%) et 18 en mathématiques (62%) ont réussi à obtenir plus de la moitié des points pour cette question. Nous constatons donc que cette question a été totalement ratée ou assez bien réussie, surtout en mathématiques où ceux qui n'ont pas eu 0 ont tous réussi la question.

Conclusion

Nous avons constaté, en analysant les copies des étudiants pour cette question, que bon nombre d'entre eux n'étaient pas parvenus à trouver des algorithmes corrects pour résoudre les deux problèmes. De même, aucun algorithme ne figure sur les copies, nous trouvons juste des codes et, de temps en temps, des annotations. Nous pensons que les étudiants veulent aller directement à la programmation et négligent donc la recherche des algorithmes. Un fait qui nous a d'autant plus surprise est qu'aucun étudiant n'a pris la peine de tester son code sur un exemple, ou alors cela a été effectué sur des feuilles de brouillon que nous n'avons pas eues en notre possession. Même si l'examen était écrit, rien n'empêchait les étudiants de tester leurs codes afin d'évaluer si ceux-ci tenaient la route. Nous sommes persuadée que si certains avaient pris la peine de passer par cette étape, ils se seraient aperçus que leurs codes ne répondaient pas aux problèmes.
Statistiquement, et contrairement à ce que nous pensions avant l'analyse des copies, il semble que cette question ait été mieux réussie chez les étudiants en mathématiques puisque 62% des étudiants ont plus de la moitié contre 44% en informatique.

VI.4.2 Troisième question

Question préliminaire

En ce qui concerne le problème présenté dans cette troisième question, nous nous posons la même question que précédemment : les erreurs commises par les étudiants relèvent-elles davantage de l'algorithme sous-jacent au programme à trouver ou davantage de la programmation avec toutes les subtilités du langage Python ?
Nous trouvons que cette question est nettement plus compliquée que la précédente car elle utilise la récursivité. Dans notre analyse du cours, nous avons vu que la récursivité était une notion assez compliquée car elle nécessite l'écriture d'un cas général, d'un cas de base qu'il faut atteindre à

un moment donné ainsi qu'une bonne compréhension de ce qui se produit lors d'un appel récursif.
Cette question nécessite toutefois moins l'utilisation des subtilités du Python que la précédente. Nous pensons donc qu'elle peut être davantage réussie par les étudiants en mathématiques car la récursivité se base sur le même principe que la démonstration par récurrence.

Analyse des copies

Parmi les 48 informaticiens à avoir présenté cet examen, 32 ont obtenu la note de 0 (67%). Chez les mathématiciens, sur les 29 étudiants présents, 13 ont obtenu cette note (45%). À nouveau, bon nombre de ces étudiants ont rendu une feuille blanche : 16 en informatique et 4 en mathématiques. Les autres étudiants ayant obtenu 0 sont des étudiants qui, d'une part, n'ont pas trouvé la bonne réponse au point b) et d'autre part, ne sont pas parvenus à construire correctement la fonction récursive. Dans les copies de ces étudiants, nous trouvons, par exemple, une fonction ne comportant pas d'appel récursif, une autre ne comportant pas de cas de base, une ne comportant qu'un cas de base erroné,... Pire encore, malgré qu'il soit explicitement demandé de créer une fonction récursive, des étudiants ont tenté de s'en sortir autrement, par exemple avec l'emploi de fonctions déjà créées en Python (« split », « extend »,...).

Ensuite, 9 étudiants en informatique et 10 en mathématiques n'ayant pas 0 ont des notes en dessous des 50%. Il s'agit d'étudiants ayant un cas de base correct et/ou ayant répondu convenablement au point b), mais chez qui le cas général de la fonction récursive n'est pas correct.

Enfin, les 7 informaticiens et 6 mathématiciens ayant réussi cette question ont des cotes dépassant les 80%. Ce sont des étudiants qui ont une fonction récursive correcte mais qui ont éventuellement commis des fautes de programmation. Par exemple, un étudiant utilise la fonction « count » au lieu de « len », certains oublient des initialisations, d'autres ne répondent pas au point b),...
Tout comme la question précédente, nous constatons que cette question a été totalement ratée ou très bien réussie.

Conclusion

En analysant les copies des étudiants, nous avons remarqué, une fois encore, que beaucoup d'entre eux ne parviennent pas à trouver une démarche

correcte pour répondre à cette question. De plus, certains ne respectent pas la consigne qui était d'écrire une fonction récursive et sont donc pénalisés en conséquence. À nouveau, aucun étudiant n'explique son code, à l'exception de quelques commentaires à l'occasion. De même, certains ne confrontent pas leurs programmes écrits au point a) avec les réponses qu'ils donnent au point b).
Cette question a été un peu mieux réussie en mathématiques (21% de réussite) qu'en informatique (15%), comme nous le pensions. Nous nous attendions à ce que les résultats soient moins bons par rapport à la question précédente, mais pas à ce que les taux de réussite soient aussi bas.

VI.4.3 Quatrième question

Questions préliminaires

Nous ne pensons pas que le point a) de cette question puisse poser problème aux étudiants puisqu'il s'agit de l'exécution d'un programme itératif. De plus, le but de ce dernier est donné dans l'énoncé. Nous regardons donc combien d'étudiants ont trouvé le résultat correct et d'où viennent les erreurs de ceux qui ne l'ont pas trouvé.
En ce qui concerne le point b), nous nous demandons si les étudiants sont capables de prouver la terminaison d'un programme avec une boucle « while ».
Pour le point c), il s'agit de prouver, par récurrence, que le programme fournit le résultat correct. Il n'est pas noté explicitement qu'il faut effectuer une telle preuve, mais le fait de devoir prouver qu'une telle assertion est un invariant de la boucle sous-entend qu'il faut procéder de la sorte car un invariant de boucle est une assertion vraie après chaque passage dans la boucle. Nous nous demandons donc si nous allons retrouver une preuve par récurrence dans les copies des étudiants et si oui, si cette preuve est acquise. Nous nous attendons clairement à obtenir de meilleurs résultats en mathématiques puisqu'un mois avant cet examen, lors du test de Mathématiques pour l'Informatique I, nous avons remarqué que peu d'étudiants en informatique maîtrisaient cette preuve.
Concernant le point d), il suffit de déduire un résultat sur la base des deux points précédents. Nous pensons que cette question doit être assez bien réussie dans l'ensemble car il s'agit d'utiliser des résultats déjà démontrés. Cependant, comme pour le test de Mathématiques pour l'Informatique I que nous avons analysé, nous pensons que certains étudiants risquent de ne pas répondre à cette sous-question s'ils n'ont pas répondu correctement aux

deux précédentes.
Enfin, le point e) est une étude de la complexité du programme. Aucune justification n'est demandée. Nous nous attendons à trouver un certain nombre de bonnes réponses car les étudiants ont effectué des calculs de la complexité plus compliqués dans le cours.

Analyse des copies

Au point a), seuls 2 étudiants d'informatique sur les 48 et 2 étudiants de mathématiques sur les 29 n'ont pas répondu à la question. Nous pensions que cette question n'aurait posé aucun souci aux étudiants, mais non constatons que 12 informaticiens et 2 mathématiciens se sont trompés. Dans les erreurs, nous trouvons des listes comme [0,2,5,1,7,6,2] (l'étudiant a diminué chaque valeur d'une unité sauf la dernière), [2,3,6,2,8,7,1] (l'étudiant n'a effectué la boucle qu'une seule fois), [1,2,5,1,7,6,1] (l'étudiant a diminué toutes les valeurs sauf la première),... Si ces étudiants avaient relu l'énoncé, ils se seraient peut-être aperçus que leurs listes étaient incorrectes puisque le programme permet de placer les nombres pairs avant les impairs et dans leurs listes, ils ont des impairs au milieu des pairs. Nous avons repéré d'autres erreurs, par exemple [6,2,8,2,1,3,7] (les étudiants ont décalé les pairs vers la gauche et les impairs vers la droite) et [2,2,6,8,7,3,1] (les étudiants ont classé les pairs par ordre croissant et les impairs par ordre décroissant). Ces réponses, contrairement aux précédentes, ne sont pas en contradiction avec l'énoncé en français.

Pour le point b), 7 informaticiens et 3 mathématiciens n'ont rien répondu. Bon nombre d'étudiants sont parvenus à prouver correctement la terminaison de ce programme : ils sont 24 en informatique et 12 en mathématiques. Pour cette question, deux façons de procéder étaient envisageables : utiliser une preuve par récurrence sur le nombre de passages dans la boucle comme cela est fait dans le cours ou constater qu'à chaque nouveau passage dans la boucle, soit i augmente, soit j diminue et donc, à un certain moment, nous aurons $i > j$.
Ceux qui n'ont pas tous les points à cette question sont soit des étudiants qui n'ont absolument rien prouvé, soit des étudiants chez qui il manque certains détails (oubli du cas où la liste est vide, manque de justifications, manque de liens avec le code,...). Citons quelques exemples de preuves erronées.

E_1 : Tôt ou tard, la fonction s'arrêtera.

E_2 : À l'entrée de la boucle, la première condition cherche un élément en position 0 qui n'existe pas. Une erreur est générée par Python.

E_3 : $j = j - 1$. Il y a -1 donc ça ne peut jamais aller jusqu'à l'infini.

E_4 : Pour une liste de taille $k + 1$, i vaut 1 si le nombre est pair et 0 si le nombre est impair.

Une erreur survenue à plusieurs reprises est de supposer, dans le cas général de la récurrence, que la liste change de taille à chaque passage dans la boucle, alors que la liste est fixée dès le début de la preuve.

Pour le point c), 21 étudiants en informatique et 4 en mathématiques rendent une copie blanche. Sur les étudiants restants, 9 informaticiens et 16 mathématiciens effectuent une preuve par récurrence correcte. Nous avions donc raison de penser que ce point serait mieux réussi chez les mathématiciens. Les autres étudiants se classent en deux catégories : ceux qui utilisent la preuve par récurrence et qui oublient l'une des deux étapes ou qui oublient certaines justifications et ceux qui n'utilisent pas cette preuve ou qui ne la maîtrisent absolument pas. Ces derniers, au nombre de 11 en informatique et de 5 en mathématiques, sont pénalisés d'une cote nulle. Citons quelques uns de leurs commentaires.

E_1 : On sait, par définition de process(A) que $i < j$ et tous les éléments impairs sont échangés.

E_2 : Au début de la fonction, on a $i = 0$ et $j = len(A) - 1$. L'invariant de boucle est respecté.

E_3 : Avant : $i = 0$ donc il n'y a aucun élément tel que l'indice est $< i$. Pour $j = len(A) - 1$, il n'y a aucun élément $> j$. Ce n'est ni faux, ni vrai.

Nous trouvons également un étudiant qui veut démontrer que l'hypothèse de récurrence est vraie et deux autres qui prouvent l'assertion sur la base de l'exemple donné au point a).

En ce qui concerne le point d), 21 informaticiens et 7 mathématiciens ne répondent rien. Nous nous attendions à obtenir de nombreuses abstentions pour ce point. Certains arrivent à justifier correctement que la fonction est exacte : ils sont 7 en informatique et 16 en mathématiques. Les erreurs commises par ceux qui ont perdu des points sont principalement dues au fait que ces étudiants n'exploitent pas les points b) et c) comme il l'est explicitement demandé dans l'énoncé. Certains essaient de prouver le point d) au moyen de la preuve par récurrence qu'ils auraient dû mettre au point c). Les autres oublient d'expliquer ce qui se passe à la fin de la boucle en ce qui concerne l'assertion qui est donnée au point précédent. Enfin, un étudiant a encore été prouver la propriété à partir de l'exemple du point a).

Finalement, pour le point e), seuls 7 étudiants en informatique et 6 en mathématiques ne répondent pas à la question. Les résultats sont assez bons, comme nous pouvions nous y attendre : 30 étudiants d'informatique et 21 de mathématiques trouvent la réponse exacte, $O(n)$. Rappelons que les étudiants ont travaillé sur des programmes pour lesquels la complexité était beaucoup plus compliquée à déterminer. Certains étaient proches de la réponse mais n'ont pas exactement donné l'ordre : leurs réponses sont $O(n-1)$, $O(\frac{5n}{2})$ et $O(i)$ où $i = n-1$. Enfin, les dernières réponses sont $O(n^2)$ (à plusieurs reprises), $O(\log_2 n)$, $O(1)$ et $O(0)$ (le programme ne contient pas d'instructions !).

Conclusion

Un premier constat vis-à-vis de cette question est que beaucoup d'échecs viennent du fait que les étudiants ne répondent pas à toutes les sous-questions. En informatique, seuls 21 étudiants sur les 48 notent un résultat pour chaque sous-question (44%). Ce nombre est nettement plus élevé en mathématiques puisqu'ils sont 21 étudiants sur les 29 dans le cas (72%). Pour nous, deux facteurs peuvent expliquer ce nombre élevé d'abstentions : d'une part, comme nous l'avons déjà signalé, cette question concerne les chapitres 8 et 9 du cours qui sont assez différents des autres car ils étudient l'algorithme en tant qu'objet et non plus seulement comme outil pour la résolution de problèmes. D'autre part, cette question était la dernière de l'examen. Peut-être que les étudiants n'ont pas eu le temps de s'y atteler.

Parmi ceux qui répondent aux questions, beaucoup perdent des points parce qu'ils n'utilisent pas les techniques de preuves rencontrées au cours, notamment en ce qui concerne la preuve par récurrence pour la justification du troisième point et d'autres ne respectent pas les consignes : ils n'utilisent pas les résultats qu'ils doivent exploiter, ils justifient leurs réponses lorsqu'ils ne le doivent pas,...
La preuve par récurrence ne semble pas encore être acquise par la majorité des étudiants : beaucoup oublient de mentionner le cas de base ou une partie du cas général et d'autres n'utilisent pas l'hypothèse de récurrence ou tentent de la démontrer.
La complexité semble être maîtrisée par une bonne partie des étudiants. Ceci dit, ce résultat est à nuancer car le calcul à effectuer n'était pas des plus complexes et aucune justification n'était à fournir. Nous ne savons donc pas si les réponses sont le fruit du hasard ou d'une réelle réflexion.

Terminons ce paragraphe consacré à la quatrième question de cet examen en donnant les taux de réussite dans les deux sections : il est de 31% en informatique et de 69% en mathématiques, ce qui reflète bien le fait que cette question portait sur les chapitres plus mathématiques de ce cours.

VI.5 Avis des étudiants après l'examen

Questions posées

Quelques semaines après l'examen de janvier 2014 du cours de Programmation et Algorithmique I, nous avons proposé un questionnaire aux étudiants d'informatique et de mathématiques ayant suivi ce cours. Nous leur avons posé trois questions et nous indiquons, au vu de l'examen, les tendances que nous pensons trouver dans leurs copies.

1. Expliquez quelles sont les difficultés que vous avez rencontrées dans ce cours.
   → Nous nous attendons à trouver les trois axes de difficultés que nous avons repérés après l'analyse du cours, à savoir : l'écriture d'algorithmes, les subtilités du langage Python et la preuve et la complexité. En effet, nous avons constaté qu'à l'examen, beaucoup d'étudiants ont écopé d'une cote nulle aux questions 2 et 3 parce qu'ils ne sont pas parvenus à trouver des algorithmes corrects pour résoudre les problèmes. Beaucoup d'étudiants ont également perdu des points parce qu'ils ne connaissaient pas bien toutes les fonctions du Python (utilisation de « size » au lieu de « len », par exemple). Ils risquent donc de parler du langage Python dans les difficultés rencontrées. Enfin, la dernière question de l'examen concernait la preuve et la complexité d'un algorithme. Comme cette question n'a pas été particulièrement bien réussie, surtout en informatique, nous nous attendons à ce que des étudiants en parlent.
2. Selon vous, quelle est la partie du cours la plus difficile ? Pourquoi ?
   → Nous pensons que les étudiants en informatique vont citer les chapitres concernant la preuve et la complexité car ce sont des chapitres plutôt mathématiques. En effet, le reste du cours étant plutôt consacré à la programmation, nous croyons que ces étudiants vont citer ces chapitres car ils sont assez différents des autres et les questions sur ces chapitres ne se résolvent pas au moyen d'un programme à écrire. Ils nécessitent plus une démarche mathématique qu'une démarche informatique.

Quant aux mathématiciens, nous nous attendons à ce qu'ils mentionnent un chapitre concernant le Python. Effectivement, nous ne pensons pas qu'ils parleront des chapitres sur la preuve et la complexité car ils sont assez mathématiques et donc plus accessibles pour eux. Ils vont donc parler d'une autre partie du cours. Comme le cours contient principalement des chapitres liés à l'apprentissage du Python, ils risquent de citer l'un ou l'autre chapitre.
Il est également fort probable que les étudiants citent la récursivité compte tenu de l'examen. En effet, la récursivité était l'objet de la troisième question et nous avons constaté que les taux de réussite étaient assez bas pour cette question parce que les étudiants ne sont pas parvenus à écrire une fonction récursive.

3. Expliquez ce qui est, selon vous, important dans ce cours.
   → Nous pensons que les étudiants vont citer l'écriture d'algorithmes, la rigueur et une bonne connaissance du langage Python.
   Nous pensons que les étudiants vont parler de l'écriture d'algorithmes car souvent, s'ils ne trouvent pas un algorithme correct pour résoudre le problème proposé, ils ne sont alors pas capables d'écrire un programme résolvant ce problème. Nous nous attendons à ce qu'ils parlent de la rigueur car le moindre double point ou la moindre initialisation manquants peuvent faire qu'un programme ne fonctionne pas. De même, justifier une preuve de terminaison ou de correction requiert de la rigueur et de la précision. Enfin, ils risquent également de parler du fait que ce cours exige une bonne connaissance du Python car s'ils ne connaissaient pas certaines structures (les boucles, par exemple), ils ne seraient tout simplement pas capables d'écrire certains programmes.

Réponses des étudiants

Les étudiants ayant répondu à notre questionnaire sont au nombre de 41 en informatique et de 27 en mathématiques. Nous regardons les réponses des étudiants question par question.

Expliquez quelles sont les difficultés que vous avez rencontrées dans ce cours.

Nous nous attendions à ce que les étudiants parlent de l'écriture d'algorithmes et effectivement, 13 informaticiens (32%) et 9 mathématiciens (33%) disent avoir eu des difficultés à concevoir un algorithme permettant de résoudre un problème donné. Ils ne trouvent pas les différentes étapes, ne savent pas comment aborder les problèmes, ont eu du mal à attraper la

logique de résolution de problèmes,...
En ce qui concerne le Python, 18 étudiants d'informatique (44%) et 10 de mathématiques (37%) le citent dans les difficultés rencontrées. Ils ont eu du mal à s'adapter à ce langage à cause des mots en anglais, des nombreuses fonctions du Python et à connaître pour l'examen, ou encore parce qu'ils avaient déjà programmé dans d'autres langages avec des syntaxes différentes.
Enfin, en ce qui concerne la preuve et la complexité, 4 étudiants en informatique (10%) soulèvent ce point car ils estiment notamment qu'il n'y a pas eu assez de pratique sur les preuves. Aucun étudiant de mathématiques n'a parlé de ces deux matières.

Trois informaticiens (7%) et 5 mathématiciens (19%) ont éprouvé des difficultés avec le fait que l'examen était sur papier alors qu'habituellement, ils travaillaient toujours sur un ordinateur. Enfin, 4 étudiants en informatique (10%) et 3 étudiants en mathématiques (11%) disent que cette matière est tout à fait nouvelle lors de l'entrée à l'université et ils trouvent que ce cours s'adressait plutôt à des personnes ayant déjà des bases en programmation.

Selon vous, quelle est la partie du cours la plus difficile ? Pourquoi ?

Des étudiants citent les chapitres sur la preuve et la complexité : ils sont 18 en informatique (44%) et seulement 1 en mathématiques (4%). Ils trouvent ces matières compliquées, abstraites et n'en ont pas toujours compris l'utilité.
En ce qui concerne les différents chapitres plutôt relatifs au langage Python, 11 informaticiens (27%) et 10 mathématiciens (37%) pointent le dernier chapitre consacré aux structures de données ; 5 informaticiens (12%) et 10 mathématiciens (37%) citent, quant à eux, l'avant-dernier chapitre sur les objets et la programmation orientée objet. Beaucoup trouvent ces chapitres difficiles car il faut avoir bien assimilé toute la matière qui précède pour les comprendre.
La récursivité a été citée dans 6 copies d'informaticiens (15%) et dans 7 de mathématiciens (26%). Ils trouvent qu'il est difficile d'évaluer ce que l'algorithme fait vraiment et que même s'ils comprennent le principe de la récursivité, ils ne parviennent pas à créer une fonction l'utilisant.
Enfin, des étudiants citent d'autres parties du cours comme les fichiers, les exceptions, les boucles,... Ils sont 2 en informatique (5%) et 5 en mathématiques (19%).

Expliquez ce qui est, selon vous, important dans ce cours.

Au final, seuls 11 étudiants en informatique (27%) et 6 en mathématiques (22%) estiment que le plus important dans ce cours est l'écriture d'algorithmes. Certains trouvent que c'est une compétence plus importante que l'écriture d'un code parce qu'avant d'écrire un programme, il faut savoir quelles sont les étapes à y intégrer.
En ce qui concerne la programmation, 10 informaticiens (24%) et 12 mathématiciens (44%) citent ce point car ils estiment que ce premier contact avec l'informatique est important.
Les autres étudiants citent des chapitres spécifiques du cours ou un travail régulier. Aucun étudiant ne fait référence à la rigueur.

Enfin, pour cette dernière question, nous nous sommes aperçue que certains termes étaient récurrents dans les copies des étudiants. En effet, nous avons repéré 28 fois le verbe « comprendre » (41% des copies), 25 allusions à des savoir-faire (37%) et 14 références à la connaissance de la matière (21%). En synthétisant le tout, nous en déduisons que les points importants de ce cours sont d'étudier la matière, de s'entraîner régulièrement et de comprendre toute la matière et pourquoi nous procédons de telle ou telle manière.

VI.6 Bilan

Dans ce chapitre, nous avons constaté que l'écriture d'un algorithme pour résoudre un problème était une compétence loin d'être acquise chez les étudiants de première année. En effet, en entrant à l'université, les étudiants ont beaucoup de mal à trouver une démarche correcte dans le but de résoudre un problème, même si elle ne nécessite que des structures de base en algorithmique (structure itérative et structure conditionnelle) qui sont également utilisées dans des cours de mathématiques dans le secondaire. Après avoir suivi un cours d'algorithmique pendant plusieurs mois, l'écriture d'un algorithme n'est toujours pas acquise par une bonne partie des étudiants. Ces étudiants essaient directement d'écrire un programme informatique sans avoir réfléchi au préalable à l'algorithme sous-jacent permettant de résoudre le problème posé et écrivent donc des programmes n'apportant aucune solution. Enfin, nous avons remarqué que l'écriture d'un algorithme récursif posait énormément de problèmes aux étudiants. Ceci nous conforte dans notre choix de ne pas étudier d'algorithmes récursifs dans le cadre de notre séquence d'enseignement destinée à des élèves du secondaire.

Trouver un algorithme résolvant un problème donné n'est parfois pas suffisant pour répondre complètement à des questions pour lesquelles l'écriture d'un programme informatique est exigée. Nous avons effectivement constaté qu'un certain nombre d'étudiants ne maîtrisaient pas complètement le langage étudié dans le cours de Programmation et Algorithmique I et étaient donc bloqués à certains moments dans l'écriture de leurs codes lors de l'examen.

Enfin, la preuve et la complexité, parties plus mathématiques de l'algorithmique, sont, sans réelle surprise, davantage maîtrisées par les étudiants en mathématiques. Bon nombre d'étudiants ne parviennent pas encore, après plusieurs cours sur le sujet, à effectuer une preuve par récurrence correctement. Les uns oublient une partie de la preuve tandis que d'autres démontrent les hypothèses de récurrence et que les derniers tentent d'effectuer une preuve d'un autre type, malgré les demandes plus ou moins explicites d'employer la preuve par récurrence. Nous remarquons donc que cette preuve pose énormément de problèmes aux étudiants universitaires et nous supposons alors que la situation ne sera guère meilleure dans l'enseignement secondaire. Cependant, nous trouvons que c'est une preuve intéressante et abordable avec des adolescents. Nous pensons donc qu'elle devrait figurer dans notre séquence de cours.
En ce qui concerne la complexité, seule une sous-question d'un questionnaire y faisait référence. Néanmoins, nous n'avons pas trouvé cette question intéressante car le calcul nous paraissait trop simple par rapport à ce qui avait été fait en classe et aucune justification n'était demandée, ce qui fait que nous ne savons pas pourquoi les étudiants ont donné telle ou telle réponse.

Finalement, les étudiants sont assez conscients de leurs difficultés puisque quand nous leur demandons les difficultés rencontrées dans le cours de Programmation et Algorithmique I, ils citent les points que nous venons de développer.
Certains étudiants ont trouvé ce cours difficile car il ne se rapportait à aucune matière vue dans le secondaire et était donc totalement nouveau à leur entrée à l'université.

Dans notre prochain chapitre, nous concevons notre séquence d'enseignement pour des élèves de l'enseignement secondaire dans l'optique qu'ils sont susceptibles de suivre, à leur tour, un cours d'algorithmique dans de futures études supérieures. Nous allons donc être davantage vigilante quant à ce qui pose problème aux étudiants universitaires en détaillant plus amplement ces différents points.

Chapitre VII

Élaboration d'une séquence de cours sur l'algorithmique

Suite aux conclusions tirées dans les chapitres précédents, nous avons élaboré une séquence de cours sur l'algorithmique en vue d'une expérimentation dans le secondaire supérieur belge. Nous décrivons la construction de cette séquence dans ce chapitre.

VII.1 Considérations générales

Le but de cette séquence de cours est de travailler, au moyen d'éléments d'algorithmique, des notions mathématiques et des compétences transversales citées dans les chapitres antérieurs avec des élèves de l'enseignement secondaire. Cependant, comme aucune séquence d'initiation à l'algorithmique n'est prévue par les programmes belges, nous devons tout reprendre depuis le début (entrée, sortie, boucles,...) avant de pouvoir nous atteler à des aspects plus mathématiques (preuve,...).

Nous avons choisi d'exprimer les différents algorithmes en pseudo-code avec la syntaxe du cours de Programmation et Algorithmique I pour plusieurs raisons. La première est qu'elle est en français. En effet, tous les élèves du secondaire n'ont pas forcément un cours d'anglais et ils auront peut-être plus facile à retenir et à comprendre les différentes structures si elles sont proches de structures qu'ils peuvent rencontrer lorsqu'ils parlent. La seconde est qu'ainsi, nous ne privilégions aucun langage de programmation. Effectivement, tous les cours de programmation donnés dans les études supérieures n'emploient pas le même langage. De même, il est

plus commode d'apprendre l'algorithmique à l'aide d'un pseudo-code qu'à l'aide d'un langage de programmation pour lequel il faudrait expliquer les subtilités. Nous avons effectivement constaté que chez les étudiants universitaires, beaucoup d'erreurs venaient d'une mauvaise connaissance du langage Python. Rappelons également que notre séquence de cours se place dans le cadre d'un cours de mathématiques et non dans le cadre d'un cours d'informatique. D'ailleurs, le site Internet de la « Computer Science Unplugged[1] » part du principe qu'il est possible d'enseigner des concepts d'informatique sans avoir recours à l'ordinateur.
Il y a encore d'autres raisons pour lesquelles nous avons choisi cette syntaxe. Nous les expliquerons au moment opportun.

Pour concevoir cette séquence d'enseignement, nous nous sommes basée sur les travaux de Modeste et Aldon, sur les cours de Mélot (2013), Vanhoof (2009-2010) et Füzfa (2011-2012) ainsi que sur les questionnaires de l'Olympiade belge d'Informatique.
L'Olympiade belge d'Informatique est une compétition de programmation, d'algorithmique et de logique à laquelle les étudiants du secondaire peuvent participer. Dans les questionnaires, nous trouvons un « aide-mémoire de pseudo-code » que nous avons regardé afin de voir quelles notions sont utiles dans la compréhension et l'écriture d'algorithmes.

Nous décrivons donc le contenu de cette séquence de cours en expliquant les différents choix que nous effectuons. Cette séquence d'enseignement est divisée en cinq parties :
- introduction,
- quelques notions utiles pour comprendre un algorithme,
- preuve d'un algorithme,
- complexité,
- le problème du site de rencontres.

La dernière partie a été ajoutée aux quatre parties décrites dans le chapitre V. Le document préparé pour les élèves se trouve à l'annexe C.

VII.2 Introduction

La première partie de notre séquence d'enseignement consiste en une introduction qui a pour but de montrer aux élèves qu'en réalité, ils connaissent des algorithmes et qu'ils en rencontrent dans la vie quotidienne. À partir de différents exemples connus des élèves, nous faisons émerger la définition

1. http://www.csunplugged.org

d'« algorithme ».

Concrètement, les deux exemples que nous proposons aux élèves sont l'utilisation d'un micro-ondes et le démarrage d'une voiture à clé. Nous avons choisi ces deux exemples car ils sont susceptibles de parler aux élèves. En effet, l'utilisation d'un micro-ondes risque de leur être bien utile durant leurs années d'études supérieures et certains élèves sont sûrement en apprentissage pour le permis de conduire.
Lorsque nous désirons mettre chauffer un plat au micro-ondes, nous procédons comme suit :

Algorithme 1 (Utilisation d'un micro-ondes)

1. Mettre le plat dans le micro-ondes.
2. Fermer la porte du micro-ondes.
3. Choisir la puissance et le temps de chauffe.
4. Mettre en route le micro-ondes.
5. Attendre que le temps se soit écoulé.
6. Ouvrir la porte du micro-ondes.
7. Prendre son plat.

Pour le démarrage de la voiture, la démarche à suivre est un peu plus compliquée :

Algorithme 2 (Démarrage d'une voiture à clé)

1. Insérer la clé dans le contacteur à clé.
2. S'assurer que le levier de vitesses se trouve en position neutre (point mort).
3. Mettre le contact.
4. Si le moteur part en moins de six secondes, relâcher la clé.
5. Si le moteur ne part pas en six secondes :
   - relâcher la clé,
   - attendre 10 secondes,
   - répéter les étapes 3, 4 et 5, mais pas plus de 5 fois en tout.
6. Si le moteur ne part pas, appeler le garage.

Sur la base de ces deux exemples, nous sommes en mesure de déterminer les éléments caractéristiques d'un algorithme. Tout d'abord, nous pouvons constater qu'un algorithme sert à résoudre un problème. Dans le cas du micro-ondes, le problème est de faire chauffer un plat et dans le cas de la voiture, de faire démarrer le moteur. C'est l'aspect « problème » de Modeste. En outre, un algorithme doit être général, il ne peut pas fonctionner uniquement pour un cas particulier. Rappelons que Modeste a identifié, dans les manuels français, de nombreux algorithmes qui n'en étaient pas car ils n'étaient pas généraux. Dans nos exemples, les procédures sont générales, elles sont valables pour résoudre une classe de problèmes (une famille d'instances).
Une dernière caractéristique est que le nombre d'étapes exécutées par chaque algorithme est fini (terminaison de l'algorithme). En effet, pour l'algorithme du micro-ondes, c'est clairement le cas puisque chaque opération n'est répétée qu'une seule fois. Il contient donc 7 étapes. Dans le cas de la voiture, c'est un peu plus compliqué car les opérations 3, 4 et 5 peuvent être répétées plusieurs fois. Cependant, comme après un certain nombre d'essais, nous appelons le garage, nous sortons donc de l'algorithme.

Les caractéristiques d'un algorithme mises en exergue, nous pouvons donner une définition d'« algorithme ». Néanmoins, nous avons besoin de deux mots de vocabulaire déjà rencontrés dans les travaux de Modeste, mais que nous définissons avec nos propres mots : *famille d'instances* et *instance*.

- ⋆ Une **famille d'instances** d'un problème est l'ensemble des configurations possibles du problème.
- ⋆ Une **instance** est un cas particulier d'une configuration du problème.

Après avoir regardé plusieurs définitions du terme « algorithme » dans les différentes ressources citées plus haut, nous choisissons comme définition celle présente dans le cours de Füzfa car c'est celle qui nous semble la plus complète et la plus intelligible pour des élèves débutant en algorithmique - celle de Modeste nous paraît très complète, mais elle est un peu longue et compliquée à comprendre. Nous modifions légèrement cette définition afin d'y faire apparaître le fait qu'un algorithme est valable pour une famille d'instances. Notre définition est donc la suivante :

> Un **algorithme** est une suite d'opérations qui, effectuées comme une séquence déterminée, fournissent, en un nombre fini d'étapes, la solution à un problème pour toute instance de celui-ci.

Notons également que l'exemple typique pour expliquer ce qu'est un algorithme est la recette de cuisine. Nous avons volontairement omis cet

exemple car la recette de cuisine n'est en fait pas un algorithme puisqu'elle n'est valable que pour un cas particulier et non pour un ensemble de configurations possibles.

La définition d'algorithme posée, nous proposons ensuite aux élèves des algorithmes que nous rencontrons en mathématiques. Rappelons que notre séquence de cours doit s'inscrire dans le cadre d'un cours de mathématiques. Les deux algorithmes proposés sont la division euclidienne par soustractions et la résolution de l'équation du second degré.
Pour la division euclidienne, nous commençons par rappeler comment calculer le quotient et le reste de la division euclidienne de 17 par 5. Nous expliquons comment les trouver à l'aide de la potence. Nous trouvons que le quotient vaut 3 et le reste 2. De là, nous rappelons qu'avant de pouvoir effectuer une division, nous procédions par soustractions pour voir combien de fois 5 pouvait se mettre dans 17.
Nous effectuons la démarche suivante :

$$\begin{aligned} 17-5 &= 12 \\ 12-5 &= 7 \\ 7-5 &= 2 \end{aligned}$$

Comme l'opération est répétée 3 fois, le quotient vaut 3 et ce qui reste à la fin, lorsque la valeur est plus petite que le diviseur, c'est le reste, qui est de 2 ici.
L'algorithme correspondant à cette démarche est le suivant :

Algorithme 3 (Division euclidienne)

Entrée : deux nombres naturels a et b, avec $b \neq 0$.
Sortie : deux nombres naturels q et r, tels que q est le quotient de a par b et r, le reste de cette division.

1. $r \leftarrow a$
2. $q \leftarrow 0$
3. **tant que** $r \geqslant b$ **faire**
4. $\quad r \leftarrow r - b$
5. $\quad q \leftarrow q + 1$
6. **fin tant que**
7. **retourner** q et r

Cet algorithme traduit bien la démarche que nous venons d'expliquer. En effet, dans l'entrée (situation initiale), nous trouvons bien que l'algorithme doit être valable pour deux naturels a et b, le deuxième devant être non nul (diviseur).
Dans la sortie (situation finale), nous trouvons la solution de notre problème : le quotient et le reste de la division euclidienne de a par b.
La démarche consistant à soustraire le diviseur autant de fois que nécessaire est bien représentée. Effectivement, l'opération $r \leftarrow r - b$ permet de soustraire le diviseur de ce qu'il reste du dividende. Cette opération est une affectation. Nous expliquons par après pourquoi nous avons choisi de la noter $\leftarrow$, comme dans le cours de Programmation et Algorithmique I. À chaque fois que nous effectuons cette opération, nous augmentons le quotient d'une unité ($q \leftarrow q + 1$). Nous répétons ces deux opérations jusqu'à ce que le reste soit plus petit que le diviseur (la condition $r \geqslant b$ n'est plus vraie). Pour ce faire, nous utilisons donc une boucle « tant que ».

Il est ensuite utile d'exécuter cet algorithme pour les valeurs de 17 et 5 afin de bien voir que la démarche de l'algorithme est la même que celle « sur papier » présentée plus haut.

Nous donnons également un autre exemple bien connu des élèves à partir de la quatrième secondaire : la résolution de l'équation du second degré. Pour résoudre une équation du second degré $ax^2 + bx + c = 0$, nous commençons par calculer le discriminant $d = b^2 - 4ac$. Les solutions dépendent ensuite de sa valeur.
S'il est strictement positif ($d > 0$), alors il existe deux solutions qui sont $x_1 = \frac{-b+\sqrt{d}}{2a}$ et $x_2 = \frac{-b-\sqrt{d}}{2a}$.
Sinon, s'il est nul ($d = 0$), alors la solution est unique : $x_1 = \frac{-b}{2a}$.
Dans le dernier cas, s'il est strictement négatif ($d < 0$), l'équation n'admet pas de solutions dans les réels.
L'algorithme suivant (le 4) permet de résoudre une telle équation suivant la démarche venant d'être décrite. Il permet d'introduire la notion de « si...alors...sinon ».
« Maîtriser le connecteur si...alors » fait d'ailleurs partie des compétences transversales à acquérir en fin de secondaire.

De même, nous exécutons cet algorithme avec différentes valeurs afin de tester les trois différents cas.

Algorithme 4 (Résolution de l'équation du second degré)

Entrée : trois nombres réels a, b et c, avec $a \neq 0$.
Sortie : deux réels x_1 et x_2, solutions de l'équation $ax^2 + bx + c = 0$ (ou un réel x_1 ou rien).

1. $d \leftarrow b^2 - 4.a.c$
2. **si** $d > 0$ **alors**
3. $\quad x_1 \leftarrow \frac{-b+\sqrt{d}}{2.a}$
4. $\quad x_2 \leftarrow \frac{-b-\sqrt{d}}{2.a}$
5. $\quad$ **retourner** x_1 et x_2
6. **sinon**
7. $\quad$ **si** $d = 0$ **alors**
8. $\quad\quad x_1 \leftarrow \frac{-b}{2.a}$
9. $\quad\quad$ **retourner** x_1
10. $\quad$ **fin si**
11. **fin si**

VII.3 Quelques notions utiles pour comprendre un algorithme

Dans cette partie, nous rediscutons des différentes notions rencontrées dans les exemples introductifs afin de les formaliser.

VII.3.1 Entrées et sorties

La première chose à faire lorsqu'il s'agit d'écrire un algorithme est d'indiquer les entrées et les sorties afin de voir de quoi nous partons et vers quoi nous voulons aller.
Les entrées représentent les instances du problème ou encore les données initiales. Dans la vie quotidienne, elles peuvent être vues comme la situation initiale. Pour le problème du micro-ondes, l'entrée est le fait que le plat est froid.

Les sorties représentent les résultats ou les données finales. Dans la vie quotidienne, elles peuvent être assimilées à la situation finale. Pour l'exemple du micro-ondes, la sortie est le fait que le plat est réchauffé.
Vanhoof résume le concept d'algorithme dans la vie quotidienne et en informatique comme illustré à la FIGURE VII.1.

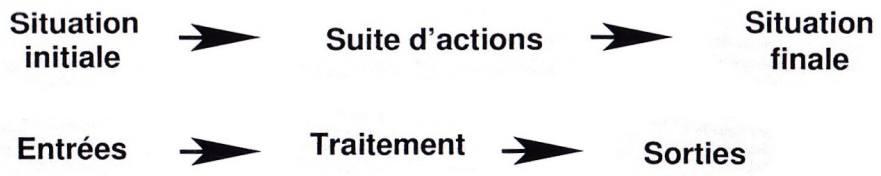


FIGURE VII.1 – L'algorithme dans la vie quotidienne et en informatique

VII.3.2 Affectation

L'affectation est une opération permettant d'assigner une valeur à une variable. Nous en avons déjà rencontré précédemment. Nous l'avons notée « $\leftarrow$ » suivant la syntaxe de Mélot afin de bien la distinguer d'une égalité. En effet, comme le souligne Nguyen (2005) dans sa thèse, l'affectation n'est pas une égalité et la nuance est parfois subtile à comprendre. L'affectation est une opération qui pose souvent problème aux étudiants universitaires car quand ils travaillent avec une variable mathématique, la valeur est fixée au début de l'exercice. Dans le cas d'un algorithme, lors d'une affectation, la valeur de la variable est en général modifiée. Nous parlons alors de « variable informatique » pour bien la distinguer de la variable mathématique.

Une différence entre l'affectation et l'égalité est que l'égalité est une opération symétrique, alors que l'affectation n'en est pas une. Illustrons ce propos à l'aide d'exemples.
Si, dans un exercice de mathématiques, à un moment donné nous avons $x = a$, dès que nous voyons un objet x, nous pouvons le remplacer par a et inversement.
Dans un algorithme, après l'affectation $x \leftarrow a$, si nous demandons la valeur de x, nous obtiendrons la valeur a, mais si nous demandons la valeur de a, nous aurons a. Nous ne pouvons donc pas dire que a est égal à x puisqu'il n'est pas permis de remplacer toutes les occurrences de a par x. Avec l'exemple de la calculatrice, cela est plus facile à comprendre. Imaginons que nous stockions dans la variable x la valeur a. Si nous appuyons sur la touche x, la

calculatrice va nous donner a comme résultat. A contrario, si nous appuyons sur la touche a, la calculatrice nous donnera a comme résultat et non x.

VII.3.3 « Si...alors...sinon »

Afin d'introduire le formalisme pour la structure du « si...alors...sinon » déjà rencontrée dans les exemples introductifs, nous proposons un problème assez simple aux élèves. Le problème est le suivant : nous souhaitons écrire un algorithme qui calcule la valeur absolue d'un nombre réel. Cet algorithme doit fonctionner quel que soit le réel reçu.
Il n'est donc pas question de créer un algorithme permettant de calculer la valeur absolue d'un réel positif et d'en créer un autre pour le calcul de la valeur absolue d'un réel négatif.

La solution à ce problème est donc d'utiliser la structure conditionnelle « si...alors...sinon ». Formellement, cette structure s'exprime comme suit :

si condition **alors**
    instructions 1
sinon
    instructions 2
fin si

Pour exécuter une telle structure, il faut d'abord se demander si la condition est vraie ou fausse. Si elle est vraie, c'est alors le bloc « instructions 1 » qui est exécuté. Le cas échéant, c'est le bloc « instructions 2 » qui l'est.

Notons également qu'il est possible, au sein d'un premier « si...alors...sinon », d'en insérer un autre, comme dans l'exemple de la résolution de l'équation du second degré. Nous ne le ferons pas dans la suite de cette séquence afin de ne pas trop compliquer les démarches à suivre.

Finalement, nous pouvons donner un algorithme correspondant à notre problème de départ, à savoir le calcul de la valeur absolue d'un nombre réel. À nouveau, différents exemples peuvent être donnés afin de convaincre le public que l'algorithme produit bien le résultat escompté (réel positif, réel nul et réel négatif).

Algorithme 5 (Valeur absolue d'un réel)

Entrée : un nombre réel x.
Sortie : abs, valeur absolue de x.

1. **si** $x \geqslant 0$ **alors**
2. $\quad abs \leftarrow x$
3. **sinon**
4. $\quad abs \leftarrow -x$
5. **fin si**
6. **retourner** *abs*

VII.3.4 Boucle « pour tout »

Nous avons déjà rencontré une boucle dans l'introduction : la boucle « tant que ». Ici, nous en présentons un deuxième type : la boucle « pour tout ». Comme pour la notion de « si...alors...sinon », nous introduisons cette notion à l'aide d'un problème afin de faire émerger la construction de cette structure. De plus, ainsi, nous ferons sentir aux élèves la nécessité d'introduire cette structure.
Notre problème est le suivant : nous souhaitons créer un algorithme qui calcule la somme des n premiers naturels. Nous supposons, bien évidemment, que nous ne connaissons pas la formule permettant de calculer cette somme.

La solution à ce problème est donc d'utiliser une boucle « pour tout ». Formellement, elle s'exprime de la sorte :

pour tout variable allant de *val1* à *val2* **faire**
    instructions
fin pour tout

Pour exécuter une telle boucle, nous commençons par attribuer la valeur « val1 » à la variable et nous exécutons une première fois le bloc « instructions ». Ensuite, nous augmentons de une unité la valeur de la variable et nous exécutons à nouveau le bloc d'instructions. Nous recommençons ce processus autant de fois que nécessaire jusqu'à ce que la variable atteigne la valeur « val2 ». Le bloc « instructions » est alors exécuté pour la dernière fois.

Nous avons volontairement appelé cette boucle la « boucle pour tout », malgré que les cours consultés l'appellent « boucle pour ». En choisissant ce nom, nous insistons sur le fait que la variable prend toutes les valeurs entre « val1 » et « val2 ». De même, nous prenons la convention que si « val2 » est strictement inférieure à « val1 », le corps de la boucle n'est pas exécuté.
Il faut reconnaître que dans certains langages de programmation (C, Matlab,...), il est possible de choisir la valeur du pas. Cependant, ce n'est pas le cas dans tous les langages (Pascal,...). Nous avons donc décidé de choisir, pour notre boucle « pour tout », un pas fixe de $+1$. Si besoin est d'utiliser un pas d'une autre valeur, il existe un deuxième type de boucle pour y parvenir.

Finalement, nous pouvons donner l'algorithme permettant de résoudre notre problème de départ :

Algorithme 6 (Somme des premiers naturels (version 1))

Entrée : un nombre naturel n.
Sortie : la somme des n premiers naturels.

1. $somme \leftarrow 0$
2. **pour tout** i allant de 1 à n **faire**
3. $\quad somme \leftarrow somme + i$
4. **fin pour tout**
5. **retourner** *somme*

Dans cet algorithme, il ne faut pas oublier d'initialiser la variable « somme » au début ($somme \leftarrow 0$) car quand elle est utilisée la première fois dans la boucle, il faut qu'elle contienne une valeur.

VII.3.5 Boucle « tant que »

Dans nos exemples introductifs, nous avons rencontré un autre type de boucle : la boucle « tant que ». Nous résolvons à nouveau le problème proposé pour la boucle « pour tout », mais en utilisant cette fois une boucle « tant que ». Formellement, cette boucle s'exprime de la sorte :

tant que condition **faire**
    instructions
fin tant que

Pour exécuter cette boucle, nous devons tout d'abord nous poser la question de savoir si la condition est vraie ou fausse. Si elle est vraie, nous devons exécuter le bloc « instructions ». Si elle est fausse, nous sortons de la boucle. Cette boucle est exécutée jusqu'à ce que la condition devienne fausse.

La principale différence par rapport à la boucle « pour tout » est que, dans une boucle « tant que », aucune variable n'a sa valeur augmentée automatiquement à chaque nouveau passage dans la boucle. Si la valeur d'une variable doit augmenter à chaque nouveau passage, nous devons l'indiquer dans les « instructions ».

Nous pouvons donner un algorithme permettant de calculer la somme des n premiers naturels à l'aide d'une boucle « tant que » :

Algorithme 7 (Somme des premiers naturels (version 2))

Entrée : un nombre naturel n.
Sortie : la somme des n premiers naturels.

1. $somme \leftarrow 0$
2. $i \leftarrow 1$
3. **tant que** $i \neq n + 1$ **faire**
4. $\quad somme \leftarrow somme + i$
5. $\quad i \leftarrow i + 1$
6. **fin tant que**
7. **retourner** *somme*

Nous constatons donc qu'il faut ajouter une opération pour augmenter la valeur de la variable i ($i \leftarrow i + 1$), opération qui n'est pas nécessaire dans une boucle « pour tout ». Nous voyons qu'il est tout à fait possible de transformer une boucle « pour tout » en une boucle « tant que » en ajoutant une opération pour augmenter la valeur de la variable compteur. À l'inverse, transformer une boucle « tant que » en une boucle « pour tout » n'est pas toujours possible. Pensons notamment à l'exemple de la division euclidienne par soustractions. Comme nous ne savons pas déterminer a priori le nombre de fois que la boucle est exécutée, il n'est pas possible de la transformer en une boucle « pour tout » pour laquelle le nombre d'exécutions de la boucle est fixé dès le début de l'exécution de l'algorithme.

VII.3.6 Exercices

Finalement, les notions pour comprendre un algorithme étant étudiées, nous proposons aux élèves des exercices afin d'exécuter des algorithmes les faisant intervenir.
Chaque algorithme comporte une seule structure (conditionnelle ou itérative) afin de voir si ces différentes structures sont bien comprises. Le but de ces exercices est de trouver la sortie des trois algorithmes en ayant l'entrée et le corps de l'algorithme. Les élèves doivent donc exécuter ces différents algorithmes avec des valeurs appartenant aux entrées.
Sur la base de ces exécutions, ils généralisent leurs cas particuliers dans le but de trouver la sortie générale de chaque algorithme.

« Formuler des généralisations et en contrôler la validité » fait partie des compétences transversales à développer dans l'enseignement secondaire belge. Le contrôle de la validité se fera dans la partie suivante.

Les trois algorithmes proposés aux élèves sont les algorithmes 8, 9 et 10. Pour l'algorithme 9, nous avons volontairement exclu la valeur $x = 0$ afin d'éviter d'avoir le cas 0^0 qui est un cas d'indétermination.

Algorithme 8 (Maximum entre deux valeurs)

Entrée : deux réels x et y.
Sortie : ? ? ?

1. **si** $x \geqslant y$ **alors**
2. $\quad$ *valeur* $\leftarrow x$
3. **sinon**
4. $\quad$ *valeur* $\leftarrow y$
5. **fin si**
6. **retourner** *valeur*

Algorithme 9 (Puissance d'un nombre réel non nul)

Entrée : un nombre réel non nul x et un nombre naturel n.
Sortie : ? ? ?

1. $y \leftarrow 1$
2. **pour tout** k allant de 1 à n **faire**
3. $\quad y \leftarrow y \times x$
4. **fin pour tout**
5. **retourner** y

Algorithme 10 (Plus grand commun diviseur de deux naturels)

Entrée : deux nombres naturels non nuls a et b.
Sortie : ? ? ?

1. $u \leftarrow a$
2. $v \leftarrow b$
3. **tant que** $v \neq 0$ **faire**
4. $\quad z \leftarrow v$
5. $\quad q \leftarrow E(\frac{u}{v})$
6. $\quad v \leftarrow u - q \times v$
7. $\quad u \leftarrow z$
8. **fin tant que**
9. **retourner** u

Remarque : $E(x)$ désigne la partie entière du réel x. La partie entière d'un nombre réel x est l'unique nombre entier tel que $E(x) \leqslant x < E(x) + 1$.
Par exemple, $E(2,3) = 2$ car $2 \leqslant 2,3 < 3$.

VII.4 Preuve d'un algorithme

Comme signalé au paragraphe précédent, il faut être capable de généraliser sur la base d'exemples, mais ensuite, il faut contrôler la validité de ces généralisations. Nous devons donc montrer que ces algorithmes donnent

toujours les résultats escomptés, quelles que soient les entrées. Prouver qu'un algorithme donne le résultat attendu consiste à en prouver la **correction**.
De même, rappelons-nous que dans la définition d'« algorithme » donnée, nous avons précisé que le nombre d'étapes devait être fini. Or, nous avons donné, dans les paragraphes précédents, plusieurs algorithmes sans avoir vérifié qu'ils satisfaisaient à la définition. Nous devons donc vérifier que le nombre d'étapes est fini, ce qui consiste à prouver la **terminaison** de ces algorithmes.
Les algorithmes sur lesquels nous travaillons sont les algorithmes proposés dans les exercices, c'est-à-dire les algorithmes 8, 9 et 10.

VII.4.1 Terminaison

Prouver que le nombre d'étapes dans un algorithme est fini peut parfois être presque trivial, mais dans certains cas, cela nécessite une réflexion plus approfondie.

Pour l'algorithme 8, la terminaison est immédiate étant donné que soit nous exécutons l'opération 2, soit la 4, avant de passer à la ligne 6, « **retourner** valeur ». Dans les deux cas, le nombre d'étapes est fini.

Pour l'algorithme 9, nous montons d'un cran en complexité. En effet, en dehors de la boucle, le nombre d'étapes est fini. Il en est de même dans la boucle puisqu'il n'y a qu'une seule opération dedans. Cependant, le corps de cette boucle est exécuté plusieurs fois. Comme k prend toutes les valeurs de 1 à n, le corps de la boucle est donc exécuté n fois, ce qui est fini.

Nous montons encore en complexité pour l'algorithme 10. Comme déjà souligné dans le paragraphe dédié à la boucle « tant que », le nombre de passages dans une telle boucle n'est pas connu à l'avance. Nous devons donc procéder autrement.
Tout d'abord, nous constatons que le nombre d'étapes hors et dans la boucle est fini. Il faut donc vérifier que la boucle « tant que » se termine bien. Dire que la boucle se termine revient à dire qu'il faut qu'à un moment donné, la condition de la boucle soit fausse afin d'en sortir. Dans notre cas, il faut donc que la condition $v \neq 0$ soit fausse, autrement dit, nous devons avoir $v = 0$.
La preuve de la terminaison de cet algorithme se base sur le fait que la valeur de la variable v diminue strictement à chaque nouveau passage dans la boucle tout en restant naturelle. De ce fait, nous construisons une suite

de nombres naturels strictement décroissante. La variable v va donc finir par atteindre la valeur 0. Il s'agit d'un raisonnement similaire à celui présenté dans le chapitre III avec les fractions égyptiennes d'Aldon.

Montrer la terminaison d'un algorithme n'est pas toujours chose aisée. Nous avons pu le constater avec l'exemple précédent. À ce jour, il existe encore des algorithmes pour lesquels la terminaison n'a pas encore été prouvée. C'est notamment le cas de la « conjecture de Syracuse ». Cette procédure est appelée « conjecture » et non « algorithme » car rappelons qu'un algorithme doit nécessairement comporter un nombre fini d'étapes.

Conjecture de Syracuse

Entrée : un nombre naturel n non nul.
Sortie : rien.

1. **tant que** $n \neq 1$ **faire**
2. **si** n est pair **alors**
3. $n \leftarrow \frac{n}{2}$
4. **sinon**
5. $n \leftarrow n \times 3 + 1$
6. **fin si**
7. **fin tant que**

Malgré le fait que ce « pseudo-algorithme » soit assez court (seulement 7 lignes), il n'est pas évident de montrer que la boucle « tant que » va s'arrêter à un moment donné. Pour cela, il faudrait montrer que la variable n atteint la valeur 1 quelle que soit la valeur initiale de n.

VII.4.2 Correction

Prouver la correction d'un algorithme revient à montrer que l'algorithme donne le résultat attendu quelles que soient les entrées. À nouveau, dans certains cas, cela est quasiment trivial et dans d'autres, cela nécessite une démarche plus complexe.

Une fois de plus, pour l'algorithme 8, la preuve est presque immédiate. Il suffit de répertorier les différents cas possibles. Puisque la condition du

« si » est $x \geqslant y$, il y a deux cas à envisager : le cas où $x \geqslant y$ et le cas où $x < y$.
À partir de là, la preuve de la correction est évidente car, dans le premier cas ($x \geqslant y$), la condition du « si » est vraie et le maximum est donc x. Dans le deuxième cas ($x < y$), la condition est fausse et donc le maximum est y.

Pour l'algorithme 9, il n'est plus envisageable de répertorier tous les cas possibles car ils sont trop nombreux. Cependant, le compteur de la boucle « pour tout » est un nombre naturel. Nous pouvons donc prouver qu'une certaine propriété est vraie à chaque nouveau passage dans la boucle. Comme il s'agit de travailler avec des naturels, une technique de preuve fort utile dans ce cas est la **preuve par récurrence**. Le principe de la preuve par récurrence fait partie des compétences terminales à atteindre en fin de secondaire dans le cours de mathématiques 6h/semaine. La preuve de cet algorithme s'insère donc particulièrement bien dans un cours de mathématiques. Elle est vraiment importante car nous avons constaté, au chapitre VI, que bon nombre d'étudiants de première année en mathématiques et en informatique ne la maîtrisent pas.
Nous devons donc montrer, par récurrence, qu'au $k^{\text{ème}}$ passage dans la boucle, nous avons $y = x^k$ (pour $0 \leqslant k \leqslant n$). Une fois cela montré, nous concluons que l'algorithme donne bien le bon résultat. En effet, lorsque l'algorithme se termine, k vaut n donc, après le dernier passage dans la boucle, la variable y possède la valeur x^n.

Pour l'algorithme 10, il n'est pas possible de travailler par récurrence même si nous travaillons dans les naturels. En effet, comme il n'est pas possible de savoir à l'avance les valeurs que vont prendre les différentes variables, la preuve par récurrence ne peut pas être envisageable.
L'astuce, pour la preuve de cet algorithme, consiste à remarquer que l'ensemble des diviseurs communs à a et à b, noté $D(a, b)$, est le même que l'ensemble des diviseurs communs aux variables u et v, noté $D(u, v)$, à chaque passage dans la boucle. De là, nous en concluons, vu que la dernière valeur de v est 0, que l'ensemble des diviseurs communs à a et à b est le même que l'ensemble des diviseurs communs à u et à 0.
Le plus grand diviseur commun à u et à 0 étant u, cela signifie que u est également le plus grand commun diviseur de a et de b. C'est exactement ce que retourne l'algorithme.
Cette preuve permet de travailler la théorie des ensembles. En effet, nous devons montrer l'égalité des deux ensembles $D(a, b)$ et $D(u, v)$. Pour montrer que $D(a, b) = D(u, v)$, nous prouvons les deux inclusions $D(a, b) \subseteq D(u, v)$ et $D(a, b) \supseteq D(u, v)$. Pour montrer l'inclusion de deux ensembles, il faut

montrer que tout élément du premier ensemble appartient au deuxième. C'est également l'occasion de travailler sur les quantificateurs.

En résumé, les preuves d'algorithmes sont très intéressantes d'un point de vue mathématique, même au niveau du secondaire, car elles permettent de travailler la démonstration par récurrence, la théorie des ensembles, la logique,...

VII.5 Complexité

La complexité est une notion propre à l'algorithmique. Elle permet d'estimer l'efficacité d'un algorithme ou de comparer différents algorithmes résolvant le même problème.
En effet, nous avons constaté que parfois, pour un problème donné, plusieurs algorithmes sont envisageables. C'est notamment le cas pour le problème consistant à calculer la somme des n premiers naturels pour lequel nous pouvions utiliser une boucle « pour tout » ou une boucle « tant que ». Dans ce cas-là, la démarche des deux algorithmes était similaire puisque nous avions créé l'algorithme avec la boucle « tant que » à partir de l'algorithme avec la boucle « pour tout ». Cependant, il peut arriver que deux algorithmes permettant de résoudre le même problème n'utilisent pas la même démarche.

Deux types de complexités existent : la complexité en mémoire (pour le stockage des données) et la complexité en temps de calcul (temps d'exécution de l'algorithme). Dans notre dessein d'insérer des éléments d'algorithmique dans le cadre d'un cours de mathématiques, nous nous intéressons uniquement à la complexité en temps de calcul car c'est celle qui est la plus intéressante d'un point de vue mathématique. La complexité en mémoire relevant, quant à elle, davantage du domaine de l'informatique.
Ensuite, nous pouvons étudier la complexité en moyenne ou dans le pire des cas. Nous choisissons de travailler la complexité dans le pire des cas car c'est celle qui est généralement la plus facile à calculer.

Nous pouvons obtenir une idée du temps d'exécution d'un algorithme en implémentant un programme informatique construit sur la base de l'algorithme et en mesurant le temps d'exécution de ce programme (divers langages de programmation proposent des fonctions permettant de mesurer ce temps). Nous avons déjà rencontré ce procédé dans le cours de Programmation et Algorithmique I.

Cependant, il ne s'agit que d'une estimation car le temps d'exécution mesuré par l'ordinateur dépend de la vitesse de l'ordinateur utilisé (le nombre d'instructions qu'il sait exécuter par seconde), du compilateur utilisé, du langage de programmation dans lequel l'algorithme a été traduit,...
Pour avoir une idée du temps d'exécution d'un algorithme indépendante des facteurs tels que l'ordinateur ou le langage de programmation, nous étudions la complexité d'un algorithme de manière théorique.

Pour cela, nous supposons que toute opération élémentaire (affectation, opération arithmétique de base,...) s'exécute en une unité de temps. Pour obtenir la complexité d'un algorithme, il faut ensuite déterminer le nombre d'opérations élémentaires exécutées par l'algorithme. Pour les boucles « pour tout » et « tant que », il y a dans l'une un compteur à incrémenter et dans l'autre une condition à tester. Pour le compteur, il existe différentes conventions : soit nous n'en tenons pas compte dans la complexité, soit nous considérons qu'il s'agit d'une unique opération, soit, si la boucle est empruntée n fois, nous considérons qu'il y a n opérations pour l'incrémentation du compteur. Dans le cadre de cette séquence de cours, nous avons choisi de ne pas rajouter d'opérations pour le compteur et la condition de la boucle « tant que » car nous trouvons que c'est plus simple ainsi.

Afin d'illustrer nos propos, nous présentons à la page suivante deux algorithmes ayant le même but : l'évaluation d'un polynôme en une valeur donnée. Ces algorithmes proviennent de Nguyen (2005).
Ces deux algorithmes ont beaucoup de points communs : utilisation d'une boucle « pour tout », deux opérations (multiplication et addition) dans la boucle, autant de lignes dans l'algorithme,... Pourtant, lorsque nous étudions leurs complexités, nous nous apercevons qu'elles sont assez différentes car le premier algorithme (le 11) fait appel à un autre algorithme.

La complexité de l'algorithme 11 est $T(n) = n^2 + 6n + 6$, tandis que celle de l'algorithme 12 est $T(n) = 4n + 1$.

Dire que la complexité de l'algorithme 11 est $T(n) = n^2 + 6n + 6$ et que celle de l'algorithme 12 est $T(n) = 4n+1$, n'est en soi pas très intéressant. En effet, rappelons que nous avons volontairement négligé plusieurs opérations. Ceci dit, ce n'est pas un problème puisque ce qui est intéressant, dans une étude de la complexité, c'est d'avoir un temps d'exécution théorique lorsque la taille de données (n dans nos deux algorithmes) devient de plus en plus grande.

Algorithme 11 (Valeur d'un polynôme (version 1))

Entrée : n, un naturel ; a_0, a_1,...,a_n, nombres réels ; t, un réel.
Sortie : la valeur du polynôme $a_0 + a_1.x + ... + a_n.x^n$ évalué en $x = t$.

1. $c \leftarrow 0$
2. **pour tout** i allant de 0 à n **faire**
3. $\quad b \leftarrow a_i \times t^i$
4. $\quad c \leftarrow c + b$
5. **fin pour tout**
6. **retourner** c

Remarque : pour calculer t^i, on utilise l'**Algorithme 9**.

Cet algorithme effectue le calcul suivant :

$$c = a_0 + a_1.t + ... + a_{n-1}.t^{n-1} + a_n.t^n.$$

Algorithme 12 (Valeur d'un polynôme (version 2))

Entrée : n, un naturel ; a_0, a_1,...,a_n, nombres réels ; t, un réel.
Sortie : la valeur du polynôme $a_0 + a_1.x + ... + a_n.x^n$ évalué en $x = t$.

1. $c \leftarrow a_n$
2. **pour tout** i allant de 0 à $n - 1$ **faire**
3. $\quad b \leftarrow c \times t$
4. $\quad c \leftarrow b + a_{n-i-1}$
5. **fin pour tout**
6. **retourner** c

Cet algorithme effectue le calcul suivant :

$$c = \Big(\big((a_n.t + a_{n-1}).t + a_{n-2}\big).t + ... + a_1\Big).t + a_0.$$

Il s'agit de l'**algorithme d'Hörner**.

De là, nous comprenons que ce qui est réellement intéressant dans une étude de la complexité, ce ne sont pas les différentes constantes en jeu, mais bien la croissance de $T(n)$. Comme il s'agit d'une étude dans le cas où n devient de plus en plus grand, ce sont les termes de plus haut degré qui nous intéressent et nous donnent le comportement du temps d'exécution.

Pour l'algorithme 11, nous disons donc que la complexité de cet algorithme est de l'**ordre** de n^2 car le terme de plus haut degré est n^2. Similairement, la complexité de l'algorithme 12 est de l'ordre de n car le terme de plus haut degré dans le calcul de la complexité est $4n$.

Afin d'appuyer nos résultats, nous implémentons les deux algorithmes dans un langage de programmation et nous appelons une fonction permettant de mesurer le temps d'exécution pour des valeurs de n allant de 1 à 2000. Ensuite, nous représentons ces points dans un repère.
La FIGURE VII.2 représente les temps d'exécution des deux algorithmes.

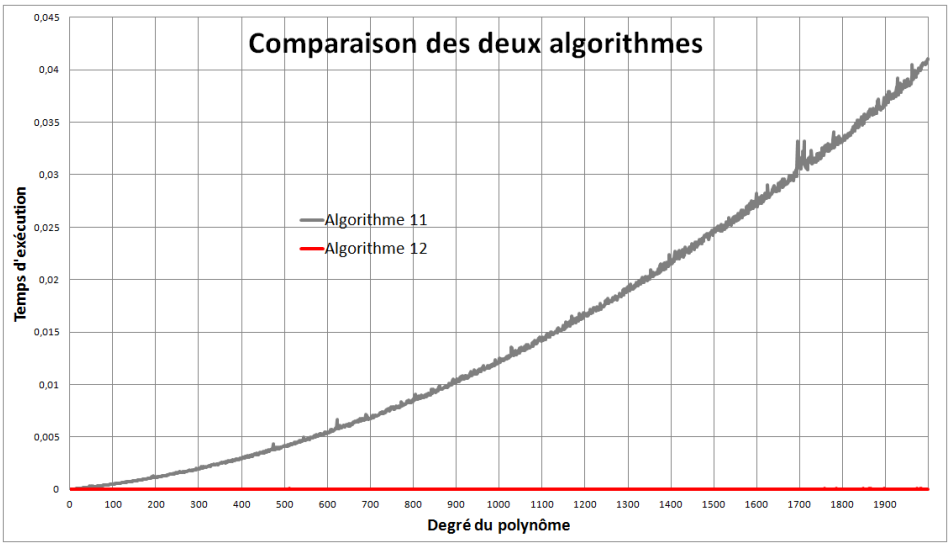


FIGURE VII.2 – Complexité des deux algorithmes en fonction de n

Nous constatons donc que la complexité de l'algorithme 11 est décrite graphiquement par une fonction du deuxième degré, comme nous pouvions nous y attendre puisque sa complexité est de l'ordre de n^2. La complexité de l'algorithme 12 est décrite graphiquement par une fonction du premier degré, ce qui est normal puisque sa complexité est de l'ordre de n.
Cependant, le temps d'exécution de l'algorithme 12 étant assez petit par

rapport à celui de l'algorithme 11, nous ne voyons pas bien la courbe le représentant. La FIGURE VII.3 nous montre uniquement la complexité de l'algorithme 12 afin de voir qu'il s'agit bien d'une droite.

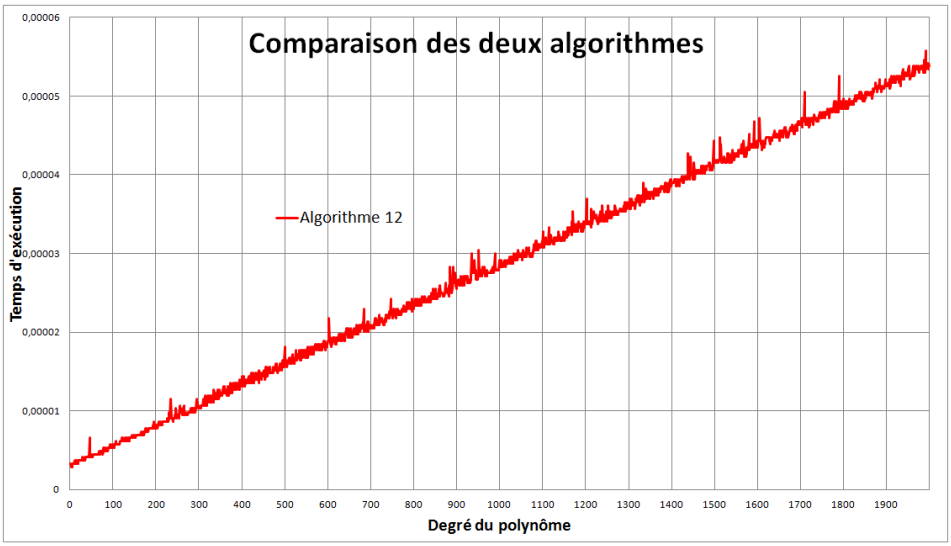


FIGURE VII.3 – Complexité de l'algorithme 12 en fonction de n

Comme la courbe de la complexité de l'algorithme 12 se trouve en dessous de celle de l'algorithme 11, l'algorithme 12 est plus rapide que l'algorithme 11 pour une taille de données de plus en plus grande : il a une meilleure complexité.

Les exemples que nous avons choisis pour étudier la complexité sont assez simples au final. En effet, dans les deux cas, compter le nombre d'étapes de chaque algorithme est suffisant pour trouver la complexité. Déterminer la complexité d'un algorithme n'est pas toujours aussi évident et parfois, nous devons avoir recours aux logarithmes, comme nous avons pu le constater dans l'exemple des fausses pièces de Modeste présenté au chapitre III.

VII.6 Le problème du site de rencontres

La dernière partie de notre séquence d'enseignement est consacrée à un problème plus concret utilisant l'algorithmique[2]. Nous avons intitulé ce

2. Papadimitriou & Vazirani (2006). *Stable Marriage - Application of Proof Techniques for Algorithmic Analysis.*

problème « le problème du site de rencontres ».
Ce problème consiste à former n couples à partir de n hommes et de n femmes tout en tenant compte de leurs préférences.

Nous proposons deux exemples aux élèves : le premier avec 6 personnes (3 couples à former) et le second avec 8 personnes (4 couples à former). Le but de proposer deux exemples aux élèves est de leur montrer que nous ne travaillons pas seulement pour le cas particulier de 3 couples à former.

Dans le premier exemple, les tableaux de préférences des 6 individus sont les suivants :

Alain	D	E	F
Benjamin	E	D	F
Cyrille	D	E	F

Delphine	B	A	C
Éline	A	B	C
Florine	A	B	C

Ces tableaux se lisent horizontalement. Sur la ligne de chaque personne se trouve l'ensemble des personnes du sexe opposé classées par ordre de préférence, de la préférée à la moins désirée.

Pour former ces couples, plusieurs critères sont envisageables. Nous pourrions maximiser le nombre de personnes mises avec leurs premiers choix ou, a contrario, minimiser le nombre de personnes mises avec leurs derniers choix. Cependant, le critère que nous retenons est celui de la stabilité.
Un ensemble de couples est dit *instable* s'il existe un homme et une femme qui préfèreraient être ensemble plutôt que d'être chacun avec la personne qui lui a été attribuée.
Par exemple, l'ensemble de couples {(Alain, Florine), (Benjamin, Éline), (Cyrille, Delphine)} est instable car Alain préfère Éline à Florine et Éline préfère Alain à Benjamin (Alain et Éline préféreraient être ensemble plutôt que d'être avec les personnes qui leur ont été attribuées).

Dans notre deuxième exemple avec les 8 personnes, les tableaux de préférences sont les suivants :

Kevin	G	I	H	J
Luc	G	H	I	J
Marc	H	G	I	J
Nicolas	I	H	G	J

Gaëlle	N	K	L	M
Hélène	K	L	M	N
Isabelle	L	M	N	K
Jeanine	N	L	M	K

Un théorème de l'article consulté nous dit qu'il est toujours possible de trouver au moins un ensemble stable de couples.
L'algorithme suivant permet de trouver un tel ensemble.

Algorithme 13 (Le problème du site de rencontres)

Entrée : n, nombre naturel non nul et $2n$ listes de préférences.
Sortie : un ensemble de n couples stable.

1. **tant qu'**il reste au moins un homme seul **faire**
2. Chaque homme se propose à la 1ère femme non barrée de sa liste.
3. Chaque femme dit « peut-être » à l'homme qu'elle préfère parmi ceux qui se sont proposés à elle.
4. Chaque homme rejeté barre de sa liste la femme à laquelle il s'est proposé.
5. **fin tant que**
6. **retourner** couples

Remarquons que cet algorithme est plus complexe que ceux rencontrés précédemment car les opérations 2, 3 et 4 ne sont pas exécutées juste une seule fois lors d'un passage dans la boucle. En réalité, si nous regardons l'opération 2, il faudrait ajouter une boucle « pour tout » afin de traduire le fait que les n hommes se proposent aux premières femmes de leurs listes.

Dans notre premier exemple, l'algorithme nous donne, après 3 passages dans la boucle, l'ensemble suivant : $\Big\{$(Alain, Delphine), (Benjamin, Éline), (Cyrille, Florine)$\Big\}$.

Il s'agit bien d'un ensemble stable car Alain et Benjamin sont avec leurs premiers choix donc ils n'ont pas envie de bouger. Le seul garçon qui aurait envie de bouger est Cyrille. Cependant, si Delphine doit choisir entre Alain et Cyrille, elle préfère rester avec Alain. De même, si Éline doit choisir entre Benjamin et Cyrille, elle va rester avec Benjamin. Notre ensemble est donc stable.

Dans notre deuxième exemple, l'ensemble stable de couples retourné est $\Big\{$(Kevin, Hélène), (Luc, Isabelle), (Marc, Jeanine), (Nicolas, Gaëlle)$\Big\}$. Nous avons construit cet exemple de sorte que la boucle soit empruntée un nombre assez important de fois (10 fois), dans le but que les élèves voient bien que le nombre de passages dans la boucle n'est pas directement proportionnel à n, le nombre de couples à former.

Nous pouvons donc maintenant vérifier que l'algorithme se termine bien. À nouveau, nous vérifions que le nombre d'étapes hors et dans la boucle est fini et ensuite, nous contrôlons que la boucle se termine. Hors de la boucle, il n'y a que l'opération « **retourner** couples », donc aucun problème. Le nombre d'étapes dans la boucle est aussi fini puisque le nombre d'hommes et de femmes est fini.
La boucle est exécutée au plus n^2 fois. En effet, à chaque passage dans la boucle - sauf au dernier -, au moins un homme barre une femme de sa liste. Comme il y a n hommes et n femmes et que chaque homme peut se proposer aux n femmes, la boucle est exécutée au plus $n \times n$ fois.
En réalité, le nombre de passages serait plutôt $n \times (n-1)$ car les hommes ne peuvent pas barrer les dernières femmes de leurs listes. Si un homme devait supprimer toutes les femmes de sa liste, cela impliquerait que l'algorithme ne se termine pas. Montrons donc qu'aucun homme ne doit barrer toutes les femmes de sa liste et que la boucle se termine donc à un moment donné.

Supposons donc, par l'absurde, qu'un homme H ait barré toutes les femmes de sa liste. Comme il y a le même nombre d'hommes que de femmes, si H est seul, il doit également rester une femme F seule. Comme H a barré toutes les femmes de sa liste, il a également barré F après s'être proposé à elle. S'il a barré F, cela signifie qu'elle lui a préféré un autre homme. Ce qui crée une contradiction puisqu'alors, F ne devrait pas être seule. La terminaison est donc prouvée.

Par cette dernière réflexion, il est donc possible de travailler une autre preuve régulièrement utilisée en mathématiques : la preuve par l'absurde.

VII.7 Bilan

Au cours de ce chapitre, nous avons donc conçu une séquence d'enseignement dans le but d'introduire des éléments d'algorithmique dans le cadre d'un cours de mathématiques.

Nous avons constaté dans les chapitres précédents qu'il était possible de travailler des compétences mathématiques à l'aide de l'algorithmique. Dans notre séquence de cours, nous proposons donc des pistes pour travailler des techniques de preuves (preuve par récurrence, preuve par l'absurde), de la logique (quantificateurs, expressions logiques), de la théorie des ensembles (égalité de deux ensembles),..., sans pour autant faire de mathématiques pures. Constatons que c'est malheureusement le manque de maîtrise de ces compétences qui est souvent reproché aux étudiants qui entrent à l'université.

Remarquons également que nous avons décidé de nous atteler davantage à la compréhension et à l'analyse d'algorithmes qu'à leur écriture. Nous savons que l'écriture d'un algorithme est un exercice difficile et que cela a posé beaucoup de problèmes lors de l'examen de Programmation et Algorithmique I en janvier 2014. Cependant, l'écriture d'algorithmes requérant beaucoup de temps et d'entraînement, nous avons décidé de ne pas y consacrer toute notre séquence de cours : nous préférons passer du temps à travailler des aspects plutôt mathématiques.

Notre séquence d'enseignement sur l'algorithmique rédigée, nous pouvons donc aller l'expérimenter dans le secondaire supérieur belge.
C'est le sujet de notre prochain chapitre.

Chapitre VIII

Expérimentation de la séquence de cours

Dans ce chapitre, nous précisons la manière avec laquelle nous comptons procéder en classe pour enseigner la séquence d'enseignement conçue au chapitre précédent. Nous détaillons les réactions attendues des élèves pour ensuite les confronter à leurs réactions lors de nos expérimentations.

VIII.1 Analyse a priori

Nous avons décidé d'enseigner notre séquence d'introduction à l'algorithmique durant six heures. Normalement, vu le document que nous avons conçu (annexe C), il nous en faudrait davantage, mais nous ne pouvons pas demander à des enseignants de nous donner trop d'heures pour une matière qui ne figure pas dans les programmes de mathématiques de l'enseignement secondaire. Nous allons également travailler avec des élèves ayant au moins six heures de mathématiques par semaine pour que notre séquence de cours puisse éventuellement faire partie des heures de préparation aux études supérieures et ainsi, ne pas faire perdre trop d'heures de cours aux enseignants.

Nous comptons répartir la matière sur ces six heures de la sorte :
- première heure : introduction,
- deuxième heure : étude des différentes notions,
- troisième heure : exercices d'exécutions d'algorithmes,
- quatrième heure : preuves d'algorithmes,
- cinquième heure : le problème du site de rencontres,
- sixième heure : introduction à la programmation et évaluation.

Bien évidemment, nous devons effectuer des choix quant aux parties à présenter. Nous choisissons de ne pas étudier en classe la structure « si...alors...sinon » car les boucles sont plus intéressantes à étudier pour tout ce qui est preuve et complexité. Effectivement, la preuve de la correction d'un algorithme comportant la structure « si...alors...sinon » est quasiment immédiate après avoir répertorié l'ensemble des cas possibles. La preuve de la terminaison d'un tel algorithme est évidente puisqu'aucune opération n'est répétée. À l'inverse, la preuve d'un algorithme avec une structure itérative peut permettre, par exemple, d'employer la preuve par récurrence. Le calcul de la complexité d'un tel algorithme est aussi plus compliqué puisqu'il faut déterminer le nombre de passages dans la boucle et parfois ce nombre n'est pas évident à trouver.
De même, nous décidons de ne pas présenter la complexité aux élèves. En effet, nous pensons que les preuves sont beaucoup plus intéressantes puisqu'elles sont généralement présentes dans des cours de mathématiques. Nous préférons aussi privilégier le problème du site de rencontres par rapport à la complexité car c'est un problème plus ludique et plus concret.

Nous voulons que le document que nous avons rédigé serve de document de référence, un peu comme un syllabus. Pour cela, nous ne le donnerons pas aux élèves directement, de sorte qu'ils puissent prendre des notes et participer activement en classe. Après chaque heure de cours, ils auront les pages du document relatives à l'heure écoulée afin d'obtenir plus d'explications sur des points qu'ils n'auraient éventuellement pas bien compris.
Nous expliquons maintenant avec plus de détails la manière dont nous comptons aborder les différents points de matière en classe et les réactions que nous attendons des élèves. Notre but est donc de faire participer un maximum les élèves.

VIII.1.1 Première heure

Durant cette heure, nous demanderons aux élèves les différentes étapes pour mettre chauffer un plat au micro-ondes. Cela ne risque pas de poser de problèmes particuliers, mais les élèves risquent de rajouter ou d'oublier des étapes. Ils pourraient rajouter les étapes « ouvrir la porte » au début et « fermer la porte » à la fin.
Pour le démarrage de la voiture, nous ne comptons pas demander les étapes aux élèves car elles ne sont pas nécessairement connues de tous. En effet, tous les élèves n'apprennent pas forcément à conduire et ils ont peut-être une voiture avec une carte et non une clé. Afin de ne pas

perdre trop de temps à écrire les algorithmes au tableau, nous donnons aux élèves, au fur et à mesure, des feuilles ne comportant que les algorithmes et les tableaux de préférences pour le problème du site de rencontres.

Une fois ces deux exemples donnés, nous demanderons aux élèves de les comparer afin de relever les caractéristiques d'un algorithme. Pour qu'ils ne partent pas dans tous les sens, nous allons essayer de les guider en leur posant des questions du style : « À quoi sert un algorithme en général ? », « Quelles sont les caractéristiques de sa structure ? »,... Nous pensons qu'ils vont voir assez rapidement l'aspect « problème ».
En ce qui concerne le fait qu'un algorithme est valable pour une classe de problèmes, nous ne nous attendons pas à ce qu'ils le trouvent directement et nous allons donc leur demander si l'algorithme du micro-ondes fonctionne pour faire chauffer une soupe, des pâtes,... De là, le fait qu'un algorithme est général devrait émerger.
Pour le nombre fini d'étapes, nous ne pensons pas non plus qu'ils vont le signaler, mais ils vont peut-être parler du fait que les étapes sont numérotées. De là, nous pourrons leur demander ce qu'ils peuvent dire sur le nombre d'étapes. Nous nous attendons à ce qu'ils disent qu'il y a 6 étapes pour le démarrage de la voiture. Nous leur ferons alors remarquer que si la voiture démarre tout de suite, toutes les étapes ne sont pas exécutées. Nous essayerons de leur faire dire que le nombre d'étapes est fini en leur demandant s'il est possible d'essayer continuellement de faire démarrer la voiture.

Enfin, nous passerons à l'algorithme de la division euclidienne. Nous commencerons par demander aux élèves comment ils effectuent une telle division dans les naturels, exemple à l'appui. De là, nous les inviterons à expliquer la façon dont ils trouvent le quotient et le reste afin qu'ils parlent du fait qu'il faut soustraire le diviseur du dividende autant de fois que nécessaire et que nous puissions introduire l'algorithme de la division euclidienne. En l'introduisant, nous allons créer des liens avec ce qui aura été vu avant en leur demandant de trouver le problème que résout cet algorithme ainsi que la famille d'instances. Ils devraient facilement voir le problème, mais ils risquent d'avoir plus de mal avec la famille d'instances car ici, il y a deux variables à considérer. Pour l'exécution de l'algorithme, nous pensons que les élèves vont avoir du mal avec les affectations qu'ils risquent de confondre avec des égalités. Nous veillerons donc à établir un parallèle entre une affectation et une case mémoire d'une calculatrice. De même, pour les affectations au sein de la boucle, les élèves risquent à chaque fois de reprendre les valeurs initiales et non les dernières valeurs. Nous expliquerons

que lors d'une affectation, les anciennes valeurs sont remplacées par les nouvelles.

À la fin de cette heure, nous attendons donc des élèves qu'ils soient capables de donner les principales caractéristiques d'un algorithme et de déterminer si une procédure en est un.

VIII.1.2 Deuxième heure

Durant cette deuxième heure, nous allons nous atteler à la deuxième partie de notre séquence de cours : les différentes notions pour comprendre un algorithme. Nous rediscuterons brièvement des notions d'entrée, sortie et affectation qui auront déjà été vues durant la première heure. Ensuite, nous introduirons la boucle « pour tout » en leur proposant le problème du calcul de la somme des n premiers naturels. Afin de les guider, nous leur demanderons comment ils procèdent pour calculer la somme des 5 premiers naturels. Normalement, ils devraient signaler qu'à chaque fois, il faut ajouter une nouvelle valeur et celle-ci va de 1 à 5. De là, l'idée de la boucle « pour tout » devrait assez facilement émerger. Dans cet algorithme, nous avons besoin d'une initialisation de la variable « somme » puisque lors du premier passage dans la boucle, cette variable prend la valeur « somme $+ i$ ». Il faut donc qu'il y ait une valeur dans cette variable pour pouvoir l'utiliser. Nous pensons qu'il sera assez facile de faire comprendre cela aux élèves et nous les guiderons de sorte que ce soit eux qui signalent qu'il faut initialiser cette variable. En ce qui concerne l'exécution de cet algorithme, ils auront peut-être du mal avec la variable i qui augmente automatiquement à chaque nouveau passage dans la boucle sans que ce soit signalé explicitement.

Enfin, nous écrirons un algorithme avec une boucle « tant que » pour résoudre le même problème. L'écriture de cet algorithme ne devrait pas prendre trop de temps, étant donné certaines opérations sont communes à l'algorithme avec la boucle « pour tout ». Cependant, les élèves devront trouver la condition à mettre après le « tant que » et devront s'arranger pour faire varier le i. Nous ne pensons pas que cela posera problème, mais ils ne verront peut-être pas pourquoi il faut noter l'opération $i \leftarrow i + 1$.

La dernière structure présente dans cette deuxième partie est l'instruction conditionnelle « si...alors...sinon ». Comme déjà signalé précédemment, nous ne comptons pas étudier cette structure en classe par manque de temps, mais nous demanderons aux élèves de lire dans les feuilles les parties la

concernant afin qu'ils soient capables d'exécuter un algorithme la contenant. Nous leur demanderons également de relire attentivement leurs notes pour le prochain cours.

Après cette heure de cours, nous attendons des élèves qu'ils soient capables d'écrire un algorithme semblable à ceux construits en cours. Ils doivent également avoir compris les différentes notions composant un algorithme et savoir exécuter un algorithme les contenant.

VIII.1.3 Troisième heure

Pour entamer cette heure de cours, nous allons signaler aux élèves qu'avant, nous leur donnions un problème, nous cherchions un algorithme pour le résoudre et nous exécutions l'algorithme créé pour des valeurs données. Pendant cette heure, nous leur donnerons des algorithmes et ils devront déterminer ce qu'ils calculent. Pour déterminer la sortie d'un algorithme, l'idée de le tester avec certaines valeurs devrait émerger. Nous leur demanderons s'il est possible de tester un algorithme avec n'importe quelles valeurs afin de bien insister sur le fait qu'elles doivent appartenir aux entrées.
Nous leur proposerons les trois exercices se trouvant dans notre document : un avec la structure « si...alors...sinon », un avec une boucle « pour tout » et un avec une boucle « tant que » (algorithmes 8, 9 et 10). Ce sera l'occasion de voir s'ils ont réellement travaillé la structure conditionnelle. Nous ne pensons pas que ces exercices poseront un problème particulier car nous aurons déjà exécuté des algorithmes durant les heures précédentes.

Les élèves choisiront les valeurs eux-mêmes, mais nous veillerons à ce qu'ils ne donnent pas des valeurs trop grandes pour ne pas passer trop de temps sur une exécution. Nous leur demanderons de compter le nombre de passages dans les boucles pour les algorithmes 9 et 10. Cela nous sera utile durant la quatrième heure.
En ce qui concerne la sortie de ces différents algorithmes, nous pensons que celles des algorithmes 8 et 9 devraient être visibles avec un seul exemple, mais que pour l'algorithme 10, plusieurs exécutions avec des valeurs différentes seront nécessaires. Afin qu'ils trouvent la sortie de cet algorithme, nous leur poserons quelques questions pour les mettre sur la piste (« La sortie est-elle plus grande/petite que les entrées ? », « Que constatez-vous par rapport aux deux entrées et à leurs diviseurs ? »).

À la fin de cette heure et afin d'introduire le sujet de l'heure suivante, nous demanderons aux élèves si nous avons la certitude que la sortie va toujours être celle trouvée, quelles que soient les valeurs appartenant aux entrées choisies. Nous pensons qu'une majorité d'élèves répondra « non » et nous leur rappellerons qu'en mathématiques, des exemples ne suffisent pas pour prouver une propriété valable pour un très grand nombre de cas : il faut effectuer une démonstration.

Après cette troisième heure, les élèves devraient être capables de comprendre les différentes notions composant un algorithme, d'exécuter un algorithme et de généraliser sur la base d'exemples.

VIII.1.4 Quatrième heure

Cette heure sera exclusivement consacrée aux preuves d'algorithmes. Nous pensons que ce sera l'heure la plus difficile pour les élèves car elle comportera des aspects beaucoup plus théoriques que les précédentes.
En ce qui concerne la correction, nous l'aurons déjà introduite à la fin de la troisième heure. Pour introduire la terminaison, nous allons demander aux élèves de relire la définition d'« algorithme » qui aura été donnée et leur demander si rien ne les trouble par rapport à tout ce que nous aurons fait jusqu'alors. Ils devraient signaler que nous n'avons jamais vérifié que le nombre d'étapes était fini.

Par manque de temps, nous allons seulement démontrer la terminaison et la correction de l'algorithme 9. Nous avons choisi cet algorithme car, pour en prouver la correction, nous utilisons une preuve par récurrence, preuve faisant partie des compétences terminales à maîtriser en fin de secondaire en mathématiques pour les scientifiques.
Pour la terminaison, les élèves ne vont peut-être pas directement comprendre pourquoi nous distinguons ce qui se trouve hors de la boucle de ce qui se trouve dedans. Nous leur montrerons, à l'aide des exécutions de l'algorithme effectuées durant l'heure précédente, que ce qui se trouve dans la boucle peut être répété. Les élèves devraient trouver le nombre de fois que le corps de la boucle est exécuté car nous leur aurons demandé de le compter lors des exécutions de l'algorithme. Ils ne devraient pas avoir trop de mal à le généraliser. Ils risquent également de se demander pourquoi nous ne comptons pas le « pour tout » et le « fin pour tout ». Nous leur expliquerons que nous avons choisi de ne pas en tenir compte et nous pourrons leur montrer que même si nous avions décidé de les compter, cela n'aurait rien

changé au fait que l'algorithme se termine bien.

Nous pensons que la correction va poser grandement problème aux élèves. Si les élèves n'ont jamais rencontré de preuves par récurrence, ils risquent d'avoir du mal à en comprendre le principe général. Si les élèves ont déjà démontré des propriétés à l'aide de cette preuve, nous pensons qu'ils auront quand même du mal à adapter cette preuve à la preuve d'un algorithme. En effet, pour un algorithme comme le 9, nous ne montrons pas directement que l'algorithme est correct : nous montrons qu'une certaine propriété est vraie après chaque passage dans la boucle. Ce simple fait pourrait déstabiliser des élèves ayant l'habitude que la propriété à démontrer leur soit directement donnée. Nous nous attendons à ce que des élèves veuillent démontrer l'hypothèse de récurrence puisque des étudiants universitaires ont commis cette erreur.

Après cette heure de cours, nous n'attendons pas des élèves qu'ils soient capables de refaire entièrement une preuve par récurrence, mais nous voulons qu'ils en aient compris le principe (un cas de base et un pas de récurrence qui se sert d'une hypothèse de récurrence).
Nous estimons cependant qu'ils doivent être capables de prouver la terminaison d'un algorithme avec une boucle « pour tout ».

VIII.1.5 Cinquième heure

Durant cette heure, nous allons donc nous intéresser au problème du site de rencontres. Nous donnerons aux étudiants les tableaux de préférences pour le premier exemple ainsi que la façon de les interpréter. À partir des tableaux, nous leur expliquerons que notre travail consistera à former des couples en respectant les préférences des différentes personnes. Nous leur demanderons des exemples de critères pour la formation des trois couples et nous nous attendons à ce qu'ils pensent à maximiser le nombre de premiers choix ou à minimiser le nombre de derniers choix. Bien évidemment, ils n'imagineront pas le critère de stabilité, critère que nous avons retenu.
En ce qui concerne l'algorithme pour obtenir un ensemble stable de couples, nous comptons effectuer au moins une fois le corps de la boucle avec les élèves avant de les laisser travailler individuellement car nous trouvons que cet algorithme est nettement plus compliqué que ceux rencontrés précédemment.
Pour la terminaison de cet algorithme, nous pensons qu'ils vont se souvenir de ce qui aura été fait précédemment et qu'ils penseront à regarder ce qu'il y

a hors de la boucle et dedans. Pour le nombre d'exécutions de la boucle, les élèves ne verront pas directement qu'elle est exécutée au plus n^2 fois. Nous devrons donc les guider en leur montrant les cases barrées des tableaux de préférences au fur et à mesure des passages dans la boucle.
Enfin, nous n'aurons sûrement pas le temps d'étudier le deuxième exemple en classe.

À la fin de cette heure de cours, les élèves doivent être capables de comprendre des algorithmes décrits en français, plus compliqués que ceux vus dans les parties précédentes, afin de les exécuter.

VIII.1.6 Sixième heure

Durant cette dernière heure, nous proposerons une introduction à la programmation d'environ 15/20 minutes. En effet, nous pensons que les élèves risquent de rester sur leur fin si nous ne leur montrons pas un peu de programmation. Pour ce faire, nous avons traduit les algorithmes construits pas à pas en classe (algorithmes 5, 6 et 7) dans les langages de programmation « Pascal », « C », « Python » et « Matlab ». Nous avons choisi ces langages de programmation car le Pascal est assez facile à comprendre ; le C est souvent utilisé dans des études supérieures et il est proche d'autres langages comme le « C++ » et le « Java » ; le Python, car c'est le langage du cours de Programmation et Algorithmique I et le Matlab car il est couramment utilisé dans des études de mathématiques et d'ingénierie civile. Nous indiquons également aux élèves des programmes à installer pour programmer dans ces différents langages, exception faite du Matlab qui est un logiciel payant.
Les différents programmes informatiques que nous comptons présenter aux élèves se trouvent dans l'annexe D. En classe, nous leur montrerons comment les différentes structures sont traduites dans ces langages et nous exécuterons les codes pour leur montrer ce qui se produit.

Enfin, nous conclurons ces six heures de cours par une évaluation de 30/35 minutes afin de voir les difficultés des élèves face à l'algorithmique. La présentation de cette évaluation et les réponses des élèves se trouvent au chapitre suivant.

VIII.2 Expérimentations

Durant le mois de mars 2014, nous avons testé notre séquence de cours dans trois classes différentes. Nous présentons brièvement ces classes, notées « Classe 1 », « Classe 2 » et « Classe 3 ». L'ordre choisi est l'ordre chronologique de nos expérimentations.

- ⋆ La classe 1 est une classe de sixième secondaire composée de 18 élèves. Ils ont 6+2 heures de mathématiques par semaine et nous notons leur enseignante « Enseignante 1 ».
- ⋆ La classe 2 est une classe de cinquième secondaire composée de 6 élèves. Ils ont 6+1 heures de mathématiques par semaine et nous notons leur enseignante « Enseignante 2 ».
- ⋆ La classe 3 est une classe de sixième secondaire composée de 14 élèves. Ils ont 6+1 heures de mathématiques par semaine et leur enseignante est également l'Enseignante 2.

Nous avons enregistré les cinq premières heures de chaque expérimentation afin de pouvoir retranscrire exactement les différentes réactions des élèves. Nous n'avons pas jugé important d'enregistrer les sixièmes heures puisqu'elles étaient consacrées aux évaluations et aux présentations des différents langages de programmation.

Nous avons également repris les notes des élèves afin de voir comment ils retravaillaient leurs notes chez eux, notamment pour la structure conditionnelle que nous n'avons pas étudiée en classe. Cependant, nous avons constaté qu'en général, les élèves n'avaient rien écrit de plus sur leurs feuilles que ce qui avait été dit en classe. En ce qui concerne la structure conditionnelle, certains n'ont même pas pris la peine de l'intégrer dans leurs notes, malgré notre demande. Ceux qui l'ont intégrée ont simplement recopié un exemple ou alors la structure générale.

Nous relatons séparément les trois expérimentations. Dans la suite, nous présentons certains échanges avec les élèves. Dès que nous intervenons, nous le signalons par la lettre « P » et dès que ce sont les élèves qui interviennent, nous le notons avec un « E » ou « E_1 » et « E_2 » si plusieurs élèves réagissent en même temps, ou encore avec « Es » si plusieurs élèves ont la même réaction en même temps. La notation « (...) » signifie que nous passons des passages de la discussion.

VIII.2.1 Expérimentation dans la classe 1

Première heure

Pour le plat au micro-ondes, comme nous pouvions nous y attendre, les élèves ont voulu ajouter des étapes, comme le fait de retirer le film plastique, mais ils ont vite été convaincus du fait que tous les plats à faire chauffer au micro-ondes n'ont pas un film plastique au-dessus. Ils ont également voulu ajouter l'étape « manger le plat », mais nous leur avons rappelé que notre but était simplement de faire chauffer le plat. Ils ont néanmoins ajouté les étapes « ouvrir la porte » et « fermer la porte » que nous n'avions pas mises, mais qui étaient acceptables.
Pour la voiture, ils ont juste signalé qu'il fallait, en plus, appuyer sur le bouton, ce à quoi nous avons répondu qu'il n'y avait pas de bouton dans les voitures fonctionnant avec une clé.

Enfin, pour trouver les éléments qui caractérisent un algorithme, les élèves ont eu des réactions assez rapides et parfois fausses et ils n'ont pas forcément cité les points que nous aurions voulu qu'ils citent. Nous avons donc eu quelques échanges avec eux pour faire émerger ces différentes caractéristiques.

E : Dans un algorithme, toutes les étapes sont effectuées.
P : Non, dans l'exemple de la voiture, si elle démarre, nous ne passons pas par l'étape 6.
E : Il y a une condition dans un algorithme.
P : Pas dans le premier algorithme. Nous cherchons des caractéristiques générales.
Es : ...
P : À quoi sert un algorithme ?
E : À faire chauffer un plat.
P : Et en général ?
Es : ...
P : Ça sert à résoudre...
Es : Un problème !
P : Le premier algorithme, est-il valable pour une soupe ?
E : Oui.
P : Pour des pâtes ?
E : Oui.
P : Que pouvez-vous dire alors sur un algorithme ?
E : Il est général.
(...)

P : Quelle est la famille d'instances pour l'algorithme du micro-ondes ?
E_1 : Tous les trucs à manger.
E_2 : Tous les plats froids.
P : Pouvons-nous tout mettre au micro-ondes ?
E : Non.
P : L'algorithme n'est valable que pour les plats à chauffer au micro-ondes. Donnez une instance de ce problème.
E : La soupe.
P : Ok ; et pour le démarrage de la voiture, quelle est la famille d'instances ?
E : Toutes les voitures à clé.
(...)
P : Que pouvez-vous dire sur les étapes ?
E : C'est en nombre variable.
P : Oui, mais encore ?
Es : ...
P : Comptez les étapes dans les deux algorithmes. Combien y en a-t-il pour le micro-ondes ?
E : 9.
P : Et pour la voiture ?
E : 6.
P : Et si elle démarre directement ?
E : 4.
P : Oui, et dans le pire des cas, il y en a beaucoup plus. Que pouvez-vous dire alors ?
E : Il y a un début et une fin.
P : Oui, vous êtes sur la bonne voie ; par rapport à la fin ?
Es : ...
P : Le nombre d'étapes est ...
E : Fini !

Afin de voir s'ils avaient bien compris qu'un algorithme était général, nous leur avons demandé pourquoi une recette de cuisine ne pouvait pas être considérée comme un algorithme et ils ont correctement répondu à la question en disant qu'elle n'était pas générale.
Nous passons maintenant à l'algorithme de la division euclidienne. Nous ne relatons pas le moment où ils ont rappelé la technique de la potence et les soustractions à effectuer pour obtenir le quotient et le reste car les réactions ont été très rapides et sans réelle discussion.

P : Quel est le problème à résoudre pour la division euclidienne ?
E : La division de 17 par 5.
P : Oui, dans notre exemple, mais en général ?
E : La division de deux naturels.
P : Quelle est la famille d'instances alors ?
E : Les naturels.
P : Il nous faut deux valeurs.
E : Un couple de naturels.
P : Nous pouvons prendre tous les naturels ?
E : Non, pas 0 pour le diviseur.
(...)
L'algorithme de la division euclidienne est maintenant donné aux élèves.
P : Que représente l'entrée ?
E : La famille d'instances.
P : Oui ; et la sortie ?
E : La résolution du problème.
P : Non, la résolution, ce sont les étapes.
E_1 : La solution.
E_2 : La réponse.
P : Ok !

Nous avons également expliqué ce que signifiaient des expressions telles que « $r \leftarrow a$ » et « tant que ... faire », mais les élèves n'ont pas du tout réagi durant nos explications.
Pour l'exécution de cet algorithme avec les valeurs $a = 17$ et $b = 5$, les élèves ont éprouvé quelques difficultés avec les affectations, comme nous le relatons ci-dessous.

P : La première étape est $r \leftarrow a$. Que faisons-nous ?
Es : ...
P : Nous changeons les lettres que nous connaissons. $r \leftarrow$ ... ?
E : 17.
P : Oui ; quelle est la deuxième étape ?
Es : ...
P : Quelle est-elle ? L'étape 2 ?
E : $q \leftarrow 0$.
P : Oui, tout est connu donc il n'y a rien à changer.
(...)
P : Quelle opération faisons-nous à la quatrième ligne ?
E : $17 \leftarrow 17 - b$
P : Non, les lettres de gauche restent ainsi, nous n'y touchons pas.

E : $r \leftarrow 17 - 5 = 12$.

Durant cette heure, nous avons donc constaté que les élèves avaient eu du mal à trouver les caractéristiques d'un algorithme sur la base de deux exemples. Cependant, nous ne nous attendions pas à ce qu'ils trouvent toutes les caractéristiques par eux-mêmes vu que certaines n'étaient vraiment pas évidentes à voir.
Pour l'exécution de l'algorithme de la division euclidienne, les élèves semblent avoir eu du mal avec l'affectation. Nous verrons dans les heures suivantes s'ils en ont maintenant bien compris le principe.

Deuxième heure

Durant la deuxième heure, nous avons demandé aux élèves de nous rappeler à quoi correspondaient les notions d'entrée et de sortie et comment s'écrivait une affectation. Ils s'en souvenaient puisque nous venions de voir ces notions durant la première heure.

Pour introduire la boucle « pour tout », les élèves ont bien su expliquer comment effectuer la somme des 5 premiers naturels en ajoutant à chaque fois une nouvelle valeur. Nous relatons la construction de l'algorithme permettant d'obtenir la somme des n premiers naturels.

P : Nous allons essayer d'écrire l'algorithme ensemble. Nous mettons quoi tout en haut ?
E : Les entrées et les sorties.
P : Quelles sont-elles ?
E : Un naturel pour l'entrée.
P : Oui, et pour la sortie ?
E : La somme des 5 premiers naturels.
P : Attention, ça, c'était un exemple.
E : La somme des n premiers naturels.
P : Écrivons l'algorithme...
Es : ...
P : Nous faisions quoi par rapport à la variable *somme* ?
E : On ajoutait à chaque fois une nouvelle valeur.
P : Comment nous notions le fait que nous ajoutions à chaque fois une nouvelle valeur ?
E : On fait une flèche.
P : Ok, notons $somme \leftarrow somme + ...$ Nous devons ajouter quoi ?
E : $n + 1$.

P : Ajoutons-nous toujours $n + 1$?
Es : Non !
P : Appelons i, ce nombre. Il varie de quelle à quelle valeur ?
E : De 1 à n.
P : Ok, notons « pour tout i allant de 1 à n ». L'algorithme est bon ainsi ou il manque encore des choses ?
E_1 : Il manque le « retourner ».
E_2 : Ça signifie quoi « retourner » ?
P : Ça veut dire que l'algorithme nous renvoie la valeur, c'est la sortie ; l'algorithme doit nous donner un résultat. Il ne manque toujours rien à l'algorithme ?
E : Affecter un nombre à la somme de départ.
P : Oui, pourquoi ? Explique.
E : Sinon, il ne sait pas partir.

Lors de l'exécution de cet algorithme, un étudiant a confondu les variables *somme* et i.

E : On est en dehors de la boucle car $6 \geqslant 5$.
P : Tout le monde est d'accord ?
E : Non ! i vaut 3.
P : Oui, et c'est i qui doit aller jusque 5.

L'écriture de l'algorithme avec une boucle « tant que » a également posé quelques problèmes aux élèves.

P : Comment écrire une boucle « tant que » ?
E : Il faut écrire une question.
P : Non, ça, c'est au moment de tester l'algorithme. La variable i peut aller jusque quelle valeur ?
E : n.
P : Ok, quelle est la condition alors ?
E : Tant que $i \geqslant n$.
P : C'est bon ?
E : Tant que $i \leqslant n$.
P : Dans une telle boucle, il faut incrémenter i, l'augmenter car il n'augmente pas automatiquement.
E : $i \leftarrow n + 1$.
P : Attention : n ne varie pas !
E : $i \leftarrow i + 1$.
P : Oui, c'est le i qui doit augmenter. Il nous manque encore des étapes.

E : *somme* $\leftarrow 0$.
P : C'est tout ?
E : Retourner *somme*.
P : Oui ! Il ne manque pas encore quelque chose ? Par rapport aux variables ?
E : i.
P : Nous le faisons commencer à quelle valeur ?
E : À n.
P : Nous faisons commencer i à n ?
Es : Non, à 1 !

L'exécution de cet algorithme pour une valeur de n donnée n'a posé aucun problème aux élèves puisque nous avions déjà exécuté un algorithme du même genre.

Au final, pour la construction des algorithmes, les élèves ont eu d'assez bonnes idées, mais nous avons dû un peu les guider pour qu'ils trouvent toutes les opérations. Cependant, c'était la première fois qu'ils avaient à concevoir des algorithmes et nous ne nous attendions donc pas à ce qu'ils trouvent toutes les opérations directement.
En ce qui concerne l'exécution d'un algorithme, nous trouvons que les élèves y arrivent assez facilement, parfois en commettant quelques erreurs de distraction.

Troisième heure

Durant le début de la troisième heure, nous avons demandé aux élèves comment il était possible de déterminer la sortie d'un algorithme. Les élèves ont tout de suite eu de bonnes idées et ont été très réactifs, comme en témoignent ces quelques échanges.

P : Comment allons-nous faire pour déterminer la sortie d'un algorithme ?
E : L'appliquer à un exemple concret.
P : Nous allons faire ça. Nous pouvons l'appliquer avec n'importe quelles valeurs ?
E : Deux réels.
P : Pour le premier algorithme, ok. Explique où tu le vois.
E_1 : Dans l'entrée.
E_2 : Ça doit respecter la condition d'entrée.
(...)
P : Et pour l'algorithme 10, nous choisissons quelles valeurs ?
E : 1000 !

P : Il nous faut une seule valeur pour le 10 ?
E : 1000 et 1000.
P : Nous allons essayer de ne pas partir dans des valeurs trop grandes.
E : 1 et π.
P : Pouvons-nous essayer avec π ?
E : Non ! Il faut des naturels !

Les élèves n'ont eu aucun problème pour exécuter l'algorithme 8 comportant la structure « si...alors...sinon » non étudiée en classe.
Dans l'algorithme 9, la variable de la boucle est k, mais k n'apparaît pas dans le corps de la boucle. Cela a troublé certains élèves qui pensaient alors que les valeurs de x et de y restaient constantes.
L'algorithme 10 n'a posé aucun souci aux élèves quant à son exécution, ils ont juste eu du mal à en déterminer la sortie générale. Ils pensaient qu'il donnait la différence des deux naturels, ou le reste de leur division, ou encore que l'algorithme donnait toujours 1. Nous leur avons donné quelques exemples supplémentaires avec d'autres valeurs pour qu'ils réfutent leurs hypothèses et qu'ils déterminent la véritable sortie.

Au final, durant cette heure de cours, les élèves n'ont pas semblé éprouver de difficultés particulières en ce qui concerne l'exécution d'un algorithme. Le seul algorithme qui a posé quelques problèmes est celui avec la boucle « pour tout » car la variable de la boucle n'intervient pas dans le corps de la boucle.

Quatrième heure

Comme prévu, pour introduire la terminaison, nous avons demandé aux élèves de relire la définition d'« algorithme » et de voir si rien ne les troublait par rapport aux algorithmes rencontrés durant la troisième heure. Ils ont eu des réactions rapides, mais n'ont pas trouvé directement le fait que nous n'avions jamais vérifié que le nombre d'étapes était fini.

P : Reprenez la définition d'« algorithme » ; rien ne vous trouble par rapport à ce que nous avons fait ?
E : On dit que c'est bon pour toute instance, mais on n'a vérifié que pour des cas particuliers.
P : Oui, mais les instances sont données dans l'entrée ; nous avons juste fait des exemples.
E : On ne résout pas de problèmes.
P : Si, c'est juste qu'au départ, nous ne savions pas quel problème puisque

nous cherchions la sortie.
E_1 : Nombre fini d'étapes.
E_2 : Le nombre d'étapes est fini !!!
P : Ok pour le 8, mais pour le 9 et le 10, les boucles peuvent être empruntées un nombre infini de fois.
E : Non ! On a le « fin pour tout » et le « fin tant que » !
P : Oui, mais pour le 10, par exemple, rien ne nous dit que la variable v va atteindre la valeur 0.
E : Haaaa !

La preuve de la terminaison de l'algorithme 9 a également posé un léger souci puisque certains élèves n'ont pas pensé au fait que les opérations dans une boucle pouvaient être répétées.

P : Combien d'étapes y a-t-il dans l'algorithme 9 ?
E : 5.
P : Attention, l'étape 3 est répétée. La boucle est exécutée combien de fois ?
E : 5 fois.
P : Oui, dans le premier exemple que nous avons fait. Combien de fois dans le cas général ?
E : n.
P : Combien d'étapes en tout alors ?
E : $2 + n$.
P : C'est fini alors ?
E : C'est figé.
P : Non, ça dépend du nombre de tours dans la boucle.
E : Ça dépend de n, n peut être infini.
P : Attention, n est un naturel, c'est fini. Ce n'est pas une limite.
E : Oui, mais l'infini ?
P : C'est un naturel ?
E : Ha non.

La preuve de la correction n'a pas posé de majeurs problèmes en classe puisque les élèves avaient déjà eu l'occasion de travailler de nombreuses preuves par récurrence avec leur enseignante.
Nous avons demandé aux élèves pourquoi, en montrant qu'une certaine assertion était vraie à chaque passage dans la boucle, la correction de l'algorithme allait être prouvée. Ils n'ont pas vu directement le lien.
De même, pour le cas de base de la récurrence, ils n'ont pas compris immédiatement que quand k valait 0, cela signifiait que nous n'étions pas encore dans la boucle.

P : Vous voyez pourquoi nous allons montrer ça à chaque passage ?
Es : ...
P : Quel est le lien avec ce que nous voulons montrer pour la correction ?
E : n.
P : Oui, k va jusque n. Quel est le lien ?
E : k va de 1 à n.
P : Oui, au dernier passage dans la boucle, k va valoir ...
E : n.
(...)
P : Si k vaut 0, nous faisons quelle(s) opération(s) ?
E : On passe au « fin pour tout ».
P : Attention, k va varier de 0 à n et nous sortons de la boucle seulement quand k vaut n. Si k vaut 0, nous ne sommes pas encore dans la boucle.

En résumé, durant cette heure, les élèves ont eu du mal avec la terminaison car ils n'avaient pas pensé au fait que les opérations dans une boucle sont répétées. Quant à la correction, ils ont voulu prouver le cas de base en considérant que n valait 0. Or, c'est k qui vaut 0 et n est fixé. Nous avons vu que plusieurs étudiants universitaires avaient commis la même erreur pour cette preuve. En ce qui concerne le pas de récurrence, les élèves n'ont eu aucun problème et ils ont réussi à nous expliquer les différentes parties de la preuve.

Cinquième heure

Nous avons demandé aux élèves de nous citer des exemples de critères pour résoudre le problème du site de rencontres, mais ils ne sont pas parvenus à en trouver de véritables.

P : Quel critère pourrions-nous choisir ?
E : Tous à moitié contents.
P : C'est-à-dire ?
E : Delphine avec Alain, comme ça, elle est à moitié contente.
P : Nous allons quand même devoir mettre un garçon avec son dernier choix. Quel autre critère ?
Es :...
P : Essayer de maximiser le nombre de premiers choix, ou à l'inverse...
E : Minimiser le nombre de derniers choix.
P : Oui, cependant, nous allons choisir le critère de *stabilité*.
(...)
E : Mais là, il n'y a pas d'ensemble stable pour cet exemple.

P : Si, le théorème nous dit que quels que soient les tableaux, il va toujours exister un ensemble de couples stable.
E : Oui, mais il y en a toujours un qui sera avec son dernier choix.
P : Oui, mais la stabilité concerne deux personnes.
E : On va mettre les deux derniers choix ensemble !
P : Oui, ici ça va être le cas.

L'exécution de l'algorithme sur le premier exemple n'a posé aucun problème en classe. Une fois l'ensemble stable de couples trouvé au moyen de l'algorithme, un élève a remarqué que, pour cet exemple, il n'existait pas qu'un seul ensemble stable de couples.

E : Mais ça aurait aussi pu marcher en prenant le premier choix de la femme et le deuxième choix de l'homme.
P : Oui, le théorème nous disait qu'il existait au moins un ensemble stable. En fait, l'algorithme favorise les hommes.

Pour trouver le nombre de fois que la boucle est empruntée au maximum, les élèves n'ont pas eu énormément de mal et ont vite trouvé le n^2.

P : Combien de fois passons-nous dans la boucle ?
E : $n-1$ fois.
P : Regardez par rapport à l'exemple, c'est possible ?
E : n fois car l'homme a n possibilités.
P : Oui, mais n'y a-t-il qu'un seul homme ?
E : n.
P : Donc combien de passages en tout ?
E : n^2.

Cette dernière heure n'a donc pas posé de problèmes aux élèves puisque, même quand ils n'avaient pas directement la bonne réponse, en leur posant quelques questions, ils parvenaient à trouver la réponse correcte.

Conclusion de cette expérimentation

La classe 1 a donc été une classe très réactive et les élèves ont presque toujours trouvé les réponses attendues, soit directement, soit en leur posant quelques questions. Ils ne semblent pas avoir éprouvé d'énormes difficultés et les quelques difficultés rencontrées, comme l'affectation durant la première heure, se sont vite résorbées. Presque tous sont parvenus à exécuter les algorithmes proposés durant la troisième heure. La preuve par récurrence

est elle aussi très bien passée en classe car les élèves l'avaient déjà assez souvent rencontrée.

VIII.2.2 Expérimentation dans la classe 2

Première heure

Pour l'algorithme du micro-ondes, les élèves n'ont pas voulu ajouter d'étapes supplémentaires par rapport à ce que nous avions préparé. Certains ont certes oublié des étapes, mais d'autres s'en sont aperçus et l'ont signalé.

Pour la recherche des différentes caractéristiques d'un algorithme, les élèves n'ont pas été très réactifs et ont eu beaucoup de mal à les trouver.

P : Nous allons essayer de donner les éléments qui caractérisent un algorithme.
E : C'est un peu comme une recette, une suite d'actions.
P : Oui ; par rapport au but, ça sert à quoi ?
E : Arriver à un objectif.
P : Ok ; ça marche avec les pâtes ? La soupe ?
E : Oui.
P : Donc, que pouvez-vous en dire ?
Es : ...
P : C'est juste valable pour des pâtes ?
E : C'est valable pour un ensemble de plats.
P : Voilà : un algorithme, c'est général.
(...)
P : Quelle est la famille d'instances pour le micro-ondes ?
E : L'alimentation.
P : Nous pouvons tout mettre au micro-ondes ?
E : Non.
P : Donc...
Es : ...
P : Les plats pour le micro-ondes. Et pour la voiture ?
E : Toutes les voitures.
P : Vraiment ?
E : Non, les voitures à clé.
P : Pourquoi une recette, ce n'est pas un algorithme ?
E : Ce n'est pas général.
(...)

P : Combien d'étapes y a-t-il pour l'algorithme de la voiture ?
E : 6.
P : Pas toujours.
E : Il y en aura toujours 5.
P : Peut-être pas. Combien au minimum ?
E : 4.
(...)
P : Alors, pour la dernière caractéristique, que pouvons-nous dire par rapport au nombre d'étapes ?
Es : ...
E : Il y a toujours un nombre d'étapes à respecter. Il ne peut pas y avoir deux étapes, il en faut au minimum 5.
P : Non, nous pourrions imaginer un algorithme en deux étapes.
E : Un début et une fin.
P : Donc le nombre d'étapes est ...
Es : ...
P : Pour la voiture, nous allons répéter les étapes indéfiniment ?
E_1 : Non, le nombre est limité.
E_2 : Ce n'est pas infini.

Pour la division euclidienne, les élèves ont su expliquer facilement le principe de la potence et des soustractions pour obtenir le quotient et le reste. Ils ont eu plus de mal à trouver la famille d'instances pour l'entrée.

P : Quelle est la famille d'instances au début du problème ?
E : Quotient et reste.
P : Ce qui est donné au début ?
E : Donc ici, c'est le quotient et le reste.
P : Ce qui est entré au début ?
E : Dividende et diviseur, des naturels.
P : Tous ?
E : Le diviseur ne peut pas être supérieur au dividende.
P : Ça, ce n'est pas grave, nous aurons un quotient de 0.
E : Sauf 0.
P : Les deux ?
E : Le diviseur.
(...)
L'algorithme de la division euclidienne est maintenant donné aux élèves.
P : Pour l'algorithme de la division euclidienne, l'entrée, ça représente quoi ?
Es : ...
P : Un terme spécifique ?

E : La famille d'instances.
P : Et la sortie ?
Es : ...
P : Par rapport au problème que nous nous posons ?
E : La réponse.

L'exécution de l'algorithme de la division euclidienne a posé assez bien de problèmes aux élèves car ils n'avaient pas compris qu'il fallait exécuter toutes les étapes de la boucle avant de repasser par la condition.

(...)
E : $17 \leftarrow 17 - 5$.
P : C'est ce que je vous ai expliqué il y a 5 minutes : le membre de gauche ne bouge pas.
E : $r \leftarrow 17 - 5$.
P : Nous passons où après ?
E : Étape 3.
P : C'est tout pour la boucle ?
E : Oui.
P : Dans la boucle, nous faisons tout ce qu'il y a dedans puis nous revenons au début.
E : On fait l'étape 5. On met dans q la valeur $12 + 1$.
P : Quelle est la valeur actuelle de q ?
E : 0.
P : À chaque fois, nous nous servons de la dernière valeur. Ensuite ?
E_1 : Étape 7.
E_2 : Non, la 1 !
E_3 : Non, la 3 !
P : Oui ; l'étape 7, ce sera quand la condition ne sera plus vraie. La 1, elle n'est pas dans la boucle, donc nous n'y passons plus.

Pour résumer cette heure, les élèves ont eu assez de mal à trouver les caractéristiques d'un algorithme sur la base des deux exemples. Cependant, comme pour la classe 1, nous ne nous attendions pas à ce qu'ils trouvent tout directement.
Pour l'exécution de l'algorithme de la division euclidienne, les élèves ont eu du mal avec l'affectation au début car ils voulaient remplacer le membre de gauche ou ils n'utilisaient pas les dernières valeurs des variables. Ils ne semblaient pas non plus avoir compris comment exécuter les opérations au sein d'une boucle « tant que ».

Deuxième heure

Les élèves n'ont eu aucun problème avec les notions d'entrée, de sortie et d'affectation vu que nous venions de les voir durant la première heure. L'écriture de l'algorithme pour le calcul de la somme des n premiers naturels avec la boucle « pour tout » n'a pas été évidente, même si les élèves avaient compris le fait qu'il fallait ajouter valeur après valeur. En leur posant des questions, ils ont réussi à trouver les différentes étapes.

P : En entrée, nous mettons quoi ?
E : Tous les nombres.
P : Les nombres ...
E : Naturels.
P : Pour la sortie ?
E : La somme des n premiers naturels.
P : Nous allons construire l'algorithme. Regardez pour le cas où $n = 5$, vous m'avez dit qu'à chaque fois, nous ajoutions une valeur.
E : À chaque fois, plus une valeur. La réponse précédente plus une valeur.
P : Par rapport aux affectations, comment pourrions-nous procéder ?
Es : ...
P : Si nous appelons le 1 de départ « somme », à l'étape suivante, ça deviendrait quoi ?
E : $somme + 2$, j'ajoute 2 à la somme.
P : Comment pouvons-nous faire pour le réutiliser après ?
E : Ça va devenir la somme.
P : Ok ; comment pourrions-nous faire pour généraliser maintenant ?
E_1 : $somme + n$.
E_2 : $somme + (n + 1)$.
P : Attention, n est fixé ! Il n'est pas interdit de définir de nouvelles variables.
E : On crée une nouvelle variable x pour ajouter à la somme. Elle va valoir 0 et à la fin de l'opération, on fait $x + 1$.
P : Ok ; et x varie comment ?
E : De 0 à n.
P : D'accord ; nous utilisons ce qui est appelé une boucle « pour tout ». Dans une boucle « pour tout », x augmente automatiquement. C'est tout pour l'algorithme maintenant ?
E : Il faut donner une réponse.
P : Nous marquons quoi alors ?
Es : Retourner somme.
P : C'est tout maintenant ? Regardez bien l'algorithme.
E : Il faut remplacer n par une valeur.

P : Non, ça c'est quand nous allons tester l'algorithme. Par rapport aux variables ?
Es : ....
E : x doit valoir 0 à la base.
P : Avec la boucle « pour tout », x va prendre la valeur 0 au premier passage.
E : Il faut mettre $x \leftarrow x + somme$.
P : Le x augmente automatiquement avec un « pour tout ». Il reste encore une variable...
E : La somme vaut 0 au début.
P : Oui, il faut l'initialiser, sinon nous ne pouvons pas nous en servir s'il n'y a pas de valeur dedans.

L'exécution de cet algorithme pour n valant 3 a également posé quelques problèmes aux élèves. Tout comme lors de l'exécution de l'algorithme de la division euclidienne, les élèves ont voulu exécuter plusieurs fois des opérations ne se trouvant pas dans la boucle.

P : Après l'étape 3, nous faisons quoi ?
E : La 1.
P : Sûrs ?
E : Non, la 2 !
(...)
E : Je mets 0+3 dans la somme.
P : Pourquoi 3 ?
E : Parce que x va jusque 3.
P : Oui, mais la valeur actuelle de x, c'est 1.
E : 0+1 alors.

Pour résoudre le problème consistant à calculer la somme des n premiers naturels, nous pouvons également utiliser une boucle « tant que ». La construction de cet algorithme s'est bien passée car les élèves avaient directement de bonnes idées.

P : Nous commençons par quoi ?
E_1 : Mettre 0 dans la somme.
E_2 : On met pas 1 ?
P : Si nous voulions la somme jusque 0, nous obtiendrions 1 sinon. Vous pouvez faire les initialisations en dernier, c'est parfois plus facile.
E : Tant que $x \leqslant 3$.
P : Dans le cas général avec n ?
E_1 : Tant que $x \leqslant n$ faire.

E_2 : $x + somme$.
P : Pour quelle variable ?
E : x.
P : Non, sinon ça n'augmenterait pas de 1.
E_1 : Mettre dans la somme la valeur $somme + x$.
E_2 : Il faut mettre qu'à chaque fois, x augmente de 1.
P : Ça se note comment ?
E_1 : On met $x + 1$ dans x.
E_2 : Au début, il faut mettre que x vaut 0.
P : Oui, il faut l'initialiser. Ça vous paraît complet ?
E : « Fin tant que » et « retourner somme ».

Les élèves n'ont pas voulu exécuter cet algorithme car nous avions déjà exécuté un algorithme avec ce type de boucle durant la première heure. Nous avons alors introduit les exercices de la troisième heure.

P : Maintenant, je ne vous donne plus la sortie. Comment faire pour la trouver ?
E : Résoudre des équations.
P : Où vois-tu des équations ?
Es : ...
E : Il faut faire un exercice.
P : Quel exercice ?
Es : ...
P : Avant, je vous donnais l'algorithme et nous l'exécutions pour des valeurs que je vous donnais. Ici, nous ne voyons pas la sortie en regardant juste l'algorithme.
E : Il faut faire un exercice.
P : Lequel ?
E : J'ai besoin de valeurs.
P : Oui ; nous pouvons essayer pour toutes les valeurs ?
E : Non, ça dépend de la condition d'entrée.
P : Oui.

Durant cette heure de cours, nous avons constaté que souvent, les élèves mélangeaient les cas particuliers et le cas général : ils voulaient construire un algorithme pour un cas particulier. La construction d'algorithmes s'est plutôt bien passée au final car les élèves avaient de bonnes idées. Nous avons parfois dû les guider en leur posant des questions.
L'exécution de la boucle « pour tout » a posé problème car les élèves voulaient, une fois encore, exécuter à plusieurs reprises une opération ne se

trouvant pas dans la boucle.

Troisième heure

Durant cette troisième heure, nous avons exécuté les trois algorithmes proposés en exercices. L'algorithme avec la structure « si...alors...sinon » a posé problème à un élève n'ayant visiblement pas lu le paragraphe sur cette structure dans les feuilles. Cet élève, avec la condition $x \geqslant y$, voulait seulement remplacer la variable y par sa valeur parce qu'il avait retenu que pour l'affectation, le membre de gauche ne changeait pas.
Pour l'algorithme avec le « pour tout », certains élèves voulaient à nouveau répéter plusieurs fois une opération située hors de la boucle et ils ont aussi eu du mal à comprendre ce qu'il se passait avec la variable k qui ne se trouvait pas dans la boucle.
L'exécution de l'algorithme avec le « tant que » n'a posé aucun problème aux élèves.

Enfin, ils ont eu assez bien de mal à trouver les sorties des différents algorithmes. Pour les algorithmes 8 et 9, ils y sont parvenus, mais seulement après quelques minutes de réflexion. Ils ont eu beaucoup plus de mal à trouver la sortie de l'algorithme 10 et nous avons dû les guider.

P : Quelle est la sortie de l'algorithme 10 ?
Les élèves l'ont exécuté avec les valeurs $a = 4$ et $b = 3$; ils ont obtenu 1.
E_1 : $a - b$.
E_2 : $|a - b|$.
P : Je vais vous proposer de tester à nouveau l'algorithme avec $a = 6$ et $b = 2$.
Après quelques minutes, les élèves ont obtenu 2 comme réponse.
P : Vos propositions marchent toujours avec cet exemple ?
Es : Non.
E : Peut-être $\sqrt{|a - b|}$.
P : Ok ; je vous donne d'autres résultats : si $a = 9$ et $b = 6$, nous obtenons 3. Si $a = 4$ et $b = 2$, nous obtenons 2. Vous voyez maintenant ?
Es : ...
P : Encore un autre exemple : si $a = 15$ et $b = 10$, ça donne 5.
E : La réponse plus b et ça donne a.
P : Ça revient à faire $a - b$ et ça ne marche pas pour $a = 6$ et $b = 2$.
E : Ha oui...
P : Les valeurs que nous obtenons sont-elles plus petites ou plus grandes que les valeurs de départ ?

E : Plus petites.
P : Regardez... 2 par rapport à 2 et 6 ?
E_1 : Ce sont des multiples.
E_2 : C'est le PPCM !
P : Non, le PPCM, ça va être plus grand que les valeurs, c'est un multiple.
Es : Le PGCD !

En résumé, certains élèves ont eu des difficultés lors de l'exécution de certains algorithmes, mais nous pensons qu'ils n'avaient pas lu nos feuilles et qu'ils n'avaient pas relu leurs notes. En effet, nous pensons que s'ils avaient relu leurs notes, ils n'auraient pas voulu exécuter plusieurs fois une opération se situant hors d'une boucle puisqu'ils avaient déjà commis cette erreur lors de la deuxième heure. Les élèves ont aussi eu énormément de mal à trouver les sorties des différents algorithmes, mais nous pensons qu'ils n'avaient pas réellement compris le but de l'exercice.

Quatrième heure

Au début de cette heure, après avoir convaincu les élèves que des exemples ne suffisaient pas à affirmer qu'un algorithme était correct, nous les avons invités à nous dire qu'il fallait également vérifier que le nombre d'étapes était fini.

P : Par rapport à la définition d'algorithme donnée, rien ne vous trouble avec ce que nous avons fait ?
E : Pour toute instance.
P : C'est-à-dire ?
E : Toute la famille d'instances.
P : Les algorithmes que nous avons travaillés, ils étaient généraux ?
E : Oui.
P : Donc pas de problèmes pour la famille d'instances.
E : Solution à un problème.
P : Les algorithmes 8, 9 et 10 sont-ils solutions d'un problème ?
E : On n'a pas donné le problème.
P : Il n'a pas été donné, mais nous le cherchions. Par exemple, l'algorithme 8 sert à trouver le maximum entre deux valeurs.
E : Le nombre fini d'étapes.
P : Oui ! Comment voir si le nombre d'étapes est fini ?
E : Ici, on passe deux fois dans la boucle.
P : Oui, mais en général, que faire ?
E : Les compter.

La preuve de la terminaison n'a pas posé de problèmes. En ce qui concerne la preuve par récurrence, l'enseignante 2 n'avait pas eu l'occasion d'en travailler énormément en classe et les élèves ont donc eu beaucoup de mal avec cette preuve. Au final, ils n'ont pas vraiment participé à la démonstration.

Nous avons noté l'assertion devant être vérifiée après chaque passage dans la boucle.
P : Pour le cas de base, il faut montrer quoi ?
E : Là, c'est $0 - 1$.
P : Non, nous faisons le cas de base pour l'instant, pas encore le pas de récurrence. Si k vaut 0, que vaut y ?
E : 0 euh ... 1 !
(...)
P : Si k vaut 0, où sommes-nous ?
Es : ...
P : Dans la boucle, k prend quelles valeurs ?
E_1 : De 1 à n.
E_2 : Donc, on n'est pas encore dans la boucle.
P : Oui, donc quelles étapes sont à exécuter ?
E : 1 et 5.
P : Attention, nous ne sommes pas encore dans la boucle.
E : La 1 alors ; je croyais qu'on prenait à l'extérieur.
(...)
P : Que signifie que la propriété est vraie en $k - 1$?
Es : ...
(...)
P : Nous allons montrer que la propriété est vraie en k, c'est-à-dire ?
Es : ...
(...)
P : Au $(k-1)^{\text{ème}}$ passage, que vaut y ?
E_1 : k.
E_2 : x^{k-1}.
(...)
P : Maintenant que nous avons prouvé que cette propriété était vraie après chaque passage dans la boucle, comment montrer ce que nous voulions montrer à la base ?
E : $y = x^k$.
P : Et en sortant de la boucle ?
E : x^n

P : Pourquoi ?
E : k, c'est n.
P : Pourquoi ?
Es : ...
P : En sortant de la boucle, que vaut k ?
E : n.
P : Donc la dernière valeur de k est n et nous avons bien le résultat.

En résumé, les élèves s'en sont très bien sortis avec la terminaison, mais la correction a posé d'énormes soucis. En effet, les élèves, peu habitués aux preuves par récurrence, n'ont pas compris ce qu'il fallait montrer et ont donc très peu participé à la preuve. Ils n'ont pas vu comment adapter la propriété à démontrer pour prouver le pas de récurrence. La preuve par récurrence ne semble donc pas du tout acquise dans cette classe.

Cinquième heure

Les élèves de cette classe sont parvenus à trouver des critères pour former les différents couples, même si au début, leurs critères n'étaient pas généraux.

P : Quels critères pourrions-nous choisir pour former les couples ?
E : Alain et Éline.
P : En général ?
E : Des critères de préférences.
P : Par rapport aux tableaux ?
E : Le plus de premiers choix.
P : Oui, ça peut être un critère. Et à l'inverse ?
E : Minimiser le nombre de derniers choix.
P : Oui. D'autres critères sont encore envisageables, mais nous allons choisir le critère de la stabilité.

Une fois la stabilité définie, nous avons proposé aux élèves l'exemple d'ensemble de couples instable repris dans nos feuilles pour voir s'ils avaient bien compris la définition.

P : Pourquoi cet ensemble est instable ?
Es : ...
P : Par rapport à la définition ?
E : Car chaque personne n'est reprise qu'une fois.
P : C'est normal, chaque personne ne va être en couple qu'avec une seule personne.

E : Ils ont choisi leurs partenaires.
P : Non, ça c'est nous, les gestionnaires du site, qui avons formé ces couples.
Es : ...
P : Par rapport aux tableaux ?
E_1 : A serait mieux avec D.
E_2 : Mais il y aura toujours quelqu'un de pas content car toutes les filles ont Cyrille en dernier.
P : La stabilité, ça concerne deux personnes. Les deux doivent avoir envie de bouger en même temps. Un exemple ?
E : A et D.

L'exécution de l'algorithme sur le premier exemple s'est relativement bien passée. Par contre, les élèves n'étaient pas vraiment convaincus que l'ensemble de couples trouvé au moyen de l'algorithme était bien stable.

P : Notre ensemble est-il stable ? Est-ce que deux personnes préféreraient être ensemble ?
E : A et B.
P : Ce sont deux hommes !
E : B-D et A-E.
P : Mais en fait, B n'a pas envie de bouger puisqu'il est avec son premier choix.
E : Oui, mais Éline préfère être avec Alain.
P : Oui, mais Alain est bien avec Delphine.
E : En fait, ce sont les hommes qui choisissent !
P : Eh oui, cet algorithme est masculin...

Finalement, pour la terminaison de cet algorithme, les élèves n'ont jamais trouvé que la boucle était empruntée n^2 fois au maximum, même en leur montrant, en dernier recours, que les tableaux de préférences comme présentés sur les feuilles formaient des carrés. Nous avons donc préféré leur donner la réponse et l'expliquer plutôt que de les laisser la trouver par hasard.

En résumé, l'exécution de l'algorithme s'est bien passée : les élèves savent exécuter un algorithme, même s'il est exprimé en français. Par contre, ils ont eu beaucoup de mal à comprendre la définition de la stabilité et n'ont donc pas su appliquer cette définition. Quant à la terminaison, cela n'a pas été un franc succès...

Conclusion de cette expérimentation

Les élèves de la classe 2 avaient parfois peur de s'exprimer, ce qui a provoqué de longs moments de silence durant les cinq heures de cours. Au final, il n'y a pratiquement que deux élèves qui participaient activement au cours et qui essayaient de faire avancer les discutions.
L'exécution d'algorithmes semble avoir posé problème. Déjà, l'affectation a posé problème durant la première heure, mais les élèves en ont ensuite bien compris le principe. En outre, certains élèves ont semblé n'avoir rien revu chez eux et ont donc commis des erreurs qu'ils n'auraient jamais faites s'ils avaient un peu travaillé, notamment avec le « si...alors...sinon » et les opérations hors d'une boucle exécutées plusieurs fois. Cependant, chez certains élèves ayant retravaillé leur cours, les problèmes liés à l'exécution d'un algorithme se sont vite résorbés.
La preuve par récurrence est très mal passée dans cette classe et nous avons quasiment dû l'effectuer sans l'aide des élèves. Nous avions l'impression qu'ils découvraient cette preuve alors qu'ils l'avaient déjà rencontrée avec leur enseignante.

VIII.2.3 Expérimentation dans la classe 3

Première heure

Pour l'algorithme du micro-ondes, les élèves ont voulu sauter des étapes en oubliant de mettre le plat dans le micro-ondes, mais ils se sont vite aperçus de leur oubli. Ils ont également rajouté l'étape « ouvrir la porte » au début et « fermer la porte » à la fin de l'algorithme.
En ce qui concerne les différentes caractéristiques d'un algorithme, les élèves les ont assez vite trouvées.

P : Quels sont les éléments qui caractérisent un algorithme ?
E_1 : C'est un enchaînement d'événements, d'étapes.
E_2 : Une marche à suivre.
P : Oui et dans quel but ?
E_1 : Arriver à sa fin.
E_2 : Résoudre un problème.
P : Oui ; pouvons-nous faire chauffer des pâtes ? Une soupe ?
Es : Oui.
P : Que pouvez-vous en dire alors ?
E_1 : Pas spécifique.

E_2 : Général.
P : Oui, il est valable pour une classe de problèmes.
(...)
P : Pour le micro-ondes, quelle est la famille d'instances ?
E : L'intensité.
P : La famille d'instances, c'est tout ce que nous pouvons « mettre » dans l'algorithme.
E : L'ensemble des configurations possibles du problème.
P : Oui, ça c'est la définition que je viens de donner. Par rapport au problème ?
Es : ...
P : Bon... Tous les plats que nous pouvons mettre chauffer au micro-ondes. Et pour la voiture ?
E_1 : Toutes les voitures.
E_2 : À clé !
(...)
P : Il nous manque encore une caractéristique... Regardez par rapport aux étapes.
E_1 : C'est dans un ordre.
E_2 : Ça redémarre.
P : Toujours ?
Es : Non !
E : Ça a un début et une fin.
P : Oui, et donc, par rapport au nombre d'étapes ? Pouvons-nous recommencer indéfiniment pour la voiture ?
Es : Non.
E_1 : Le nombre d'étapes est limité.
E_2 : Fini.
(...)
P : En fait, combien y a-t-il d'étapes pour la voiture ?
E : 6.
P : Toujours ?
Es : ...
P : Par rapport à l'algorithme, il y en a toujours 6 ? Si ça démarre tout de suite, combien d'étapes ?
E : 4.
P : Oui, et dans le pire des cas, ça va aussi s'arrêter.

Comme dans les deux classes précédentes, nous avons aussi demandé aux élèves, sur la base de la définition d'algorithme donnée, pourquoi la recette de cuisine n'en était pas un. Contrairement aux classes précédentes,

les élèves de la classe 3 n'ont pas trouvé directement la réponse.

P : Pourquoi la recette de cuisine n'est pas un véritable algorithme ?
E : Elle ne donne pas la solution à un problème.
P : Si, c'est d'arriver à un gâteau, par exemple.
E : C'est pas pour toute instance.
P : Oui, une recette, ce n'est pas général.

Ensuite, nous en arrivons à la division euclidienne. Les élèves ont également pu expliquer comment effectuer la division de 17 par 5 au moyen de soustractions et n'ont pas eu trop de mal à trouver l'entrée et la sortie.

P : Elles doivent appartenir à quoi ces valeurs ?
E : Tous les naturels.
P : Tous ?
E_1 : Ils ne peuvent pas être pareils.
E_2 : $a > b$.
P : Ce n'est pas grave.
E_1 : $b \neq 0$.
E_2 : Et si a vaut 0 ?
P : Ce n'est pas grave, 0 peut être divisé.
(...)
L'algorithme de la division euclidienne est maintenant donné aux élèves.
P : Alors, l'entrée ça correspond à quoi par rapport à ce que nous avons dit avant ?
E : La famille d'instances.
P : Et la sortie ?
E : La solution.

Lors de l'exécution de l'algorithme, les élèves n'ont eu du mal qu'avec l'affectation.

P : Quelle est la première étape ?
E : $17 - 5$
P : La première étape de l'algorithme.
E : $5 \leftarrow 17$.
P : Bon, l'étape 1 consiste à mettre quelle valeur dans la variable r ?
E : a.
P : Oui ; le membre de gauche d'une affectation ne change jamais.
E : Donc $r \leftarrow 17$.

Durant cette première heure, nous avons donc constaté que les élèves étaient très réactifs et ils étaient toujours plusieurs à répondre en même temps aux questions. Les élèves ont trouvé toutes les caractéristiques d'un algorithme assez rapidement.
Ils ont éprouvé quelques difficultés avec l'affectation, mais elles se sont rapidement résorbées. La boucle « tant que » n'a, semble-t-il, pas posé de problèmes aux élèves après que nous leur en ayons expliqué le principe.

Deuxième heure

Le rappel sur les notions d'entrée, de sortie et d'affectation a été assez rapide étant donné que nous venions de voir ces notions durant la première heure.
Pour l'écriture de l'algorithme permettant de calculer la somme des n premiers naturels à l'aide de la boucle « pour tout », les élèves n'ont pas vraiment eu de mal après avoir compris qu'il fallait ajouter les valeurs au fur et à mesure.

P : Quelle est la première chose à mettre dans un algorithme ?
E : Entrée et sortie.
P : Nous mettons quoi pour l'entrée ?
E : Un naturel.
P : Et pour la sortie ?
E : Un naturel.
P : La sortie doit donner la solution, nous pouvons la mettre en français.
E : La somme.
(...)
P : Que pourrions-nous faire pour construire l'algorithme par rapport aux affectations ?
E : On fait $1 + 2$ puis on reprend la valeur, on ajoute 3. On prend une variable x pour mettre la somme.
P : Oui, c'est ça la base. Nous allons ajouter i ; i va varier comment alors ?
E : De 1 à n.
(...)
P : C'est tout pour l'algorithme ?
E : Retourner.
P : Retourner quoi ?
Es : ...
P : Par rapport à la sortie, il faut retourner ...
E : La somme.
P : Oui, donc x. Il manque encore quelque chose.

Es : ...
P : Regardez par rapport aux variables.
E : Tant que $x = n$.
P : Nous verrons l'algorithme avec la boucle « tant que » après. Alors ?
Es : ...
P : Quand nous utilisons le x pour la première fois ?
E : Le x ne sera pas défini au début.
P : Oui, nous mettons quoi ?
E_1 : 1.
E_2 : 0.
P : Oui, 0, parce que sinon, si n vaut 1, nous aurions que la somme jusque 1 vaut 2.

L'exécution de cet algorithme n'a posé aucun souci en classe. L'écriture de l'algorithme résolvant le même problème, mais écrit avec une boucle « tant que » s'est également relativement bien passée, même si parfois, il a fallu poser des questions aux élèves pour les remettre dans le droit chemin.

E : On fait $x \leftarrow 1 + x$, jusque n.
P : C'est le x qui contient la somme ; qui va valoir n à la fin ?
E : Tant que $i = n$.
P : Le i vaut n tout de suite ?
E : Non, $i \leqslant n$.
P : Nous avons le « tant que », nous faisons quoi après ?
E : Mettre 0 dans x.
P : Oui ; ensuite ?
E_1 : « Fin tant que » et « retourner x ».
E_2 : Ça sert à quoi le « fin tant que » ?
P : Pour délimiter, pour voir de où à où ça va. Il manque encore quelque chose par rapport aux variables...
Es : ...
P : Concernant le i.
E : Il va de 1 à n.
P : Oui, mais nous ne l'avons mis nulle part.
E : On met au-dessus $i \leftarrow 1$.
P : Et comment faire varier i ? Sinon i va toujours valoir 1.
E : $i + 1$.
P : Dans quoi ?
E_1 : Dans x.
E_2 : Dans i.
P : Oui, à chaque fois, i doit augmenter.

Nous avons exécuté cet algorithme à la demande des élèves. L'exécution n'a cependant pas posé de problèmes puisque nous avions déjà exécuté un algorithme avec une boucle « tant que ».

En résumé, cette heure de cours s'est assez bien passée puisque les élèves avaient de bonnes idées pour construire les deux algorithmes. Un bémol toutefois : parfois, ils semblent avoir du mal à comprendre qu'il est possible d'attribuer à une variable une valeur contenant sa valeur actuelle (par exemple, $i \leftarrow i + 1$).
L'exécution des deux algorithmes s'est très bien passée et nous pouvons donc espérer que la troisième heure ne pose pas trop de soucis.

Troisième heure

Pour commencer l'heure, nous avons d'abord demandé aux élèves comment ils pouvaient faire pour déterminer la sortie d'un algorithme. Ils ont eu des réactions très rapides, mais nous avons quand même dû leur poser des questions pour les faire avancer dans leurs réflexions.

P : Comment pouvons-nous déterminer la sortie ?
E : Déterminer l'entrée.
P : Non, elle vous est donnée ; regardez les algorithmes.
E : En faisant l'algorithme.
P : Oui ; vous voyez, juste en regardant l'algorithme, ce qu'il fait ?
E : On remplace les valeurs.
P : Par quoi ?
E : Avec la condition.
P : Laquelle ?
E : $x \geqslant y$.
P : Oui, mais comment tu fais si tu n'as pas de valeurs pour x et y ?
E : On en prend une.
P : Comment nous choisissons ?
E : Deux réels.
P : Ok pour l'algorithme 8.

L'exécution de l'algorithme 8 avec la structure conditionnelle n'a pas posé de problèmes aux élèves et nous l'avons exécuté avec deux couples de valeurs différents car les élèves n'arrivaient pas à trouver la sortie. Par contre, l'exécution de l'algorithme 9 a posé quelques soucis aux élèves, notamment à cause de la variable k non présente dans la boucle. Les élèves

ont tout de même pu trouver la sortie facilement.

P : k va de 1 à n. Nous commençons à quelle valeur alors pour k ?
E_1 : 1.
E_2 : Comment on sait la valeur de y ?
P : C'est la dernière valeur de y par affectation.
E : Et le k, il intervient nulle part ?
P : Il sert juste à répéter les étapes.
(...)
E : Ça aurait été plus simple si vous aviez mis que l'algorithme donnait x^n...
P : C'était justement ça, le but de l'exercice !

Enfin, les élèves sont parvenus à exécuter l'algorithme 10 sans problèmes majeurs. Ils en ont assez rapidement trouvé la sortie.

E : La sortie, c'est $a - b$.
P : Donnons un autre exemple... Si $b = 1$ et $a = 4$, ça donne 1.
E_1 : Le PPCM.
E_2 : Le PGCD.
P : Le PPCM, ça doit être un multiple alors.
P *(Parlant à l'autre élève)* : Oui, c'est ça, mais laisse chercher les autres !
P *(À tous)* : Regardez 10 et 15 par rapport à 5.
E_1 : Plus petit commun diviseur.
E_2 : Plus grand !

En résumé, les élèves de cette classe ne semblent pas avoir éprouvé d'énormes difficultés pour exécuter les algorithmes que nous leur avons proposés. Ils sont parvenus assez rapidement à en trouver les différentes sorties. L'exécution d'algorithmes et la généralisation sur la base d'exemples semblent donc être acquises.

Quatrième heure

Au début de l'heure, nous avons introduit la correction des algorithmes en convainquant les élèves que des exemples n'étaient pas suffisants pour nous assurer que la sortie trouvée était bien la sortie exacte, valable pour toutes les instances des problèmes. Ensuite, nous leur avons demandé de relire la définition d'« algorithme » afin de leur faire dire que nous n'avions jamais vérifié que le nombre d'étapes était fini.

P : Reprenez la définition d'« algorithme ». Nous avons dit que les algorithmes 8, 9 et 10 étaient des algorithmes, mais nous n'avons pas vérifié. Regardez s'il n'y a pas un problème.
E : Pour toute instance.
P : Ça veut dire que les algorithmes doivent être généraux. Ils le sont ?
E : On ne sait pas.
P : Pour le 8, x et y sont réels.
E : C'est un cas particulier.
P : Est-ce que l'algorithme est uniquement valable pour $x = 3$ et $y = 4$?
E : Non, alors il est général.
P : Donc, ce n'est pas ça le problème.
Es : ...
E : Le nombre d'étapes.

La preuve de la terminaison de l'algorithme 9 s'est assez bien passée, même si les élèves ont eu du mal à voir que le nombre d'étapes était fini.

P : Comment faire pour voir si le nombre d'étapes est fini ?
E : Regarder l'algorithme.
P : Oui, mais pour voir si le nombre d'étapes est fini ?
Es : ...
P : En regardant juste l'algorithme... Que pourrions-nous tout bêtement faire ?
E : Compter les étapes.
P : Oui.
(...)
P : Combien de fois la boucle est-elle exécutée ?
E : 3 fois.
P : Dans le cas général ?
E : n fois.
(...)
P : Alors, le nombre d'étapes est fini ?
E : Non, n peut prendre n'importe quoi comme valeur.
P : n, c'est un naturel.
E : Ça peut être 1252.
P : Et c'est fini ?
E : Ha oui !

Les élèves de la classe 3 n'avaient pas eu l'occasion de travailler énormément de preuves par récurrence avec leur enseignante et ils semblaient donc totalement perdus lors de la preuve de la correction de l'algorithme 9.

P : Pour le cas de base, k vaut 0. Nous allons montrer quoi ?
Es : ...
E : $y = 0$.
P : Si k vaut 0 ?
E : $y = 1$.
(...)
P : Par rapport à la boucle, si k vaut 0, où sommes-nous ?
E : Avant la boucle.
P : Oui ; quelles sont donc les étapes à faire ?
E : La 1 et la 2.
P : La 2 si k vaut 0 ?
E : Non.
(...)
P : Pour le pas de récurrence, nous allons supposer quoi ?
Es : ...
P : Nous supposons qu'au $(k-1)^{\text{ème}}$ passage dans la boucle, nous aurons ...
Es : ...
P : $y = x^{k-1}$. Il faut montrer quoi ?
E : Au $k^{\text{ème}}$ passage dans la boucle, nous aurons $y = x^k$.
(...)
P : Lors du $k^{\text{ème}}$ passage, nous avons quelle opération ?
Es : ...
E : $y \leftarrow y \times x$.
P : Vous pouvez en dire plus ? Puisque nous venons du $(k-1)^{\text{ème}}$ passage, avec l'hypothèse de récurrence...
Es : ...
P : Par rapport aux variables ?
Es : ...
P : Au $(k-1)^{\text{ème}}$ passage, nous avons $y = x^{k-1}$. Nous passons à nouveau dans la boucle et cette opération-là est effectuée ($y \leftarrow y \times x$). Que pouvons-nous remplacer ?
E : Le y.
P : Oui, par quoi ?
E : Par ... x^{k-1} ?
P : Voilà !
(...)
P : C'est tout par rapport à ce que nous voulions montrer au début ?
Es : Euh... Oui.
P : Nous avons montré qu'une assertion était vraie à chaque passage dans la boucle, mais nous voulons montrer qu'à la fin, nous avons $y = x^n$.

E : Ça ne change rien, c'est un changement de variables.
P : Non, les variables ne se changent pas ainsi. À la fin de l'algorithme, que vaut k ?
E : n.
P : Donc, nous avons bien l'égalité à la fin.

En résumé, les élèves n'ont pas eu de gros problèmes pour prouver la terminaison de l'algorithme 9. Par contre, il n'en a pas été de même en ce qui concerne la preuve de la correction de cet algorithme. En effet, les élèves n'avaient pas souvent eu l'occasion de travailler la preuve par récurrence et ont donc été très peu réactifs durant cette heure de cours. Ils ont eu du mal à comprendre ce que signifiait l'hypothèse de récurrence et n'ont pas su l'exploiter. Enfin, ils n'ont pas compris le lien entre la correction de l'algorithme et l'assertion vraie à chaque passage dans la boucle que nous avions prouvée à l'aide de la récurrence.

Cinquième heure

Durant cette dernière heure, seuls 9 étudiants sur les 14 étaient présents. Les réponses ont donc moins fusé que durant les autres heures de cours.
Les élèves ne sont pas parvenus à trouver de critères pour former les couples. Nous leur en avons donc donné quelques uns.

P : Quels critères pourrions-nous envisager pour former les couples ?
E : L'algorithme.
P : Ce n'est pas un critère.
Es : ...
P : Nous pourrions essayer de mettre le plus de personnes avec leurs premiers choix. Vous avez un autre critère à proposer ?
Es : ...
P : Si nous maximisons le nombre de premiers choix, à l'inverse, nous pourrions ...
E : On va mettre C et F ensemble.
P : Oui. Bon, un autre critère serait de minimiser le nombre de derniers choix.

Les élèves ont ensuite eu un peu de mal à comprendre la définition d'un ensemble stable de couples. Ils n'ont pas compris directement pourquoi l'ensemble que nous leur avons proposé était instable.

P : Cet ensemble est instable. Pourquoi ?
E : Il y en a un qui est d'accord et l'autre non.

P : Oui ; vous voyez deux personnes qui seraient mieux ensemble ?
E_1 : A-F, ça pose problème.
E_2 : D-C aussi.
E_1 : Mais à chaque fois, il n'y en a qu'un qui a envie de bouger.
P : La stabilité, c'est une personne de chaque couple qui a envie de bouger.
E : Ha ! Donc A et D.

L'exécution de l'algorithme sur le premier exemple s'est passée sans trop de difficultés. Les élèves ont même pu expliquer, une fois l'ensemble de couples trouvé, pourquoi il était bien stable.
Pour trouver le nombre d'exécutions de la boucle dans l'algorithme, un élève a assez rapidement trouvé le n^2. Nous avons ensuite convaincu le reste de la classe que sa réponse était bonne en leur demandant le nombre de choix possibles pour un homme et le nombre d'hommes dans le problème, le tout dans le cas général avec n couples à former.

En résumé, au début de cette heure de cours, les élèves n'étaient pas très imaginatifs pour trouver des critères. Ils ont également eu un peu de mal à comprendre la définition de la stabilité, mais ensuite, ils sont parvenus à l'appliquer sans trop de mal. L'exécution d'un algorithme, même s'il est assez différent de ceux rencontrés précédemment, ne semble pas poser de problèmes aux élèves. La preuve de la terminaison s'est, elle aussi, déroulée sans trop d'accrocs.

Conclusion de cette expérimentation

Durant les premières heures, les élèves de la classe 3 ont été très réactifs et ils répondaient souvent à plusieurs aux questions. Au fur et à mesure des heures, ils ont réagi de moins en moins et nous avons dû les guider pour qu'ils trouvent les réponses aux différentes questions que nous leur posions.
L'exécution d'algorithmes n'a pas posé de problèmes majeurs aux élèves. Ils ont juste eu du mal avec l'affectation durant la première heure, mais sinon, ils ont très bien compris les différentes structures.
L'heure consacrée à la preuve d'un algorithme a été la plus dure dans cette classe. D'une part, pour la terminaison, les élèves ont pu déterminer facilement le nombre d'étapes, mais ils ont eu du mal à conclure. D'autre part, pour la preuve de la correction, les élèves étaient peu habitués à la preuve par récurrence. De ce fait, ils ne semblent pas avoir compris ce que nous essayions de montrer et ils n'ont pas su utiliser l'hypothèse de récurrence à bon escient.

VIII.3 Bilan

Dans ce chapitre, nous avons donc présenté les différentes expérimentations effectuées de notre séquence d'enseignement présentée au chapitre précédent. Nous l'avons expérimentée dans trois classes : la classe 1 (sixième secondaire), la classe 2 (cinquième secondaire) et la classe 3 (sixième secondaire). Ce sont trois classes avec plus de 6 heures de mathématiques par semaine.

Après ces quelques heures de cours, l'exécution d'un algorithme semble être maîtrisée par la majorité des élèves, même s'ils ont eu du mal au début. Dans les trois classes, l'affectation a posé des problèmes durant la première heure car les élèves voulaient modifier le membre de gauche.
En ce qui concerne les différentes structures, l'exécution d'un algorithme avec un « si...alors...sinon » a été problématique uniquement dans la classe 2 puisque des élèves confondaient la condition du « si » avec une affectation.
L'exécution des boucles a été plus délicate, notamment dans la classe 2 encore, où les élèves ont voulu, à plusieurs reprises, répéter des opérations ne se trouvant pas dans une boucle. Toutes les classes ont également eu du mal à comprendre l'algorithme 9 car il comporte une boucle « pour tout » avec une variable qui n'est pas présente dans le corps de la boucle.
Les élèves ont en général pu trouver les différentes sorties sur la base de quelques exécutions des algorithmes.

L'écriture d'algorithmes n'a pas été une tâche aisée dans les trois classes. En effet, nous avons dû poser assez bien de questions aux élèves pour qu'ils créent entièrement les algorithmes. Cependant, au final, dans les trois classes, les différentes opérations ont toutes - ou presque - été trouvées par les élèves. Nous ne pensons pas forcément que l'écriture d'algorithmes soit acquise dans ces classes, mais nous pensons qu'elle est en voie d'acquisition car les élèves semblent avoir compris la manière de concevoir un algorithme pas à pas.

Enfin, la preuve d'un algorithme a été la partie la plus délicate de ces différentes expérimentations. Même si les preuves de la terminaison n'ont pas posé trop de soucis aux élèves, il n'en a pas été de même pour les preuves de la correction. En effet, celle que nous leur avons proposée employait la preuve par récurrence. Cette preuve n'a pas posé de problèmes dans la classe 1 car les élèves y étaient habitués. Dans les deux autres classes, les élèves ne sont pas parvenus à l'effectuer, ne voyant pas, bien souvent, ce qu'il fallait démontrer.

Dès lors, nous allons relater les évaluations dans les trois classes afin de voir si les compétences travaillées durant les expérimentations sont acquises ou en voie de l'être chez les élèves. C'est l'objet du prochain chapitre.

Chapitre IX

Évaluation de la séquence de cours

Dans ce chapitre, nous présentons le questionnaire que nous avons proposé aux élèves durant la dernière heure de notre expérimentation. Nous relatons également les réponses des élèves aux différentes questions.

IX.1 Présentation du questionnaire

Nous proposons aux élèves un questionnaire en deux parties : la première comporte des questions sur la matière vue durant les cinq heures de cours et la deuxième comporte des questions pour obtenir leur avis sur la séquence d'enseignement. Nous détaillons ces deux parties.

Première partie : questions sur la matière

Nous prévoyons deux questions pour évaluer les élèves sur la matière que nous leur avons enseignée. Pour la première, nous leur donnons l'algorithme ci-après et nous leur posons les questions suivantes :

(a) Que retourne l'algorithme avec les entrées $x = 2$ et $y = 5$?

(b) Que retourne l'algorithme avec les entrées $x = 3$ et $y = 4$?

(c) Que retourne l'algorithme avec les entrées $x = 0$ et $y = 2$?

(d) Dans le cas général, que calcule l'algorithme ?

(e) Pour en prouver la correction, quelle technique de preuve pourriez-vous utiliser ? Expliquez-en les grands principes adaptés à cet exemple.

Algorithme

Entrée : deux nombres naturels non nuls x et y.
Sortie : ? ? ?

1. $t \leftarrow x$
2. **pour tout** j allant de 1 à y **faire**
3. $\quad t \leftarrow t/x$
4. **fin pour tout**
5. **retourner** t

Pour les points (a), (b) et (c), il s'agit d'exécuter l'algorithme avec les valeurs données. La réponse au point (a) est $\frac{1}{16}$, celle du point (b) est $\frac{1}{27}$ et celle du point (c) est que 0 n'est pas une entrée acceptable. Nous nous attendons à ce que ces trois points soient réussis par quasiment tous les élèves car l'algorithme est assez similaire à l'algorithme 9 rencontré durant les heures de cours. Nous avons changé les noms des variables afin que les élèves exécutent réellement l'algorithme sans essayer de faire des parallèles avec l'algorithme 9.
Pour le point (c), nous pensons que les élèves feront référence à l'entrée car nous avons fortement insisté sur cette notion durant les heures de cours.

Le point (d) consiste à trouver la sortie générale de l'algorithme sur la base des points précédents. La réponse correcte est $t = \frac{1}{x^{y-1}}$. Nous pensons que les élèves n'auront pas trop de mal à trouver cette réponse avec les calculs qu'ils auront effectués aux points précédents.

Enfin, pour le point (e), nous attendons des élèves qu'ils parlent de la preuve par récurrence. Comme nous avons constaté que ce type de preuve avait posé quelques soucis aux élèves durant nos expérimentations et que beaucoup d'étudiants universitaires ne la maitrisaient pas après plusieurs cours sur le sujet, nous ne pouvons pas exiger des élèves qu'ils produisent l'entièreté d'une preuve par récurrence, surtout que nous n'avons eu l'occasion de n'en travailler qu'une seule avec eux. Cependant, nous voulons quand même voir s'ils en ont compris le principe. Nous attendons donc qu'ils nous disent qu'il faut effectuer la preuve par récurrence sur le nombre de passages dans la boucle. Nous voulons aussi qu'ils nous parlent du cas de base et du pas de récurrence en expliquant brièvement ce qu'il y a à montrer. Nous ne pensons pas que cette question soit réellement compliquée,

mais certains élèves risquent de la passer en voyant qu'il s'agit d'une preuve. Pour ceux qui y répondront, nous croyons que beaucoup feront référence à la preuve par récurrence.

Pour la deuxième question de cette évaluation, nous demandons d'écrire un algorithme permettant de résoudre un problème. Notre problème est le suivant :

> La factorielle d'un nombre naturel non nul est définie comme suit :
>
> $$n! = 1 \times 2 \times 3 \times ... \times (n-1) \times n.$$
>
> Par exemple, $1! = 1$ et $6! = 1 \times 2 \times 3 \times 4 \times 5 \times 6 = 720$.
> Écrivez un algorithme permettant de calculer la factorielle d'un naturel non nul.

Une solution possible est l'algorithme suivant :

Algorithme

Entrée : un nombre naturel non nul n.
Sortie : $n!$

1. $fact \leftarrow 1$
2. **pour tout** i allant de 1 à n **faire**
3. $\quad fact \leftarrow fact \times i$
4. **fin pour tout**
5. **retourner** $fact$

Même si nous n'avons pas construit beaucoup d'algorithmes en classe, nous souhaitons quand même interroger les élèves sur cet aspect car nous avons vu que l'écriture d'algorithmes posait souvent problème aux étudiants universitaires. Cependant, pour résoudre le problème posé dans cette question, l'algorithme n'est pas si compliqué à trouver puisque son principe est en fait le même que celui de l'algorithme pour calculer la somme des n premiers naturels que nous avons construit pas à pas en classe. Ici, au lieu d'ajouter une valeur lors de chaque passage dans la boucle, il faut multiplier une nouvelle valeur.
Néanmoins, cet algorithme comporte quand même une différence notable par rapport à l'algorithme pour la somme : l'initialisation. En effet, pour

la somme, il fallait commencer à 0 ($somme \leftarrow 0$), tandis qu'ici, il faut commencer à 1 puisqu'il s'agit d'un produit ($fact \leftarrow 1$). Ce sera une façon de voir si les élèves ont bien compris comment initialiser les variables.

Nous avons volontairement choisi d'éliminer la valeur 0 de l'entrée car sinon, il aurait fallu en préciser la factorielle dans l'énoncé. Si nous l'avions mis, peut-être que des élèves auraient voulu traiter séparément le cas $n = 0$ et le cas $n \neq 0$, mais nous ne voulions pas que les élèves utilisent une structure conditionnelle en plus de la structure itérative.
Nous pensons que les élèves vont facilement trouver l'entrée et la sortie car elles sont explicites dans l'énoncé. En ce qui concerne le corps de l'algorithme, nous nous attendons quand même à retrouver des boucles dans les copies et de bons raisonnements. Nous regarderons également si les élèves écrivent les différentes notions correctement (affectation, boucle).

Deuxième partie : avis des élèves

Afin d'obtenir l'avis des élèves sur notre séquence de cours, nous leur posons les trois questions suivantes :

(a) Quels sont les points positifs ?

(b) Quels sont les points négatifs ?

(c) Quels points seraient à améliorer ?

IX.2 Réponses des élèves

IX.2.1 Classe 1

Les 18 élèves de la classe 1 étaient présents lors de l'évaluation.

Première partie : questions sur la matière

Question 1

Au point (a), 15 élèves ont trouvé la réponse exacte. Deux des élèves ayant faux ont exécuté la boucle une fois de trop et ont donc trouvé comme réponse $\frac{1}{32}$ au lieu de $\frac{1}{16}$. Le dernier a, quant à lui, le bon raisonnement, mais il a fait d'énormes fautes mathématiques ($\frac{1}{2}/2 = 1$).
Tous les élèves de cette classe semblent donc savoir exécuter un algorithme

correctement.

Au point (b), 16 élèves ont trouvé la réponse correcte. Les deux qui ont mauvais ont fait des fautes de calculs : $\frac{1}{3}/3 = 1$ et $\frac{\frac{1}{3}}{3} = \frac{1}{24}$.
Plus aucun élève n'a donc exécuté la boucle une fois de trop.

Au point (c), 11 élèves sur les 18 constatent qu'il y a un problème avec la valeur $x = 0$ et signalent donc que l'algorithme ne peut pas être exécuté avec cette valeur. Parfois, ils ne font pas référence à l'entrée, mais ils se sont aperçus qu'il fallait diviser par 0 dans l'algorithme et donc qu'il y avait un problème.
Six élèves qui ont mauvais renvoient la valeur $t = 0$, tandis que le dernier renvoie $t \leftarrow ND$.

Pour le point (d), 11 élèves sont parvenus à trouver la sortie correcte sans le moindre problème. Quatre élèves sont proches de la réponse correcte mais se trompent dans l'exposant. Ils donnent les réponses x^{-y}, x^{-y-1}, $x^{-(y+1)}$ et $\frac{1}{x^{j-1}}$ (cet élève s'est trompé dans le nom de la variable).
Les trois autres élèves donnent les réponses 1 (il s'agit de l'élève qui s'est trompé dans le calcul des fractions), $\frac{1}{x}.\frac{1}{8}$ (l'élève donne juste la réponse du premier point) et une phrase en français (« l'algorithme calcule les résultats pour une famille d'instances et un problème donné »).
Les élèves n'ont donc, en général, pas eu trop de mal à généraliser sur la base des points précédents. Certains auraient quand même dû tester la sortie avec les résultats trouvés aux points (a) et (b) et ils se seraient aperçus des erreurs dans les exposants.

Au point (e), 3 élèves ne répondent rien. Seuls 10 élèves parlent de la correction dans leurs copies, les 5 autres ne parlant que de la terminaison. En réalité, dans leur interrogation, nous avions juste noté qu'il fallait prouver cet algorithme, sans préciser que nous attendions la preuve de la correction. Nous avons bien sûr modifié le document pour les classes 2 et 3.
Chez les 10 élèves qui travaillent sur la correction de l'algorithme, nous trouvons :

- 1 élève qui note juste « preuve par récurrence »,
- 1 élève qui note juste le nom des deux étapes de la preuve,
- 4 élèves qui expliquent le principe général d'une preuve par récurrence, sans faire de référence à l'algorithme,
- 4 élèves qui font référence à l'algorithme.

Parmi les 4 élèves de la dernière catégorie, 3 n'ont pas compris que la récurrence devait s'effectuer sur le nombre de passages dans la boucle et ils ont donc construit leurs preuves sur les valeurs des variables x et y fixées au début de l'algorithme. Rappelons que des étudiants universitaires avaient commis la même erreur.
Nous constatons aussi que les élèves ont été habitués à travailler sur des preuves par récurrence avec la variable n car nous trouvons des n dans leurs copies.

E_1 : Pas initial : si $y = 0$.
Pas de récurrence : on considère que l'algorithme est vrai jusque $t = \frac{1}{x^n}$, puis on montre que $\frac{\frac{1}{x^n}}{x} = \frac{1}{x^{n+1}}$.

E_2 : Nous devons admettre que si $x = n - 1$ et $y = a - 1$, l'algorithme va donner $\frac{1}{(n-1)^{a-1}}$. À partir de là, il faut prouver que pour $x = n$ et $y = a$, l'algorithme sera encore valable.

E_3 : Pas initial : prouver que l'algorithme marche avec $y = 1$.
Pas de récurrence : on considère que cela marche pour $y = n - 1$. On montre ensuite que cela marche par conséquent avec $y = n$.

Enfin, le dernier élève semble avoir compris que la récurrence devait s'effectuer sur le nombre de passages dans la boucle, mais il ne développe pas correctement et entièrement son idée.

E_4 : $\frac{1}{x^0}$ si $j = 1$; $\frac{1}{x}$ si $j = 2$; $\frac{1}{x^2}$ si $j = 3$.
On suppose vrai pour $j = n - 1$, donc pour $j = n$, on utilise ça.

Pour ces quatre élèves, même si leurs réponses ne sont pas correctes, ils semblent au moins avoir compris que, pour une preuve par récurrence, il faut supposer qu'une certaine propriété est vraie pour une valeur et montrer qu'elle reste vraie pour la valeur suivante.

Question 2

Deux élèves n'ont pas répondu à cette question. Chez les 16 restants, tous ont utilisé une boucle « pour tout » et leurs affectations sont notées correctement. Ils ne sont que 2 à avoir oublié d'initialiser les variables intermédiaires. Trois encore ont testé leurs algorithmes sur un exemple.
Parmi les 16 élèves qui répondent, 12 indiquent les entrées et les sorties, mais 4 d'entre eux ajoutent dans les entrées les variables intermédiaires utilisées au sein de l'algorithme.

Trois élèves n'ont absolument rien compris au problème posé et leurs réponses n'ont donc aucun sens (utilisation d'exposants, par exemple). Un autre a écrit le début d'un algorithme pour le calcul de la factorielle de 3.

E_1 : Entrée : $n \in \mathbb{N}_0$ et $x = 3$.
Sortie : $1.2...(n-1).n$
1. $n \leftarrow 3$
2. pour tout i allant de 1 à 3 faire

Un élève a compris qu'il était possible de créer un algorithme récursif pour résoudre ce problème, mais comme nous n'avons pas étudié la récursivité en classe, son algorithme n'a donc pas vraiment de sens. De plus, il n'a pas initialisé sa variable t.

E_2 : 1. pour tout j allant de 1 à n
2. $t \leftarrow (j-1)! \times j$
3. fin pour tout

Un autre élève a constaté qu'il s'agissait en fait de multiplier entre eux les n premiers naturels. Il détermine donc ces n naturels et les multiplie entre eux, sauf que nous n'avons pas vu comment stocker plusieurs valeurs dans une liste, par exemple. Son algorithme n'est donc pas tout à fait correct.

E_3 : 1. $t \leftarrow 0$
2. pour tout j allant de 1 à n
3. $t \leftarrow t + 1$
4. fin pour tout
5. multiplier les t entre eux
6. retourner

Trois élèves commettent exactement la même erreur : dans la boucle, ils utilisent à deux reprises la valeur courante de la variable devant contenir la factorielle, ce qui fait que leurs algorithmes ne sont pas corrects. Un élève a testé le sien et il s'est aperçu qu'il devait y avoir un problème quelque part.

E_4 : 1. $y \leftarrow 1$
2. pour tout k allant de 1 à x faire
3. $y \leftarrow y.(y+1)$

E_5 : 1. $n \leftarrow 1$
2. pour tout i allant de 2 à n faire
3. $n \leftarrow n.(n+1)$
(Cet élève fait en plus varier la valeur de n qui était fixée au départ.)

E_6 : 1. $t \leftarrow 1$
2. pour tout i allant de 1 à $n-1$ faire
3. $t \leftarrow t.(t+1)$

Enfin, chez 4 élèves, les algorithmes sont tout à fait corrects. Chez 3 autres encore, les algorithmes sont presque bons à un détail près :

E_7 : 1. $t \leftarrow 1$
2. pour tout j allant de 1 à n faire
3. $t \leftarrow t.(j+1)$
(Le « +1 » est en trop car alors, la factorielle de 1 vaudrait 2.)

E_8 : 1. $p \leftarrow 1$
2. pour tout k allant de 1 à n faire
3. $p \leftarrow n.p$
(L'élève a mis n au lieu de k dans sa ligne 3.)

E_9 : 1. $t \leftarrow n$
2. pour tout j allant de 1 à n faire
3. $t \leftarrow t.j$
(L'initialisation n'est pas correcte.)

En général, les élèves ont donc quand même eu de bonnes idées pour l'écriture de leurs algorithmes. Souvent, s'ils les avaient testés, ils se seraient sûrement aperçus de certaines de leurs erreurs et ils auraient donc pu obtenir des algorithmes tout à fait corrects.

Deuxième partie : avis des élèves

Quels sont les points positifs ?

Les élèves ont trouvé que la matière était expliquée simplement, clairement, qu'elle apportait de bonnes bases et qu'elle était bien détaillée et intéressante.

Quels sont les points négatifs ?

Plusieurs élèves ont reproché la lenteur de l'introduction qu'ils n'ont pas jugée utile. Certains auraient voulu avoir les feuilles durant les heures de cours afin de ne pas devoir prendre de notes. Deux élèves encore auraient aimé aller un peu plus loin dans la matière.
Enfin, certains n'ont tout simplement pas trouvé de points négatifs.

Quels points seraient à améliorer ?

Des élèves ne trouvent rien à améliorer. Le document a été jugé trop précis et trop long pour deux élèves. Un élève l'aurait voulu plus aéré tandis qu'un autre le trouve très clair.
Un élève aurait souhaité effectuer un exemple de preuve supplémentaire. Enfin, des élèves auraient voulu un rythme un peu plus rapide.

IX.2.2 Classe 2

Les 6 élèves de la classe 2 étaient présents lors de l'évaluation.

Première partie : questions sur la matière

Question 1

Au point (a), seuls 2 élèves ont trouvé la réponse exacte. Un élève a la bonne démarche, mais il a voulu mettre les fractions sous forme décimale et s'est trompé dans le calcul : « $t = \frac{0{,}125}{2} = 0,05125$ ». Les trois autres élèves n'ont pas su exécuter la boucle « pour tout » correctement et retournent tous la valeur 1. Leur exécution de l'algorithme est la suivante :

$E_{1,2,3}$: 1. $t \leftarrow 2$
2. pour tout j allant de 1 à 5 faire
3. $t \leftarrow 2/2$
4. fin pour tout
5. retourner 1

Au point (b), à nouveau, il n'y a que 2 élèves qui ont trouvé la réponse exacte. L'élève ayant fait une faute de calcul au point (a) en a encore commis une : « $t = \frac{0{,}11}{3} = 0,000003$ ». Les trois autres n'ont, une fois de plus, pas su exécuter la boucle « pour tout ».
Nous pensions que ces exécutions ne poseraient pas de problèmes car nous avions travaillé sur un algorithme similaire en classe et les élèves avaient su l'exécuter après quelques explications.

Au point (c), les 2 élèves ayant trouvé les bonnes réponses aux points précédents se rendent compte qu'il y a un problème avec la valeur $x = 0$ et notent que l'algorithme ne retourne rien. Un élève dit que l'algorithme retourne 0/0 et les trois autres notent « $t \leftarrow 0/0$ » et ensuite « retourner 0 ».

Dans cette classe, aucun élève n'a donc fait explicitement référence à l'entrée.

Pour le point (d), deux élèves trouvent la sortie correcte : il s'agit d'un ayant bon en (a) et en (b) et de celui ayant commis les fautes de calcul. L'autre élève qui a bon aux deux premiers points note que l'algorithme retourne $\sqrt{x}$. Il n'a donc pas su généraliser sur la base des exemples des points précédents.
Enfin, les trois élèves n'étant pas parvenus à exécuter la boucle donnent, sans réelle surprise, le résultat $\frac{x}{x}$.

Au point (e), trois élèves ne répondent rien pour la correction : un tente de prouver la terminaison, un note qu'il n'a pas le temps et le dernier ne sait plus comment procéder.
Les réponses des trois autres élèves sont les suivantes :

E_1 : Preuve par récurrence.

E_2 : Démonstration par récurrence.

E_3 : Comparer les algorithmes avec des entrées différentes.

Aucun élève ne développe donc sa réponse et ne l'adapte à l'exemple, comme il l'était explicitement demandé dans la question. Rappelons quand même que les élèves de la classe 2 n'avaient pas beaucoup travaillé la preuve par récurrence avec leur enseignante (enseignante 2).

Question 2

Un élève n'a rien répondu à cette question. Chez les 5 restants, 3 ont utilisé une boucle « pour tout », 1 a utilisé une boucle « tant que » et le dernier n'a noté sur sa feuille que l'entrée et la sortie de l'algorithme. De plus, leurs affectations sont notées correctement et 3 élèves ont pensé aux initialisations.
Sur les 5 élèves qui répondent quelque chose, seuls 2 indiquent une entrée et une sortie. Cependant, ils ajoutent tous deux des variables intermédiaires dans l'entrée et leurs sorties ne sont pas correctes : « n termes $\times(n-1)\times n$ » et « un naturel non nul ».

Un élève n'a pas su résoudre le problème, mais il a constaté que le problème ressemblait fortement à celui pour la somme des n premiers naturels.

E_1 : 1. pour tout k allant de 1 à n faire
2. *somme* $\leftarrow 1.2.3...(n-1).n$
3. fin pour tout
4. retourner *somme*

Un élève écrit un algorithme correct avec une boucle « tant que », mais il ne précise pas l'entrée et la sortie. Les deux derniers se trompent dans les initialisations (le dernier a également une erreur de variables pour l'opération dans la boucle).

E_2 : Entrée : 2 naturels non nuls x et y.
Sortie : un naturel non nul.
1. $v \leftarrow x$
2. pour tout k allant de 1 à y faire
3. $v \leftarrow v \times k$

E_3 : 1. $x \leftarrow 1$
2. $y \leftarrow n$
3. pour tout x allant de 1 à n faire
4. $y \leftarrow x.n$

Dans cette classe, les élèves ont donc eu assez bien de mal à écrire un algorithme pour résoudre le problème proposé. Souvent, ils oublient l'entrée et la sortie, ce qui fait qu'il est parfois compliqué de comprendre où ils veulent en venir avec leurs algorithmes.
Trois élèves sur les six ont quand même une bonne base, mais ils auraient dû tester leurs algorithmes pour s'assurer qu'ils étaient correctement écrits. En effet, aucun élève de cette classe n'a vérifié son algorithme sur un exemple.

Deuxième partie : avis des élèves

Quels sont les points positifs ?

Les élèves sont d'avis de dire que les explications étaient bonnes et que les exemples de l'introduction étaient bien choisis, comme il s'agissait d'exemples de la vie courante. Ils disent avoir appris des choses.

Quels sont les points négatifs ?

Deux élèves n'ont pas relevé de points négatifs. Les autres ont trouvé que la prise de notes était compliquée le premier jour, que les calculs étaient trop

longs et qu'il aurait fallu plus d'heures afin de travailler sur plus d'exercices.

Quels points seraient à améliorer ?

Quatre élèves n'ont pas d'améliorations à suggérer. Les deux autres auraient aimé voir plus d'algorithmes et avoir des devoirs pour s'exercer chez eux.

IX.2.3 Classe 3

Sur les 14 élèves de la classe 3, seuls 9 étaient présents lors de l'évaluation.

Première partie : questions sur la matière

Question 1

Au point (a), seulement 3 élèves ont trouvé la réponse correcte. Trois élèves commettent la même erreur que celle rencontrée dans la classe 2 consistant à ne pas exécuter la boucle et ils trouvent donc 1 comme réponse. Un élève recopie un bout de l'algorithme et ne retourne rien. Les deux derniers ont compris l'exécution d'une telle boucle, mais l'un divise par j plutôt que par x dans le corps de la boucle tandis que l'autre note l'égalité $\frac{1}{2} = 0$ à un moment donné et retourne donc 0.

Au point (b), les résultats sont tout à fait similaires puisque les 3 mêmes élèves ont trouvé la bonne réponse et les 6 autres commettent les mêmes erreurs qu'au point (a).
À nouveau, nous constatons que l'exécution de cet algorithme a posé plus de problèmes que nous le pensions.

Au point (c), 2 élèves s'aperçoivent qu'il y a un problème avec la valeur $x = 0$ et notent que l'algorithme ne peut pas être exécuté. L'un des deux fait clairement référence aux entrées. D'autres élèves retournent 0 (3 d'entre eux) ou $\frac{0}{0}$ (1 élève). Un élève ne répond rien à la question tandis que les deux élèves restants ne terminent pas leurs raisonnements et ne donnent donc pas de réponse finale.

Pour le point (d), seul un élève est parvenu à trouver la sortie correcte. Un élève a noté la réponse $\frac{1}{(y-1)^x}$, réponse qui fonctionne avec les valeurs des deux premiers points. Par contre, par rapport au point (c), il aurait pu

constater que la valeur $x = 0$ ne pose aucun problème avec sa réponse.
Trois élèves ne répondent rien à ce point et deux autres répondent $\frac{x}{x}$, comme dans la classe 2. Enfin, les deux derniers élèves se contentent d'une phrase en français n'ayant pas beaucoup de sens.

E_1 : Les instances du problème.

E_2 : L'algorithme permet de calculer les étapes pour un problème.

Au point (e), 3 élèves ne parlent pas de la correction : soit ils ne répondent rien, soit ils prouvent la terminaison. Quatre élèves notent juste « récurrence » sur leurs copies. Enfin, les deux derniers parlent de la récurrence et essaient d'en expliquer le principe, mais sans réel succès.

E_1 : Il faut prouver qu'il est possible de trouver une réponse pour $j = 0$. Puis ...

E_2 : La technique de récurrence en utilisant $x - 1$.

Les élèves de la classe 3 ne semblent donc pas avoir compris le principe de la preuve par récurrence et ils sont donc incapables de l'adapter à l'exemple proposé. Le dernier élève dont nous avons parlé semble vouloir effectuer une preuve sur les valeurs et non sur le nombre de passages dans la boucle (j). Nous avons déjà constaté cette erreur à plusieurs reprises chez les étudiants universitaires et dans la classe 1.
Enfin, rappelons que l'enseignante 2 n'avait pas eu l'occasion de travailler énormément la preuve par récurrence avec la classe 3. Les élèves y étaient donc peu habitués.

Question 2

Tous les élèves de cette classe ont répondu à cette question. Parmi ces 9 élèves, 7 ont compris qu'il fallait travailler avec une boucle pour résoudre le problème et 5 pensent à initialiser leurs variables. Cependant, il semble que 3 élèves de cette classe utilisent le symbole « = » pour l'affectation.
Ils ne sont que 3 élèves à indiquer l'entrée et la sortie. Dans leurs entrées, nous ne trouvons pas de variables supplémentaires comme nous en avions trouvé dans les deux autres classes. Cependant, chez un de ces trois élèves, l'entrée est donnée pour un cas particulier ($n = 1$).
En outre, un élève a indiqué l'entrée mais pas la sortie.

Parmi les élèves de cette classe, quatre d'entre eux n'ont absolument rien compris au problème posé et leurs réponses n'ont donc aucun sens.

Un autre élève écrit l'algorithme pour un cas particulier et le teste en l'écrivant. Constatons également que cet élève ne maîtrise pas les deux types de boucles rencontrés.

E_1 : Entrée : $n = 1$
Sortie : $n!$
1. $n \leftarrow 1$
2. tant que $x < 3$ avec j allant de 0 à 3 faire
2. $x \leftarrow n.(n + j) = 1.(1 + 0) = 1$
3. $1 < 3$? Oui.
2. $x \leftarrow n.(n + j) = 1.(1 + 1) = 2$
3. $2 < 3$? Oui.
2. $x \leftarrow 1.(1 + 2) = 3$
3. $3 < 3$? Non.
4. fin tant que
5. retourner $n! = 6$

Un élève ne définit pas de nouvelle variable pour contenir la factorielle et utilise donc une même variable pour contenir deux éléments distincts.

E_2 : 1. pour tout a allant de 1 à n faire
2. $a \leftarrow a.(n - 1)$
3. fin pour tout
4. retourner a

Enfin, un élève a écrit un algorithme correct en n'omettant pas l'entrée et la sortie et il l'a même testé afin de vérifier qu'il n'y avait aucun problème. Chez les deux derniers élèves, l'un exprime mal sa boucle et l'autre oublie d'augmenter la valeur de sa variable dans la boucle.

E_3 : Entrée : un nombre naturel non nul (n).
1. $x \leftarrow 1$
2. tant que z va de 2 à $n \rightarrow 2 \leqslant z \leqslant n$
3. $x \leftarrow x.z$
4. fin tant que
5. retourner x

E_4 : Entrée : n naturel non nul.
Sortie : $n!$
1. $x = 0$
2. $i = 0$
3. tant que $i \leqslant n$ faire

4. $x \leftarrow x.(i+1)$
5. fin tant que
6. retourner x

(En plus de l'opération manquante, les affectations ne sont pas notées correctement au début et la condition de la boucle doit être $i \leqslant n-1$.)

Certains élèves de cette classe ont donc eu de bonnes idées pour l'écriture de leurs algorithmes. Chez deux élèves, la distinction entre les boucles « pour tout » et « tant que » ne semble pas être claire puisqu'ils mélangent les deux. Des élèves ont pensé à vérifier leurs algorithmes sur un exemple, mais ce n'est pas encore une démarche systématique pour tous. Pour au moins un élève, s'il avait testé son algorithme sur un exemple, il aurait vu qu'il y avait un problème.

Deuxième partie : avis des élèves

Quels sont les points positifs ?

Les élèves ont jugé que le cours et les explications étaient clairs. Ils ont trouvé la matière intéressante, amusante, assez différente des matières habituelles et étaient contents d'avoir vu de la matière d'études supérieures. Certains ont apprécié le fait de devoir prendre des notes. Trois élèves ont jugé que le nombre d'exercices était correct et qu'ils étaient variés.
Enfin, un élève a trouvé que tout était positif.

Quels sont les points négatifs ?

Trois élèves n'ont pas trouvé de points négatifs à notre séquence d'enseignement. Certains ont trouvé la matière difficile et d'autres déplorent le manque de temps et auraient aimé voir plus d'exercices. Un élève a jugé que le document n'était pas assez aéré.

Quels points seraient à améliorer ?

Cinq élèves ne trouvent aucun point à améliorer. Les autres auraient voulu plus d'exercices, plus de détails dans les preuves d'algorithmes et un cours écrit pour ne pas devoir prendre de notes.

IX.2.4 Résultats globaux

Nous présentons des tableaux reprenant les résultats globaux pour chaque question dans les trois classes de manière à mettre en évidence ce qui a été le mieux réussi par les élèves, mais également ce qui leur a posé des problèmes.
Pour chaque point évalué, nous donnons le taux de réussite dans chaque classe et la moyenne globale (en %).

Question 1

L'énoncé de cette question se trouve à la page 171. Dans le tableau ci-dessous, la numérotation des colonnes correspond aux aspects suivants :

1. Point a) : exécution d'un algorithme.
2. Point b) : exécution d'un algorithme.
3. Point c) : exécution d'un algorithme avec une entrée incorrecte.
4. Point d) : généralisation.
5. Point e) : présence du terme « récurrence ».
6. Point e) : présence des deux étapes.
7. Point e) : référence à l'algorithme dans la preuve par récurrence.
8. Point e) : récurrence sur le nombre de passages dans la boucle.

	1.	2.	3.	4.	5.	6.	7.	8.
Classe 1	83	89	61	61	56	50	22	6
Classe 2	33	33	33	33	33	0	0	0
Classe 3	33	33	22	11	67	0	0	0
Moyenne globale	61	64	45	42	55	27	12	3

Question 2

L'énoncé de cette question se trouve à la page 173. Dans le tableau ci-dessous, la numérotation des colonnes correspond aux aspects suivants :

1. Utilisation d'une boucle.
2. Affectations et boucle notées correctement.
3. Initialisation des variables intermédiaires.
4. Présence de l'entrée et de la sortie.
5. Raisonnement correct.
6. Algorithme correct.

	1.	2.	3.	4.	5.	6.
Classe 1	89	89	78	67	39	22
Classe 2	67	67	50	33	50	17
Classe 3	78	33	56	44	33	11
Moyenne globale	82	70	67	55	39	18

IX.3 Bilan

Dans ce chapitre, nous avons donc présenté l'évaluation proposée aux élèves lors de la sixième heure de notre expérimentation, ainsi que leurs réponses. Nous avons constaté que l'exécution d'un algorithme n'était pas encore acquise dans les classes 2 et 3. En effet, bon nombre d'élèves ne parviennent pas à exécuter un algorithme comportant une boucle « pour tout », même s'il ressemble fortement à un travaillé en classe.

Les élèves ne semblent pas encore totalement maîtriser les notions d'entrée et de sortie puisque, lorsque nous leur demandons d'exécuter un algorithme pour des valeurs données, peu d'entre eux se soucient de vérifier que ces valeurs appartiennent bien aux entrées.

En ce qui concerne la généralisation sur la base d'exemples pour trouver la sortie d'un algorithme, il est assez difficile de dire si les élèves maîtrisent cette compétence car, comme beaucoup d'élèves se sont trompés dans les exemples, leurs généralisations ne pouvaient pas être correctes. Cependant, certains élèves formulent des généralisations sans les vérifier sur les exemples. Nous avons constaté ce fait dans la classe 1 où les élèves se trompaient dans l'exposant.

Pour la preuve de la correction de l'algorithme proposé, beaucoup d'élèves ont pu dire qu'il fallait utiliser la preuve par récurrence. Cependant, dans la majorité des cas, ils ne développent pas leurs réponses ou ne font pas de référence à l'algorithme. Les seuls élèves qui font explicitement référence à l'algorithme sont des élèves de la classe 1, mais souvent, ils effectuent la preuve sur les valeurs des entrées et non sur le nombre de passages dans la boucle. Nous avions déjà constaté ce problème chez les étudiants universitaires.
Au terme de l'évaluation, la preuve par récurrence de la correction d'un algorithme ne semble donc être acquise par aucun élève des trois classes, même s'ils avaient plus ou moins de facilités durant les heures de cours.

Concernant l'écriture d'un algorithme pour résoudre un problème donné, beaucoup d'élèves oublient de stipuler les entrées et les sorties. De plus, certains ne semblent pas encore maîtriser les différentes structures comme l'affectation (notée « = » dans certaines copies) et les deux boucles.
Certains élèves ont de très bons algorithmes, tandis que d'autres n'ont pas réellement réfléchi au problème et n'y apportent donc aucune solution. Enfin, chez certains élèves, les algorithmes sont presque corrects, à l'un ou l'autre détail près. Nous sommes persuadée que si ces élèves avaient testé leurs algorithmes, ils se seraient probablement aperçus des problèmes. Il s'agit là d'un reproche que nous avions déjà formulé à l'égard des étudiants universitaires.

Finalement, lorsque nous demandons l'avis des élèves sur cette séquence d'enseignement, nous constatons qu'ils sont assez d'accord pour dire qu'ils l'ont trouvée intéressante, même si elle comporte quelques défauts, comme le peu d'heures de cours qui y a été consacré.

Nous présentons l'analyse critique de cette séquence d'enseignement dans la conclusion.

Conclusion

Bilan du travail

L'algorithmique ne fait pas partie des programmes actuels de mathématiques de l'enseignement secondaire belge. Notre objectif, dans le cadre de ce travail, a donc été de concevoir une séquence d'enseignement sur l'algorithmique en vue d'une expérimentation dans l'enseignement secondaire.

Avant toute chose, nous avons étudié différentes raisons pour lesquelles l'enseignement de l'algorithmique dans le secondaire pouvait être légitimé. Tout d'abord, dans un ancien manuel de mathématiques pour le secondaire, nous avons relevé la présence d'algorithmes. Les algorithmes ont donc en réalité disparu de l'enseignement des mathématiques dans le secondaire belge.
En outre, dans les études universitaires de mathématiques, le recours à l'algorithmique semble presque inévitable pour parvenir à résoudre certains problèmes posés par les mathématiques. Nous en avons donc conclu que les mathématiques et l'algorithmique entretenaient des liens particuliers.
Ensuite, nos voisins français ont intégré une part d'algorithmique dans leurs cours de mathématiques au lycée (enseignement secondaire supérieur belge) depuis septembre 2009, preuve qu'il est tout à fait envisageable de travailler cette matière dans un cours de mathématiques au niveau de l'enseignement secondaire.
Enfin, des travaux en didactique des mathématiques existent sur le sujet puisque des didacticiens français se sont intéressés de plus près à cette introduction de l'algorithmique dans les cours de mathématiques du lycée. Il ressort de leurs analyses qu'en plus d'être un outil pour la résolution de certains problèmes, l'algorithme peut aussi être étudié en tant qu'objet des mathématiques, d'où l'intérêt d'introduire de l'algorithmique dans un cours de mathématiques. Cependant, l'algorithme n'est que très rarement travaillé en tant qu'objet dans les différentes ressources mises à la disposition des enseignants français du lycée. Au lycée, les objectifs concernent davantage

l'écriture d'un algorithme en vue d'une implémentation dans un langage de programmation.
Pourtant, nous avons montré qu'étudier l'algorithme en tant qu'objet permet de travailler des compétences mathématiques, compétences qui sont préconisées dans l'enseignement secondaire dans notre pays. Notre but a donc été de concevoir une séquence d'enseignement sur l'algorithmique à intégrer dans un cours de mathématiques et permettant de travailler des compétences transversales et terminales que les élèves doivent maîtriser à la fin de leurs études secondaires.

Sur un plan méthodologique, nous avons choisi d'étudier des cours universitaires pour élaborer notre séquence d'enseignement tout en intégrant les résultats de recherche sur l'algorithmique issus de la situation en France. Cette démarche nous semble également permettre de réfléchir à la transition secondaire-université dans le domaine de l'algorithmique. Ainsi, dans le cours d'algorithmique analysé, nous avons constaté que l'algorithme était bien évidemment travaillé en tant qu'outil pour la résolution de problèmes, mais qu'il l'était également en tant qu'objet des mathématiques. Les deux facettes de l'algorithme étant présentes dans ce cours, nous en avons donc extrait les éléments que nous jugions pertinents à intégrer dans notre séquence d'enseignement en nous focalisant davantage sur les points de la matière qui semblent être problématiques pour des étudiants universitaires et que nous avons relevés au moyen d'évaluations de certains cours.
D'après les évaluations que nous avons analysées, les points qui semblent poser des problèmes à ces étudiants sont l'écriture d'un algorithme pour résoudre un problème, l'assimilation d'un langage de programmation et la preuve par récurrence pour prouver qu'un algorithme itératif donne bien le résultat pour lequel il a été conçu.
Étant donné que notre travail s'inscrit dans le cadre d'un cours de mathématiques, nous n'avons pas jugé nécessaire d'introduire des éléments d'un langage de programmation au sein même de notre séquence d'enseignement.

La seconde partie de notre travail a donc consisté en l'élaboration de la séquence d'enseignement, en son expérimentation et en son évaluation.
Dans notre séquence, nous voulions proposer des activités mettant en jeu l'algorithme en tant qu'objet puisque c'est ainsi qu'il est le plus intéressant d'un point de vue mathématique, mais comme les élèves belges n'ont aucune notion d'algorithmique, nous avons d'abord dû nous attarder sur le côté outil des algorithmes afin de leur en présenter l'utilité et la manière de les comprendre et de les écrire.
Nous avons ensuite veillé à travailler des compétences terminales et trans-

versales telles que la généralisation, l'argumentation, la logique ou encore la preuve par récurrence au sein de notre séquence d'enseignement puisque notre objectif était de travailler des compétences mathématiques grâce à des éléments d'algorithmique.

Nous avons expérimenté notre séquence d'enseignement dans une classe de cinquième secondaire et dans deux classes de sixième, toutes ayant plus de six heures de mathématiques par semaine. Notre choix de classes s'est quelque peu imposé à nous puisque nous avions besoin de classes avec beaucoup d'heures de mathématiques pour ne pas trop pénaliser les élèves en ce qui concerne la matière au programme puisque, rappelons-le, l'algorithmique n'en fait pas partie.
La majorité des élèves de ces classes a réellement apprécié cette séquence car bon nombre d'entre eux se destinent à des études contenant un cours d'algorithmique et/ou de programmation. En plus, ils étaient ravis d'étudier une matière assez différente de celles habituellement étudiées dans les cours de mathématiques.
En ce qui concerne la matière étudiée, comme notre nombre d'heures était limité (6 heures pour l'expérimentation et l'évaluation dans chaque classe), nous n'avons pas pu nous attarder sur tout ce que nous avions préparé. Au final, nous avons plutôt travaillé l'algorithme en tant qu'outil durant ces heures car, comme relaté plus haut, nous avons d'abord dû introduire les différentes notions pour comprendre un algorithme afin de pouvoir ensuite étudier l'algorithme en lui-même.

Concernant les points que nous avons travaillés avec les élèves et que nous avons évalués, il s'est avéré que les élèves du secondaire ont éprouvé les mêmes difficultés que celles rencontrées par les étudiants universitaires. C'est un élément qui plaide en faveur d'une réflexion pour favoriser la transition secondaire-université dans ce domaine.
L'écriture d'un algorithme pour résoudre un problème a également posé des problèmes chez les élèves du secondaire. Lors des expérimentations, ils sont parvenus, en s'entraidant et sur la base des questions que nous leur avons posées, à construire un algorithme résolvant un problème donné. Cependant, lors des évaluations, certains élèves n'ont pas trouvé une démarche correcte pour apporter une solution au problème. D'autres encore ont une bonne démarche, mais ils ne vérifient pas leurs algorithmes sur des exemples et ne se rendent donc pas compte que ces derniers sont presque corrects, souvent à un détail près.
La preuve par récurrence a fortement posé problème chez les étudiants universitaires et, sans réelle surprise, elle est très mal passée lors de deux

des trois expérimentations car les élèves en connaissaient à peine le principe. Lors de l'évaluation, les résultats n'ont guère été plus brillants puisque le peu d'élèves qui a tenté une démonstration par récurrence ne l'a pas effectuée sur les bonnes variables. Ils ont donc semblé avoir eu du mal à trouver ce qu'il fallait prouver.

Contrairement aux étudiants universitaires, de nombreux élèves du secondaire ont éprouvé des difficultés à exécuter un algorithme lors de l'évaluation, alors que durant les expérimentations, ils avaient l'air d'y parvenir. Ce résultat est toutefois à nuancer puisque, dans une classe, tous les élèves sont parvenus à exécuter l'algorithme de l'évaluation, avec éventuellement des erreurs dans les calculs.
En ce qui concerne la généralisation sur la base d'exemples, les élèves n'ont en général eu aucun problème à généraliser la sortie d'un algorithme sur la base d'une ou de deux exécutions.

Même si les résultats individuels lors de l'évaluation ne sont pas brillants pour certains élèves [1], nous avons constaté, durant les expérimentations, que les compétences liées à l'algorithmique étaient en bonne voie d'acquisition. Ainsi, un point fort du travail est que l'introduction d'une part d'algorithmique dans un cours de mathématiques semble donc être tout à fait envisageable et accessible pour des élèves de l'enseignement secondaire.

Analyse critique

Il est maintenant intéressant de procéder à une analyse critique de notre séquence de cours afin d'en dégager les points positifs et les points qui seraient à améliorer. Commençons donc par les points positifs. Tout d'abord, nous pensons que notre introduction comprenant des exemples d'algorithmes de la vie quotidienne et des exemples issus des mathématiques connus des élèves constitue un réel point positif. En effet, en travaillant sur des exemples connus des élèves, nous leur montrons que notre but est de leur présenter une nouvelle matière, mais qu'en fait, ils ont déjà travaillé certains éléments de celle-ci. Les étudiants universitaires ont justement relaté qu'ils avaient eu des difficultés au début avec leur cours d'algorithmique parce que la matière était totalement nouvelle, sans aucun lien avec ce qu'ils avaient étudié durant leurs études secondaires.

1. Il faut dire aussi qu'il n'est pas évident d'engager les élèves dans une évaluation quand on n'est pas le titulaire du cours. Il n'y a pas de véritable enjeu pour les élèves.

De plus, lors des expérimentations, les élèves ont apprécié étudier des exemples autres que mathématiques au début afin de se familiariser avec la notion d'« algorithme ».

Ensuite, un autre point positif de cette séquence de cours est que même si elle introduisait l'algorithmique qui est une matière située à la jonction entre l'informatique et les mathématiques, elle restait malgré tout assez fortement connectée à un cours de mathématiques. Effectivement, elle était liée aux mathématiques, comme nous l'avons déjà souligné, avec le fait que notre objectif était de travailler l'algorithme en tant qu'objet des mathématiques, dans le but de travailler des compétences transversales et terminales. De plus, nous avons choisi expressément des exemples mathématiques dans les parties suivant l'introduction de sorte que cette séquence s'inscrive effectivement dans un cours de mathématiques.

En outre, nous trouvons que le document reprenant l'entièreté de la séquence d'enseignement que nous avons rédigé pour les élèves était assez structuré car les différentes parties étaient clairement séparées et nous avons établi des liens pour passer d'une partie à une autre. Il était aussi suffisamment détaillé de sorte que les élèves puissent éventuellement aller relire les passages relatifs à une partie de la matière qu'ils n'auraient pas comprise. De même, pour les parties de la matière que nous n'avons pas eu le temps d'aborder en classe, nous pensons que les élèves seraient en mesure de les comprendre par eux-mêmes avec nos explications.

Enfin, puisque nous avons abordé le sujet du document destiné aux élèves, nous pouvons discuter de la façon dont nous avons procédé durant les différentes expérimentations. Le but n'était donc pas de donner directement le document aux élèves et de le leur faire lire, mais de leur donner cours un peu comme à l'université. Ils devaient donc prendre des notes. La prise de notes est un grand point positif de nos expérimentations car les élèves des différentes classes n'étaient absolument pas habitués à devoir prendre des notes « au vol » durant les cours de mathématiques. Cela a donc permis de les y habituer pour leurs futures études.
Toujours en ce qui concerne le déroulement des cours, notre but était de faire participer le plus possible les élèves. En procédant de cette manière, ils étaient assez actifs en classe et cela leur donnait l'impression de construire le cours par eux-mêmes. La participation active des élèves leur a également permis de poser toutes les questions qui leur passaient par la tête pour ainsi ne laisser aucune zone d'ombre.

Néanmoins, lors de nos expérimentations et surtout lors des évaluations, nous avons pu constater que notre séquence d'enseignement comportait des manques. Le premier que nous avons ressenti concerne les exécutions d'algorithmes. En effet, lors des évaluations, nous avons constaté qu'un nombre non négligeable d'élèves n'est pas parvenu à exécuter l'algorithme proposé alors qu'il ressemblait assez fortement à un des algorithmes que nous avions travaillé durant les heures de cours. Nous pensons donc que nous aurions dû intégrer davantage d'exécutions d'algorithmes dans notre séquence d'enseignement afin que les élèves puissent s'entraîner. Certains d'entre eux nous ont demandé des devoirs et nous aurions peut-être dû leur en donner afin qu'ils travaillent individuellement cette compétence. Effectivement, en classe, ils étaient assez souvent guidés par leurs voisins et nous ne nous sommes donc pas aperçue de l'ensemble des difficultés éprouvées par chaque élève.

Un autre manque est que nous aurions dû proposer aux élèves des algorithmes comportant à la fois une structure conditionnelle et une structure itérative. En effet, tous les algorithmes que nous avons étudiés dans cette séquence de cours comportaient l'une ou l'autre de ces structures, mais jamais elles n'étaient mélangées. Même si certains élèves ont déjà eu du mal avec des algorithmes ne comportant qu'une seule structure, il aurait été intéressant d'en intégrer avec plusieurs structures car après, dans des études supérieures, il est assez rare de ne travailler que sur des algorithmes ne comportant qu'une seule structure. De même, cela aurait intéressé plusieurs élèves qui auraient aimé étudier des algorithmes plus complexes.

Dans notre séquence de cours, nous avons constaté qu'il manquait une seconde preuve de la correction d'un algorithme requérant l'utilisation de la preuve par récurrence. Certains élèves ont signalé qu'ils auraient aimé travailler un deuxième exemple de preuve par récurrence. Nous pensons que l'ajout d'une autre preuve aurait été bénéfique pour deux raisons.
La première raison est que la preuve par récurrence d'un algorithme est assez différente des preuves que les élèves auraient éventuellement pu rencontrer dans des chapitres comme les suites en cinquième secondaire. En effet, la preuve d'un algorithme s'effectue par une récurrence sur le nombre de passages dans la boucle et non sur les valeurs initiales. Nous pensons donc qu'avec un deuxième exemple d'une telle preuve, les élèves pourraient plus facilement comprendre que la preuve s'effectue de la sorte. Nous avons retrouvé cette situation dans une classe puisque les élèves avaient déjà effectué de nombreuses preuves par récurrence et en connaissaient donc le principe, mais lors de l'évaluation, ils n'ont en général pas su dire que la

preuve devait s'effectuer sur le nombre de passages dans la boucle.
La deuxième raison est que si les élèves n'ont jamais réellement travaillé de preuves par récurrence, un seul exemple n'est bien entendu pas suffisant pour leur faire comprendre le principe d'une telle preuve. Un deuxième exemple ne serait alors pas de trop. Nous avons retrouvé cette situation dans les deux autres classes puisque les élèves de ces classes ne connaissaient pas vraiment le principe d'une telle preuve et lors de l'évaluation, aucun élève de ces classes n'a développé sa réponse.

Enfin, un dernier manque dans notre séquence d'enseignement concerne l'écriture d'un algorithme pour résoudre un problème donné. En effet, comme notre but lors de la conception de cette séquence était de ne pas travailler l'algorithme uniquement en tant qu'outil pour la résolution de problèmes, nous avons choisi de ne pas consacrer énormément de temps à l'écriture d'algorithmes : nous avons préféré plutôt les étudier que de les construire. Ceci dit, écrire plus d'algorithmes aurait peut-être intéressé certains élèves car dans des cours de programmation et/ou d'algorithmique en haute école ou à l'université, l'écriture d'algorithmes est une compétence primordiale à maîtriser. À nouveau, nous aurions peut-être dû donner des devoirs aux élèves afin qu'ils s'entraînent à cette compétence.

Avec l'introduction de tous ces manques dans notre séquence de cours, nous pensons que les élèves pourraient obtenir de meilleurs résultats lors de l'évaluation car ils auraient alors eu l'occasion de travailler davantage les différents points qui ont posé des problèmes lors des évaluations menées dans le cadre de ce travail. Cependant, ces quelques ajouts nécessiteraient davantage de travail à domicile de la part de ces élèves et davantage d'heures de cours, mais nous pensons que ce travail pourrait être bénéfique. Beaucoup d'étudiants universitaires avaient d'ailleurs insisté sur le fait que leur cours d'algorithmique nécessitait un travail régulier.

Une limite méthodologique majeure de cette séquence d'enseignement concerne les classes dans lesquelles nous l'avons expérimentée. Effectivement, nous ne l'avons testée que dans trois classes, dont deux avaient la même enseignante pour le cours de mathématiques. Nous n'avons donc pas eu un public varié dans ce sens. De même, le public de ces trois classes était assez similaire puisque tous ces élèves avaient plus de six heures de mathématiques par semaine. Afin d'obtenir des conclusions plus générales sur l'introduction d'une part d'algorithmique dans l'enseignement secondaire belge, il faudrait tester notre séquence d'enseignement dans d'autres classes de niveaux différents. Cependant, pour la tester dans des classes avec moins d'heures

de mathématiques par semaine, le contenu des expérimentations devrait être adapté. En effet, par exemple, la preuve par récurrence n'est pas une compétence terminale exigée chez des élèves avec moins de six heures de mathématiques par semaine.

Enfin, il aurait été intéressant de suivre les élèves se destinant à des études dans lesquelles un cours d'algorithmique ou de programmation est prévu afin de voir s'il est possible que notre séquence d'enseignement puisse les aider vis-à-vis de ces cours. En effet, comme relaté par les étudiants universitaires, l'algorithmique et la programmation sont des matières totalement nouvelles lors de l'entrée à l'université et nous nous demandons donc si notre séquence d'enseignement pourrait permettre à ces élèves de partir confiants avec ces matières parce qu'ils les ont déjà abordées.
Cependant, ces recherches n'auraient pas été possibles dans le cadre de ce travail.

Perspectives

Bien que l'algorithmique soit partie intégrante des cours de mathématiques du lycée en France depuis 2009 et de certains cursus universitaires belges, elle ne se retrouve certes pas dans les programmes de mathématiques de l'enseignement secondaire supérieur belge actuels et elle ne fera pas non plus son apparition dans les nouveaux programmes de mathématiques devant entrer prochainement en vigueur.

Cependant, au travers de ce travail, nous avons constaté qu'il était tout à fait envisageable de travailler des compétences des mathématiques à l'aide de l'algorithmique. Si, à l'avenir, nous disposons de classes ayant des heures de préparation aux études supérieures, nous pourrons leur proposer cette introduction à l'algorithmique afin de montrer aux élèves qu'il est possible de travailler des compétences mathématiques avec des matières moins mathématiques et plus ludiques.
Même en dehors de ce travail spécifique sur l'algorithmique, ces heures supplémentaires devraient donc être l'occasion de travailler des mathématiques autrement et non simplement servir à approfondir la matière du cours de mathématiques six heures par semaine. Nous nous souvenons trop bien de ces heures supplémentaires servant à calculer des intégrales dont la résolution prenait une page recto-verso pour chacune et que nous n'avons plus jamais rencontrées à l'université... Ces heures de préparation aux études supérieures

devraient donc permettre aux élèves d'étudier des matières non prévues par les programmes dans le but de les préparer, comme leur nom l'indique, aux études supérieures, tout en restant connectées à un cours de mathématiques. Notre travail constitue donc un exemple de ce genre.

Bibliographie

Références didactiques

[1] Aldon G., Germoni J., Meny J.-M. (2012), Complexité d'un algorithme : une question cruciale et abordable, *Repères-Irem*, 86, 27-50.

[2] Irem de Grenoble (2012), *Document d'accompagnement des stages de formation à l'algorithmique*, http://www-irem.ujf-grenoble.fr/spip/spip.php?rubrique15.

[3] Kahane J.-P. (2000), *Commission de réflexion sur l'enseignement des mathématiques, informatique et enseignement des mathématiques*, http://smf4.emath.fr/Enseignement/CommissionKahane/.

[4] Lovász L. (1988), *Algorithmic mathematics : an old aspect with a new emphasis*, http://www.mathunion.org/fileadmin/ICMI/files/Digital_Library/ICMEs/ICME_06_1988_Budapest.pdf (p.67-78).

[5] Modeste S., Gravier S., Ouvrier-Buffet C. (2010), Algorithmique et apprentissage de la preuve, *Repères-Irem*, 79, 51-72.

[6] Modeste S. (2012), *Enseigner l'algorithme pour quoi ? Quelles nouvelles questions pour les mathématiques ? Quels apports pour l'apprentissage de la preuve ?*, Thèse de Doctorat de l'Université de Grenoble.

[7] Nguyen C.T. (2005), *Étude didactique de l'introduction d'éléments d'algorithmique et de programmation dans l'enseignement mathématique secondaire à l'aide de la calculatrice*, Thèse de Doctorat en co-tutelle Grenoble et Viêt-nam.

Ressources pédagogiques

[8] Adam A., Bastin R., Close P., Lousberg F. (2002), *Espace Math 4*, Éd. De Boeck.

[9] Adam A., Goossens F., Lousberg F. (1993), *Mathématisons 57*, Éd. De Boeck.

[10] Antoine P., Descy J., Goffin M., Van Hooste C. (2004), *Actimath 4*, Éd. Van In.

[11] BO n° 30 du 23 juillet 2009, http://media.education.gouv.fr/file/30/52/3/programme_mathematiques_seconde_65523.pdf.

[12] BO spécial n° 9 du 30 septembre 2010, http://media.education.gouv.fr/file/special_9/20/9/mathsES+L_155209.pdf et http://media.education.gouv.fr/file/special_9/21/1/mathsS_155211.pdf.

[13] BO spécial n° 8 du 13 octobre 2011, http://media.education.gouv.fr/file/special_8_men/98/2/mathematiques_ES_L_195982.pdf, http://media.education.gouv.fr/file/special_8_men/98/4/mathematiques_S_195984.pdf et http://www.education.gouv.fr/pid25535/bulletin_officiel.html?cid_bo=57572.

[14] Choquer-Raoult A., Cocault M., Hanouch B., Joffrédo T. (2010), *Maths Repères Seconde*, Hachette Éducation.

[15] Ministère de la communauté française (1999), *Compétences terminales et savoir requis en mathématiques*, http://www.enseignement.be/download.php?do_id=503&do_check=.

[16] Ministère de l'éducation nationale (2009), Ressources pour la classe de seconde - Algorithmique -, http://media.eduscol.education.fr/file/Programmes/17/8/Doc_ress_algo_v25_109178.pdf.

[17] Ministère de l'éducation nationale (2014), *L'enseignement des mathématiques : rapport sur la mise en œuvre du programme de mathématiques en classe de seconde*, http://media.eduscol.education.fr/file/Mathematiques/01/6/CSM-projet-rapport2013_293016.pdf.

Cours universitaires

[18] Füzfa A. (2011-2012), *Introduction au calcul scientifique et aux algorithmes mathématiques*, Facultés Universitaires Notre-Dame de la Paix.

[19] Libert A.-S. (2009-2010), *Initiation à la démarche mathématique*, Facultés Universitaires Notre-Dame de la Paix.

[20] Mélot H. (2013), *Programmation et algorithmique I*, Université de Mons.

[21] Sartenaer A. (2011-2012), *Analyse numérique*, Facultés Universitaires Notre-Dame de la Paix.

[22] Strodiot J.-J. (2009-2010), *Calcul différentiel et intégral*, Facultés Universitaires Notre-Dame de la Paix.

[23] Strodiot J.-J. (2011-2012), *Introduction to optimization*, Facultés Universitaires Notre-Dame de la Paix.

[24] Vanhoof W. (2009-2010), *Programmation*, Facultés Universitaires Notre-Dame de la Paix.

Références Internet

[25] Bell T., Witten I.H., Fellows M. (2009), *Computer Science Unplugged, L'informatique sans ordinateur*, http://csunplugged.org/sites/default/files/books/CS_Unplugged-fr.pdf.

[26] Papadimitriou C., Vazirani U. (2006), *Stable Marriage - Application of Proof Techniques for Algorithmic Analysis*, http://www.cs.berkeley.edu/~vazirani/f06cs70.html.

[27] Olympiade belge d'Informatique, http://www.be-oi.be/fr/.

Annexe A

IREM de Grenoble : la dichotomie

Cette page et les suivantes sont issues du document d'accompagnement des stages de formation à l'algorithmique produit par l'IREM de Grenoble.

7 Recherche dichotomique

7.1 Je cherche un nombre entre 1 et 1000

Prérequis algo : Instructions conditionnelles et boucles conditionnelles.
Prérequis math : Mathématiques du collège. Récurrence pour la preuve de la dernière question.

Ce jeu se joue à deux joueurs A et B. Le joueur A choisit secrètement un nombre cible compris strictement entre 1 et 1000. Le joueur B doit deviner ce nombre en faisant le minimum de propositions. À chaque proposition du joueur B, le joueur A répond par « le nombre cherché est plus grand », « le nombre cherché est plus petit » ou « bravo, vous avez gagné » selon la position de la proposition par rapport à la cible à atteindre.

Exercice 1 :

Le but de cet exercice est de jouer contre l'ordinateur.

Question 1 – Proposer un algorithme pour que l'ordinateur tienne le rôle du joueur A.

Question 2 – Programmer et tester cet algorithme en langage Python. On utilisera les instructions suivantes :

```
from random import *  // pour importer la librairie
randrange(1000)       // pour tirer un nombre entre 0 et 999
```

Question 3 – Imaginer une stratégie « optimum » qui permette au joueur B de toujours gagner avec un maximum de 10 propositions. Tester cette stratégie avec le programme de la Question 2.

Question 4 – Proposer un algorithme pour que l'ordinateur tienne maintenant le rôle du joueur B et applique la stratégie « optimum » précédente. A chaque proposition de l'ordinateur vous répondrez par « + », « - » ou « bravo » pour « le nombre cherché est plus grand », « le nombre cherché est plus petit » ou « c'est gagné ».

Question 5 – Écrire et tester cet algorithme en langage Python.

Question 6 – Quels sont les nombres à choisir pour que l'ordinateur trouve la solution en au moins 9 coups ?

Question 7 – Montrer que l'algorithme « optimum » permet de toujours trouver l'entier en au plus 10 essais.

Indication – On peut utiliser un raisonnement par récurrence sur K : si $B - A \leq 2^K$, alors on peut trouver le nombre compris strictement entre A et B en au plus K essais.

Remarque – Dans une situation pour la classe, avec les mêmes exercices sans les questions 6 et 7, on essayera de faire émerger chez les élèves la méthode de recherche par dichotomie.

7.2 Recherche de la solution d'une équation $f(x) = 0$ par dichotomie

Prérequis math : fonction, fonction croissante, décroissante.
Prérequis algo : tantque, instruction conditionnelle.
Objectif math : solution d'une équation $f(x) = 0$.
Objectif algo : mettre en place un algorithme itératif de calcul.

Cet exercice, souvent présenté comme l'un des plus naturel pour illustrer la liaison entre mathématique et algorithmique, se révèle en fait assez complexe.

La principale difficulté est que les calculs sont effectués de façon approchée. Un test $f(x) = 0$ pose problème car la quantité $f(x)$ n'est évaluée que de façon approchée.

Dans certains cas, calcul exact, cette question peut ne pas se poser, mais dans le cas général il nous faudra réfléchir au préalable à cette question.

Hypothèse faite dans tout le paragraphe : Nous ferons l'hypothèse que la fonction f est définie, croissante sur l'intervalle $[a, b]$ et que l'équation $f(x) = 0$ possède une solution unique sur $[a, b]$. La propriété de continuité n'est pas nécessaire !

Calcul exact

Partons tout d'abord avec l'hypothèse que le calcul de $f(x)$ se fait de façon exacte et examinons l'algorithme suivant :

```
Entrer a
Entrer b
Entrer precision
tantque (b-a) > precision
  c ← (a+b)/2
  si f(c) > 0
    b ← c
  sinon
    a ← c
Écrire c
```

Argumentation : considérons le point $c = \frac{a+b}{2}$ et regardons ce que nous pouvons déduire du test :
- si $f(c) > 0$, comme la fonction est croissante, la solution unique cherchée est dans l'intervalle $[a, c]$ (même $[a, c[$).
- dans le cas contraire ($f(c) \leq 0$) la solution cherchée est dans l'intervalle $[c, b]$.

Conclusion : en chaque début d'itération on peut affirmer que la solution appartient à l'intervalle $[a, b]$, l'amplitude de l'intervalle étant divisé par 2 à chaque étape, il est assez facile de prouver que l'algorithme s'arrête.

Remarque 1 : on peut également montrer que si la solution appartient à l'intervalle $[a, b[$ alors cette condition est maintenue tout au long de l'algorithme. En effet :
- si $f(c) > 0$, comme la fonction est croissante, la solution unique cherchée est dans l'intervalle $[a, c[$.
- dans le cas contraire ($f(c) \leq 0$) la solution unique est dans l'intervalle $[c, b[$.

Remarque 2 : Considérons l'exemple suivant. Soit n un entier $0 \leq n \leq 1000$ et considérons la fonction f définie sur $[0, 1001[$ par :

$$\begin{cases} f(x) = -1, \ x \in \ [0, n[\\ f(n) = 0 \\ f(x) = 1, \ x \in \ ]n, 1001[\end{cases}$$

On remarquera que l'algorithme de dichotomie résout le problème de la recherche du nombre n à condition que la longueur de l'intervalle résultant soit inférieur à 1, ce qui est réalisé après 10 itérations.

Calcul exact et arrêt éventuel de l'algorithme sur la valeur exacte

Nous supposons toujours que le calcul se fait de façon exacte et nous voulons intégrer dans l'algorithme un arrêt éventuel quand la valeur exacte est trouvée.

Nous partons de l'hypothèse que la solution recherchée est dans l'intervalle $]a, b[$, ce qui signifie que l'on a testé auparavant les valeurs extrêmes a et b.

L'adaptation de l'algorithme proposée est la suivante :

```
Entrer a
Entrer b
Entrer precision
trouve = 0
tantque ((b-a) > precision) et (non trouve)
  c ← (a+b)/2
  si f(c) = 0
    trouve = 1
  sinon
    si f(c) > 0
      b ← c
    sinon
      a ← c
si trouve = 1
  Écrire "solution exacte =",c
sinon
  Écrire "solution approchée dans", [a,b[
```

Calcul approché

Nous abordons le cas de calcul approché, comme le pratiquent les ordinateurs en arithmétique flottante. Nous ferons les hypothèses raisonnables suivantes pour l'évaluation approchée d'une quantité ξ :
- si la valeur exacte $\xi < 0$, alors sa valeur approchée est strictement négative.
- si la valeur exacte $\xi = 0$, alors sa valeur approchée est soit nulle, soit strictement positive, soit strictement négative, en fait on ne peut rien conclure,
- si la valeur exacte $\xi > 0$, alors sa valeur approchée est strictement positive.

Avec ces hypothèses, quelle signification peut-on alors donner au test $f(c) > 0$ sachant que le calcul de $f(c)$ a été fait de façon approchée :
- soit la valeur exacte de $f(c)$ est effectivement strictement positive,
- soit, du fait de l'approximation, on a pour la valeur exacte $f(c) = 0$.

Avec ces mêmes hypothèses, dans le cas contraire où la valeur approchée de $f(c)$ vérifie le test $f(c) \leq 0$ alors :
- soit la valeur exacte de $f(c)$ est effectivement strictement négative,
- soit, du fait de l'approximation, on a pour la valeur exacte $f(c) = 0$.

Avec une fonction f définie, continue, croissante sur l'intervalle $[a, b]$ et une solution unique de l'équation $f(x) = 0$ sur $[a, b]$ considérons le point $c = \frac{a+b}{2}$ et regardons ce que nous pouvons déduire du test $f(c) > 0$.
- Si la quantité exacte $f(c)$ est effectivement strictement positive, alors la solution unique cherchée est dans l'intervalle $[a, c]$ (même $[a, c[$).
- si du fait de l'approximation la valeur exacte de $f(c)$ est effectivement nulle, alors la solution unique cherchée est également dans l'intervalle $[a, c]$.

Dans le cas contraire où le test vérifie $f(c) \leq 0$.
- Si la quantité exacte $f(c)$ est effectivement strictement négative, alors la solution unique cherchée est dans l'intervalle $[c, b]$ (même $]c, b]$),
- si du fait de l'approximation la valeur exacte de $f(c)$ est effectivement nulle, alors la solution unique cherchée est également dans l'intervalle $[c, b]$.

Dans les deux cas on peut donc remplacer l'intervalle $[a, b]$ soit par l'intervalle $[a, c]$ soit par l'intervalle $[c, b]$ et l'algorithme déjà présenté :

```
Entrer a
Entrer b
Entrer precision
tantque (b-a) > precision
   c ← (a+b)/2
   si f(c) > 0
      b ← c
   sinon
      a ← c
Écrire c
```

se trouve "robuste aux calculs approchés", mais, *attention : des problèmes de précision peuvent apparaître si l'on demande un encadrement trop précis.*

Annexe B

Questionnaires proposés aux étudiants

B.1 Test de Mathématiques pour l'Informatique I (décembre 2013)

L'image ci-dessous présente une question du test qui permet de travailler, entre autres, la preuve d'un algorithme.

Question 4. On considère l'algorithme (en pseudo-code) ci-dessous.

Entrées : $a \in \mathbb{N}$, $b \in \mathbb{N}_0$
Initialisation : $c := 0$, $d := a$

Tant que $(d \geq b)$ **faire**
  $d := d - b$
  $c := c + 1$
  Si $(d = 0)$ **alors** $b := b^2$
fin faire

Sortie : d

(a) Que retourne l'algorithme avec les entrées $a = 131$ et $b = 4$?

(b) Dans le cas général, que calcule l'algorithme ?

(c) Prouvez (par induction sur le nombre de passage dans la boucle) que chaque fois que l'algorithme entre dans la boucle, l'égalité suivante est vérifiée : $a = b.c + d$.

(d) Utilisez le point (c) pour prouver ce que vous avez affirmé au point (b).

B.2 Examen de Programmation et Algorithmique I (janvier 2014)

Les quatre images suivantes représentent l'ensemble de l'examen relatif à ce cours qu'ont passé les étudiants de première année en sciences informatiques et mathématiques en janvier 2014.

Problème 1 :

Qu'est-ce qui sera affiché par chacun des trois scripts suivants ? Veuillez n'écrire que ce qui sera *réellement* affiché par chacun des scripts : il ne faut donc pas répéter ou noter des instructions, etc.

a) **Fonction récursive.**

```
def fonctionRecursive(i):
    if i >= 0:
        print "a"
        fonctionRecursive(i - 1)
        fonctionRecursive(i - 2)
        print "b"

fonctionRecursive(3)
```

b) **Listes.**

```
liste1 = [1, 2, 3]
liste2 = [4, 5, 6]
liste3 = [7, 8, 9]
liste4 = [10, 11, 12]
liste2.append(7)
liste3 = liste3.append(10)
liste1.append(liste2)
liste4 = liste4 + liste1

print liste1
print liste2
print liste3
print liste4
```

c) Copie d'objets.

```
import copy

class Dimension(object):
    def __init__(self, size = 180, weight = 75):
        self.size = size
        self.weight = weight

class Personne(object):
    def __init__(self):
        self.name = ""
        self.age = 0
        self.dimension = Dimension()

    def takeKg(self, a):
        self.dimension.weight += a

    def takeCm(self, a):
        self.dimension.size += a

robin = Personne()
robin.age = 37
robin.name = "hood"
robin.dimension.size = 170
robin.dimension.weight = 73

oliver = Personne()
oliver.age = 28
oliver.name = "queen"
oliver.dimension.size = 186
oliver.dimension.weight = 78

hawkeye=copy.copy(oliver)
robin=copy.deepcopy(oliver)

print robin.dimension.weight
print oliver.dimension.weight
print hawkeye.dimension.weight

oliver.takeKg(5)
print robin.dimension.weight
print oliver.dimension.weight
print hawkeye.dimension.weight

oliver.name = oliver.name + " the hood"
print robin.name
print oliver.name
print hawkeye.name
```

Problème 2 : Stars et fans

Soit la classe `Person` représentant une personne. Chaque objet de la classe personne possède un attribut `knows`. Cet attribut contient une liste d'objets de type `Person` que la personne courante connaît. Par exemple, si `ed`, `bill` et `john` sont trois objets de type `Person`, et que `ed` connait les deux autres, alors la valeur de `ed.knows` sera équivalente à `[bill, john]`.

Soit une liste `people` contenant des objets de type `Person`. Une *star* dans `people` est une personne connue par au moins la moitié des personnes de `people`. Un *fan* dans `people` est une personne connaissant toutes les stars.

Questions :

a) Écrivez une fonction `stars(people)` qui, étant donnée une liste `people` de personnes, retourne une liste des stars.

b) Écrivez une fonction `fans(people)` qui, étant donnée une liste `people` de personnes, retourne une liste des fans.

Vous pouvez supposer que les objets de type `Person` ont une méthode `__hash__` définie et qu'ils peuvent donc être utilisés comme clés d'un dictionnaire.

Problème 3 : Permutations

Vous allez devoir générer récursivement toutes les permutations d'un mot donné. Par exemple, à partir du mot *algo*, les 24 mots générés seront : algo, lago, gola, laog, goal,... Afin de faciliter le travail, nous supposons que les mots considérés ne comportent pas deux fois le même caractère.

Le principe fonctionne comme suit :
- Si le mot n'est composé que d'un seul caracètre, celui-ci est la seule permutation du mot ;
- Sinon, pour chaque caractère c du mot, on crée toutes les permutations possibles commençant par ce caractère. Pour se faire, on crée récursivement toutes les permutations possibles pour le mot sans le caractère c. On place ensuite ce caractère devant chacune des permutations obtenues par l'appel récursif.

Par exemple, pour le mot *algo* et le caractère a, on crée toutes les permutations de lgo. On obtient alors les mots lgo, gol, log, glo, olg et ogl. Si on place le caractère a devant ces mots (algo, agol, alog,...), on obtient toutes les permutations du mot $algo$ commençant par le caractère a. Il faut alors appliquer le même procédé avec les trois autres caractères pour obtenir toutes les permutations du mot $algo$.

Questions :

a) Donnez le code python d'une fonction **récursive** `permutations(m)` qui étant donné une chaine de caractères `m` ne contenant pas deux caractères identiques, retourne une liste de chaines de caractères contenant toutes les permutations de `m`.

b) Quels seraient les dix premiers éléments de la liste des permutations retournée par votre algorithme avec le mot `bouc` en entrée ?

Problème 4 :

Voici le code Python d'une fonction `process` qui prend en entrée une liste qui contient des entiers strictement positifs (précondition). Cette fonction ne retourne rien mais modifie l'ordre des entiers contenus dans la liste de telle sorte à placer les nombres pairs avant les nombres impairs.

```
def process(A):
    i = 0
    j = len(A) - 1
    while i < j:
        if A[i] % 2 == 0:
            i = i + 1
        else:
            A[i], A[j] = A[j], A[i]
            j = j - 1
```

Questions :

a) Si la liste `[1, 3, 6, 2, 8, 7, 2]` est passée en entrée de `process`, que contiendra-t-elle à la fin de l'exécution de cette fonction ?

b) Prouvez que la fonction `process` s'arrête toujours, si `A` est une liste (qui peut être vide).

c) Prouvez que l'assertion suivante est un invariant de boucle.

> *Les éléments de A dont les indices sont $< i$ sont pairs et les éléments de A dont les indices sont $> j$ sont impairs.*

d) En vous appuyant sur les points précédents, prouvez que la fonction `process` est exacte, c'est-à-dire qu'elle réalise correctement son travail qui consiste à placer les nombres pairs avant les nombres impairs.

e) Donnez la complexité (notation grand-O) dans le pire des cas de la fonction `process` en fonction du nombre n d'éléments contenus dans la liste `A`. Il n'est pas nécessaire de justifier votre réponse.

Annexe C

Séquence d'enseignement : introduction à l'algorithmique

Le but de cette séquence de cours est d'introduire quelques notions relatives à l'algorithmique.
Pour commencer, nous allons montrer que nous rencontrons des algorithmes quotidiennement. La notion d'algorithme va alors émerger en comparant ces différents exemples. Nous étudierons également des algorithmes que nous rencontrons en mathématiques depuis l'école primaire.
La définition d'algorithme posée, nous expliquerons certaines notions utiles dans la compréhension d'un algorithme. Sur la base de ces notions, nous aurons à en tester plusieurs.
Tout comme un exemple ne suffit pas à prouver un théorème, nous devrons prouver les algorithmes.
Finalement, nous aborderons une notion spécifique à l'algorithmique : la complexité.

C.1 Introduction

C.1.1 Définition

L'*algorithmique*, discipline se situant à l'intersection entre les mathématiques et l'informatique, est la « science des *algorithmes* ».

Nous rencontrons des algorithmes fréquemment, par exemple lorsque nous mettons un plat à réchauffer au micro-ondes ou encore lorsque nous démarrons une voiture.

Pour réchauffer un plat au micro-ondes, nous suivons les étapes suivantes :

Algorithme 1

1. Mettre le plat dans le micro-ondes.
2. Fermer la porte du micro-ondes.
3. Choisir la puissance et le temps de chauffe.
4. Mettre en route le micro-ondes.
5. Attendre que le temps se soit écoulé.
6. Ouvrir la porte du micro-ondes.
7. Prendre son plat.

Pour démarrer une voiture (fonctionnant avec une clé), nous procédons comme suit :

Algorithme 2

1. Insérer la clé dans le contacteur à clé.
2. S'assurer que le levier de vitesses se trouve en position neutre (point mort).
3. Mettre le contact.
4. Si le moteur part en moins de six secondes, relâcher la clé.
5. Si le moteur ne part pas en six secondes :
   - relâcher la clé,
   - attendre 10 secondes,
   - répéter les étapes 3, 4 et 5, mais pas plus de 5 fois en tout.
6. Si le moteur ne part pas, appeler le garage.

À partir de ces deux exemples, nous pouvons donner quelques éléments qui caractérisent un algorithme.

Tout d'abord, un algorithme sert à **résoudre un problème** (réchauffer son plat et démarrer une voiture dans les deux exemples). Mieux encore, un algorithme sert à résoudre une classe de problèmes (une *famille d'instances*).

En effet, l'algorithme pour mettre réchauffer un plat au micro-ondes fonctionne si j'y insère un plat de pâtes, mais il fonctionne aussi si je veux faire réchauffer de la soupe. De même, l'algorithme pour le démarrage de la voiture fonctionne pour ma voiture (une *instance* du problème), mais il fonctionne également pour la voiture de mon voisin. Mieux encore, il fonctionne pour toutes les voitures à clé (*famille d'instances*).

Ensuite, le **nombre d'étapes** dans un algorithme est **fini**. Le problème du micro-ondes comporte sept étapes (elles sont numérotées). Pour le problème de la voiture, compter le nombre d'étapes est plus compliqué car cela dépend du nombre d'essais qu'il faut pour faire démarrer le moteur. Dans le meilleur des cas (la voiture démarre au premier essai), il y a quatre étapes (les étapes 1 à 4). Dans le pire des cas (la voiture ne démarre toujours pas après les cinq essais), le nombre d'étapes est également fini car nous n'essayons pas indéfiniment de mettre en marche le moteur, nous appelons le garage.
Si nous décidions d'essayer de mettre en route le moteur jusqu'à ce qu'il se mette réellement en marche, nous pourrions avoir un nombre infini d'étapes (car nous n'avons pas la certitude qu'il va se mettre en route).

Avant de donner une définition d'*algorithme*, il convient de définir ce que sont une *famille d'instances* et une *instance*.

- ⋆ Une **famille d'instances** d'un problème est l'ensemble des configurations possibles du problème.
- ⋆ Une **instance** est un cas particulier d'une configuration du problème.

Par exemple, l'ensemble des naturels non nuls, noté $\mathbb{N}_0$, peut être une famille d'instances ; 1 est une instance.

Nous pouvons maintenant définir la notion d'*algorithme*.

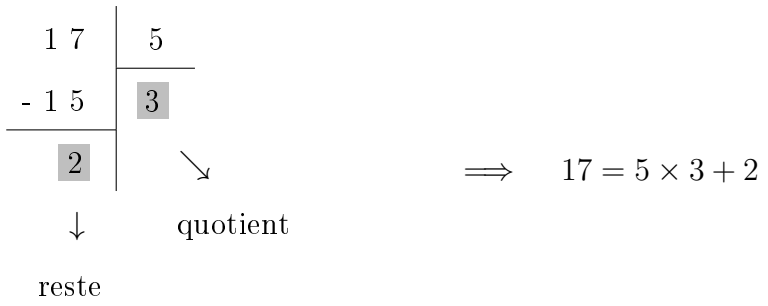
> Un **algorithme** est une suite d'opérations qui, effectuées comme une séquence déterminée, fournissent, en un nombre fini d'étapes, la solution à un problème pour toute instance de celui-ci.

C.1.2 Exemples mathématiques

Voici deux algorithmes que vous avez déjà rencontrés au cours de votre scolarité. Le premier sert à déterminer le quotient et le reste d'une division euclidienne. Le deuxième permet de trouver les solutions d'une équation du second degré.

Division euclidienne

Avant de travailler sur l'algorithme de la division euclidienne, il n'est pas inutile de se rappeler le principe de celle-ci.
À titre d'exemple, nous allons effectuer la division euclidienne de 17 par 5.



$$\Longrightarrow \quad 17 = 5 \times 3 + 2$$

Pour calculer le quotient de la division euclidienne de 17 par 5, nous déterminons le nombre de fois que 5 peut se mettre dans 17. Nous trouvons 3. Ensuite, nous calculons le produit du quotient et de 5 et nous soustrayons cette valeur à 17, ce qui nous donne le reste. Nous obtenons donc un reste de $17 - 5 \times 3 = 2$.

L'algorithme qui décrit la division euclidienne s'appuie sur ce principe.

Algorithme 3

Entrée : deux nombres naturels a et b, avec $b \neq 0$.
Sortie : deux nombres naturels q et r, tels que q est le quotient de a par b et r, le reste de cette division.

1. $r \leftarrow a$
2. $q \leftarrow 0$
3. **tant que** $r \geqslant b$ **faire**
4. $\quad r \leftarrow r - b$
5. $\quad q \leftarrow q + 1$
6. **fin tant que**
7. **retourner** q et r

Avant d'exécuter cet algorithme pour la division euclidienne de 17 par 5, expliquons d'abord tout ce qui se trouve dans l'algorithme.

- ⋆ **Entrée** : les entrées représentent les instances du problème. Ici, $a \in \mathbb{N}$ et $b \in \mathbb{N}_0$, où $\mathbb{N}$ et $\mathbb{N}_0$ sont des familles d'instances.
- ⋆ **Sortie** : les sorties désignent la réponse au problème posé. Ici, le problème est de déterminer le quotient q et le reste r de la division euclidienne de a par b.
- ⋆ $r \leftarrow a$: cette expression signifie que la valeur a est attribuée à la variable r. Il s'agit d'une *affectation*.
- ⋆ **tant que** $r \geqslant b$ **faire** : il faut répéter tout ce qui se trouve entre le mot « **faire** » et le « **fin tant que** » autant de fois que nécessaire, jusqu'à ce que la condition $r \geqslant b$ ne soit plus vérifiée.

Exécutons maintenant cet algorithme pour $a = 17$ et $b = 5$.

1. $r \leftarrow 17$
2. $q \leftarrow 0$

3. $17 \geqslant 5$? Oui, donc il faut exécuter l'intérieur de la boucle.
4. $r \leftarrow 17 - 5 = 12$
5. $q \leftarrow 0 + 1 = 1$

3. $12 \geqslant 5$? Oui, donc il faut exécuter l'intérieur de la boucle.
4. $r \leftarrow 12 - 5 = 7$
5. $q \leftarrow 1 + 1 = 2$

3. $7 \geqslant 5$? Oui, donc il faut exécuter l'intérieur de la boucle.
4. $r \leftarrow 7 - 5 = 2$
5. $q \leftarrow 2 + 1 = 3$

3. $2 \geqslant 5$? Non, donc il faut passer au « **fin tant que** » (ligne 6).
7. **retourner** 3 et 2.

Résolution de l'équation du second degré

Pour résoudre l'équation du second degré $ax^2 + bx + c = 0$, la première étape est de calculer le discriminant (ou réalisant) $\Delta = b^2 - 4ac$.
Ensuite, suivant la valeur du discriminant, il y a deux solutions distinctes ($\Delta > 0$), une solution ($\Delta = 0$) ou pas de solution dans $\mathbb{R}$ ($\Delta < 0$).

L'algorithme suivant permet de résoudre n'importe quelle équation du second degré dans $\mathbb{R}$.

Algorithme 4

Entrée : trois nombres réels a, b et c, avec $a \neq 0$.
Sortie : deux réels x_1 et x_2, solutions de l'équation $ax^2 + bx + c = 0$ (ou un réel x_1 ou rien).

1. $d \leftarrow b^2 - 4.a.c$
2. **si** $d > 0$ **alors**
3. $\quad x_1 \leftarrow \frac{-b+\sqrt{d}}{2.a}$
4. $\quad x_2 \leftarrow \frac{-b-\sqrt{d}}{2.a}$
5. $\quad$ **retourner** x_1 et x_2
6. **sinon**
7. $\quad$ **si** $d = 0$ **alors**
8. $\quad\quad x_1 \leftarrow \frac{-b}{2.a}$
9. $\quad\quad$ **retourner** x_1
10. $\quad$ **fin si**
11. **fin si**

Dans cet algorithme se trouve une nouvelle structure : la structure « **si...alors...sinon** ».
Il faut d'abord évaluer le premier test ($d > 0$?). Si ce test est évalué à « vrai » (d est strictement positif), alors il faut exécuter ce qui suit le « **alors** ». Si ce test est évalué à « faux » (d est négatif ou nul), c'est ce qui suit le « **sinon** » qu'il faut exécuter.
Le deuxième « **si** » se trouve dans le « **sinon** ». Le deuxième test ne sera donc effectué que si d est négatif ou nul. Nous procédons de même pour l'exécution.

Exemples : calculer, à l'aide de cet algorithme, les solutions des équations suivantes :

⋆ $x^2 - 3x + 2 = 0$.

1. $d \leftarrow (-3)^2 - 4.1.2 = 1$
2. $d > 0$? Oui, donc nous exécutons ce qui se trouve après le « **alors** ».
3. $x_1 \leftarrow \frac{3+\sqrt{1}}{2.1} = 2$
4. $x_2 \leftarrow \frac{3-\sqrt{1}}{2.1} = 1$
5. retourner 2 et 1

⋆ $x^2 + 2x + 1 = 0$.

1. $d \leftarrow 2^2 - 4.1.1 = 0$
2. $d > 0$? Non, donc nous passons directement au « **sinon** » (ligne 6).
7. $d = 0$? Oui, donc nous exécutons ce qui se trouve après le « **alors** » de la ligne 8.
8. $x_1 \leftarrow \frac{-2}{2.1} = -1$
9. retourner -1

⋆ $x^2 - x + 1 = 0$.

1. $d \leftarrow (-1)^2 - 4.1.1 = -3$
2. $d > 0$? Non, donc nous passons directement au « **sinon** » (ligne 6).
7. $d = 0$? Non, donc nous n'exécutons pas ce qui se trouve après le « **alors** ». L'algorithme ne retourne rien.

C.2 Quelques notions utiles pour comprendre un algorithme

C.2.1 Entrées et sorties

Tout algorithme comporte des entrées et des sorties qui sont spécifiées au début de l'algorithme.

Les entrées représentent les instances, c'est-à-dire les données initiales. Pour des situations de la vie courante, les entrées peuvent être vues comme la

situation initiale.
Pour le problème du micro-ondes, l'entrée est le fait que le plat est froid (situation initiale). Pour le démarrage de la voiture, la situation initiale est le fait que la voiture est à l'arrêt.

Les sorties représentent les données finales ou les résultats. Pour des situations de la vie quotidienne, les sorties peuvent être vues comme la situation finale.
Pour le micro-ondes, la sortie est le fait que le plat est chaud (situation finale). Pour le démarrage de la voiture, la sortie est le fait que la voiture démarre.

C.2.2 Affectation

L'*affectation* est une opération permettant d'assigner une valeur à une variable. Par exemple, l'opération $x \leftarrow 1$ est une *affectation*. Elle se lit « attribuer la valeur 1 à x ».

x est appelé *variable informatique* parce qu'au sein d'un même algorithme, x peut prendre successivement plusieurs valeurs, contrairement à une variable mathématique qui ne contient qu'une seule valeur durant tout un calcul.
Par exemple, lorsque nous avons exécuté l'algorithme d'Euclide pour la division euclidienne de 17 par 5, la variable r a pris successivement les valeurs 17, 12, 7 et 2.

C.2.3 « Si...alors...sinon »

Problème : écrire un algorithme qui calcule la valeur absolue d'un nombre réel. L'algorithme doit fonctionner quel que soit le réel reçu.
Pour résoudre ce problème, une nouvelle structure doit être introduite.

Solution : nous allons utiliser la structure « **si...alors...sinon** » (structure conditionnelle). Si la condition située après le « **si** » est évaluée à vrai, il faut exécuter ce qui se trouve après le « **alors** ». Si la condition est évaluée à faux, il faut exécuter ce qui se trouve après le « **sinon** ».

Une structure conditionnelle s'exprime comme suit :

si condition **alors**
    instructions 1
sinon
    instructions 2
fin si

Pour exécuter une structure conditionnelle, il faut d'abord déterminer si la condition est vraie ou fausse. Pour cela, nous devons nous poser la question suivante : « la condition est-elle vraie ? ». Si la réponse est oui, c'est le bloc « instructions 1 » qui est exécuté. Si la réponse est non, c'est le bloc « instructions 2 » qui l'est.

Remarquons que, pour une condition donnée, les blocs « instructions 1 » et « instructions 2 » ne pourront pas être exécutés tous les deux car la condition ne peut pas être à la fois vraie et fausse en même temps. De même, un des deux blocs sera forcément exécuté car la condition est soit vraie, soit fausse.

Le « **fin si** » indique que la structure conditionnelle est terminée et tout ce qui se trouve après le « **fin si** » ne dépend plus de la condition (les mêmes opérations sont exécutées, que ce soit le bloc « instructions 1 » ou le bloc « instructions 2 » qui ait été exécuté précédemment).

Nous sommes maintenant en mesure d'écrire un algorithme donnant la valeur absolue d'un nombre réel :

Algorithme 5

Entrée : un nombre réel x.
Sortie : abs, valeur absolue de x.

1. **si** $x \geqslant 0$ **alors**
2. $\quad abs \leftarrow x$
3. **sinon**
4. $\quad abs \leftarrow -x$
5. **fin si**
6. **retourner** *abs*

Exemples : calculer, à l'aide de cet algorithme, la valeur absolue des réels suivants :

⋆ $x = 3$.

1. $3 \geqslant 0$? Oui, donc nous exécutons ce qui se trouve après le « **alors** ».
2. $abs \leftarrow 3$
6. **retourner** 3

⋆ $x = -5$.

1. $-5 \geqslant 0$? Non, donc nous passons directement au « **sinon** » (ligne 3).
4. $abs \leftarrow -(-5) = 5$
6. **retourner** 5

⋆ $x = 0$.

1. $0 \geqslant 0$? Oui, donc nous exécutons ce qui se trouve après le « **alors** ».
2. $abs \leftarrow 0$
6. **retourner** 0

C.2.4 Boucle « pour tout »

Problème : écrire un algorithme qui calcule la somme des n premiers nombres naturels (en supposant que nous ne connaissons pas la formule).
Comme précédemment, une nouvelle structure est ici aussi nécessaire.

Solution : nous allons utiliser une boucle **pour tout**. Une boucle permet de répéter l'exécution d'une ou de plusieurs instructions.

Une boucle « **pour tout** » s'exprime comme suit :

```
pour tout variable allant de val1 à val2 faire
    instructions
fin pour tout
```

Lorsqu'il faut exécuter une boucle « **pour tout** », il faut commencer par attribuer la valeur « val1 » à la variable et exécuter le bloc « instructions ». Une fois le bloc « instructions » exécuté, il faut attribuer la valeur qui suit « val1 » à la variable et exécuter à nouveau le bloc « instructions ». Ce processus est répété autant de fois que nécessaire, jusqu'à ce que la variable atteigne la valeur « val2 ». Le processus est alors exécuté pour la dernière fois. Lorsque le bloc « instructions » est exécuté pour la dernière fois, il faut exécuter les opérations qui suivent le « **fin pour tout** ».

Voici un algorithme nous donnant la somme des n premiers nombres naturels :

Algorithme 6

Entrée : un nombre naturel n.
Sortie : la somme des n premiers naturels.

1. $somme \leftarrow 0$
2. **pour tout** i allant de 1 à n **faire**
3. $\quad somme \leftarrow somme + i$
4. **fin pour tout**
5. **retourner** *somme*

Exemple : calculer, à l'aide de cet algorithme, la somme des 4 premiers nombres naturels :

1. $somme \leftarrow 0$
2. **pour tout** i allant de 1 à 4 **faire**

2. $i \leftarrow 1$
3. $somme \leftarrow 0 + 1 = 1$

2. $i \leftarrow 2$
3. $somme \leftarrow 1 + 2 = 3$

2. $i \leftarrow 3$
3. $somme \leftarrow 3 + 3 = 6$

2. $i \leftarrow 4$
3. $somme \leftarrow 6 + 4 = 10$

4. **fin pour tout**
5. **retourner** 10

Remarque : si n vaut 0, la ligne 2 nous dit qu'il faut faire varier i de 1 à 0. Dans ce cas, la convention est que le corps de la boucle n'est jamais exécuté. Nous devons donc directement passer à la ligne 4 (« **fin pour tout** »).

C.2.5 Boucle « tant que »

Il existe un autre type de boucle : la boucle « **tant que** ». Nous l'avons déjà rencontrée dans l'algorithme pour la division euclidienne.
Une boucle « **tant que** » se base sur un fonctionnement similaire à celui d'une boucle « **pour tout** ». Une grande différence par rapport à la boucle « **pour tout** » est que la variable, dans une boucle « **tant que** », ne passe pas automatiquement à la valeur suivante.

Une boucle « **tant que** » s'exprime comme suit :

tant que condition **faire**
    instructions
fin tant que

Lorsqu'il faut exécuter une boucle « **tant que** », il faut, en premier lieu, étudier la véracité de la condition.

- ⋆ Si la condition est fausse, il faut directement passer au « **fin tant que** ».
- ⋆ Si la condition est vraie, nous devons exécuter le bloc « instructions ». Ensuite, nous devons à nouveau déterminer la véracité de la condition.
  - Si la condition est fausse, il faut directement passer au « **fin tant que** ».
  - Si la condition est vraie, nous devons exécuter à nouveau le bloc « instructions ». Ensuite, nous devons à nouveau déterminer la véracité de la condition.

Ce processus est répété jusqu'à ce que la condition soit fausse. À ce moment-là, il faut passer au « **fin tant que** ». Ce sont alors les opérations situées après le « **fin tant que** » qui seront exécutées.

Voici un algorithme nous donnant la somme des n premiers nombres naturels :

Algorithme 7

Entrée : un nombre naturel n.
Sortie : la somme des n premiers naturels.

1. $somme \leftarrow 0$
2. $i \leftarrow 1$
3. **tant que** $i \neq n+1$ **faire**
4. $\quad somme \leftarrow somme + i$
5. $\quad i \leftarrow i + 1$
6. **fin tant que**
7. **retourner** *somme*

Exemple : calculer, à l'aide de cet algorithme, la somme des 4 premiers nombres naturels :

1. $somme \leftarrow 0$
2. $i \leftarrow 1$

3. $1 \neq 5$? Oui, donc il faut exécuter l'intérieur de la boucle.
4. $somme \leftarrow 0 + 1 = 1$
5. $i \leftarrow 1 + 1 = 2$

3. $2 \neq 5$? Oui, donc il faut exécuter l'intérieur de la boucle.
4. $somme \leftarrow 1 + 2 = 3$
5. $i \leftarrow 2 + 1 = 3$

3. $3 \neq 5$? Oui, donc il faut exécuter l'intérieur de la boucle.
4. $somme \leftarrow 3 + 3 = 6$
5. $i \leftarrow 3 + 1 = 4$

3. $4 \neq 5$? Oui, donc il faut exécuter l'intérieur de la boucle.
4. *somme* $\leftarrow 6 + 4 = 10$
5. $i \leftarrow 4 + 1 = 5$

3. $5 \neq 5$? Non, donc il faut passer au « **fin tant que** » (ligne 6).
7. **retourner** 10

C.2.6 Exercices

Voici trois algorithmes. Déterminez la sortie de chacun d'entre eux.

Algorithme 8

Entrée : deux réels x et y.
Sortie : ? ? ?

1. **si** $x \geqslant y$ **alors**
2. $\quad$ *valeur* $\leftarrow x$
3. **sinon**
4. $\quad$ *valeur* $\leftarrow y$
5. **fin si**
6. **retourner** *valeur*

Algorithme 9

Entrée : un nombre réel non nul x et un nombre naturel n.
Sortie : ? ? ?

1. $y \leftarrow 1$
2. **pour tout** k allant de 1 à n **faire**
3. $\quad$ $y \leftarrow y \times x$
4. **fin pour tout**
5. **retourner** y

Algorithme 10

Entrée : deux nombres naturels non nuls a et b.
Sortie : ? ? ?

1. $u \leftarrow a$
2. $v \leftarrow b$
3. **tant que** $v \neq 0$ **faire**
4. $\quad z \leftarrow v$
5. $\quad q \leftarrow E(\frac{u}{v})$
6. $\quad v \leftarrow u - q \times v$
7. $\quad u \leftarrow z$
8. **fin tant que**
9. **retourner** u

Remarque : $E(x)$ désigne la partie entière du réel x. La partie entière d'un nombre réel x est l'unique nombre entier tel que
$E(x) \leqslant x < E(x) + 1$.
Par exemple, $E(2,3) = 2$ car $2 \leqslant 2,3 < 3$.

C.3 Preuve d'un algorithme

Pour déterminer le but des trois algorithmes précédents, vous avez probablement testé ces différents algorithmes pour différentes valeurs initiales appartenant à l'ensemble des entrées et en avez déduit les sorties.

Comment savoir si l'algorithme va toujours donner le bon résultat, c'est-à-dire quelles que soient les entrées ?
Une autre question est de savoir si nous avons la certitude que tout algorithme va se terminer.

Pour ce faire, nous allons prouver que chaque algorithme se termine en un nombre fini d'étapes (c'est ce qu'on appelle la **terminaison**) et qu'il donne le résultat attendu (**correction** d'un algorithme).

C.3.1 Terminaison

Prouver la terminaison d'un algorithme, c'est montrer que le nombre d'étapes exécutées par l'algorithme est fini.

Algorithme 8 :

L'algorithme se termine en un nombre fini d'étapes. En effet, quelles que soient les valeurs de x et de y, le test $x \geqslant y$ doit être évalué (1 étape).
- Si le test est vrai ($x \geqslant y$), l'opération 2 est effectuée. Ensuite, il faut passer au « **fin si** » et la dernière opération, « **retourner** valeur », est alors exécutée. Dans ce cas, il y a trois étapes.
- Si le test est faux ($x < y$), il faut directement passer au « **sinon** » (ligne 3). Ensuite, l'opération 4 est exécutée. Après le « **fin si** », l'opération 6 est exécutée. Dans ce cas, trois étapes suffisent également.

En conclusion, quelles que soient les valeurs de x et de y, l'algorithme se termine en trois étapes. Le nombre d'étapes est donc fini.

Algorithme 9 :

Quelles que soient les entrées, l'opération 1 ($y \leftarrow 1$) est exécutée (1 étape). Dans la boucle, entre le « **pour tout** » et le « **fin pour tout** », il n'y a qu'une opération : $y \leftarrow y \times x$. Cependant, comme cette opération se trouve dans une boucle « **pour tout** », il y a de fortes chances pour qu'elle soit exécutée plusieurs fois.
Pour connaître le nombre de fois que cette opération sera exécutée, nous allons déterminer le nombre de passages dans la boucle. Pour cela, il suffit de déterminer le nombre de valeurs que k va prendre. Nous savons que k ne prend que les valeurs comprises entre 1 et n (inclus). Dès lors, lorsque k aura pris toutes les valeurs comprises entre 1 et n, la boucle ne sera plus exécutée. De 1 à n, il y a n valeurs, donc la boucle sera exécutée n fois. Comme n est un nombre fini, le contenu de la boucle sera donc effectué un nombre fini de fois.
Après être sorti de la boucle, seule reste l'opération 5 à effectuer.
En conclusion, le nombre d'étapes qui doivent être effectuées dans cet algorithme est fini.

Algorithme 10 :

Dans cet algorithme seront d'abord exécutées les opérations 1 et 2 (2 étapes). Ensuite, nous entrons dans la boucle « **tant que** ». Dans cette boucle, il

y a 4 opérations (les opérations 4, 5, 6 et 7). Comme pour l'algorithme précédent, nous devons déterminer le nombre de fois que cette boucle va être exécutée. Cependant, contrairement à l'algorithme précédent, le nombre de fois que la boucle va être exécutée n'est pas connu à l'avance. En effet, nous ne savons pas dire a priori après combien d'itérations la variable v va atteindre la valeur 0.
Puisque nous ne savons pas déterminer le nombre de fois que la boucle sera exécutée, nous allons donc devoir procéder autrement.

Pour sortir de la boucle, il faut que la condition $v \neq 0$ soit fausse. Il faut donc que la variable v soit égale à 0. Si la variable v atteint la valeur 0 après un certain nombre d'itérations, alors le nombre de fois que nous allons répéter la boucle est fini.

Montrons donc que v atteint effectivement la valeur 0.

1. La valeur de la variable v diminue strictement après chaque passage dans la boucle.

   - Si $u < v$, alors $E(\frac{u}{v}) = 0$. En effet, $\frac{u}{v} < 1$ et $\frac{u}{v} \geqslant 0$ car u et v sont deux nombres naturels donc positifs.
     Comme $E(\frac{u}{v}) = 0$, par les opérations dans la boucle « **tant que** », les variables sont actualisées comme suit : $v \leftarrow u - E(\frac{u}{v}) \times v = u$ et $u \leftarrow v$. Les valeurs des deux variables sont donc inversées avant d'exécuter à nouveau la boucle.
   - Si $u \geqslant v$, alors $E(\frac{u}{v}) \geqslant 1$.
     Après un passage dans la boucle, la valeur de la variable v ($v \leftarrow u - q \times u$) est-elle strictement plus petite que la valeur de v avant l'affectation ? Autrement dit, l'inégalité $u - q \times u < v$ est-elle vérifiée ?

$$
\begin{aligned}
& u - q \times v \overset{?}{<} v \\
\Leftrightarrow \quad & u \overset{?}{<} q \times v + v \\
\Leftrightarrow \quad & u \overset{?}{<} (q+1) \times v \\
\Leftrightarrow \quad & \frac{u}{v} \overset{?}{<} q + 1
\end{aligned}
$$

Cette dernière inégalité est vraie. En effet, comme q représente la partie entière de $\frac{u}{v}$, par définition de la partie entière, $q \leqslant \frac{u}{v} < q + 1$.

Remarque : nous avons pu effectuer la division par v car v est différent de 0 comme nous sommes à l'intérieur de la boucle (condition pour y entrer : $v \neq 0$).

La valeur de la variable v diminue donc strictement après chaque passage dans la boucle.

2. La valeur de la variable v reste-t-elle toujours positive ? Vérifions donc que la valeur de v après l'affectation $v \leftarrow u - q \times v$ est positive ou nulle.

$$\begin{aligned} & u - q \times v \overset{?}{\geqslant} 0 \\ \Leftrightarrow \quad & u \overset{?}{\geqslant} q \times v \\ \Leftrightarrow \quad & \frac{u}{v} \overset{?}{\geqslant} q \end{aligned}$$

Cette dernière inégalité est vraie car q est la partie entière de $\frac{u}{v}$.

La valeur de v est donc toujours positive ou nulle.

3. Après chaque passage dans la boucle, la nouvelle valeur de la variable v est strictement plus petite que la valeur avant affectation. À chaque passage dans la boucle, nous soustrayons un nombre naturel ($q \times v$ car la partie entière, q, est un naturel) à un nombre naturel u. Le résultat est donc un nombre naturel car nous avons montré que la valeur restait positive.
   Grâce à cet algorithme, nous construisons donc une suite de nombres naturels strictement décroissante. Donc v va atteindre la valeur 0 après un certain nombre d'itérations.

En conclusion, nous venons de montrer que la valeur de la variable v va bien atteindre la valeur 0. De ce fait, la boucle sera exécutée un nombre fini de fois.

Une fois sorti de la boucle, il ne nous reste plus qu'à exécuter l'opération 9 qui consiste à retourner la valeur u.

L'algorithme se termine donc en un nombre fini d'étapes.

Conjecture de Syracuse :

Prouver la terminaison d'un algorithme n'est pas toujours chose aisée. À l'heure actuelle, les chercheurs ne sont pas encore parvenus à démontrer la terminaison de certains supposés algorithmes (« supposés » car, par définition, un algorithme comporte un nombre fini d'étapes).
Le supposé algorithme suivant, appelé « conjecture de Syracuse », est un exemple d'algorithme dont la terminaison n'a pas encore été prouvée.

Conjecture de Syracuse

Entrée : un nombre naturel n non nul.
Sortie : rien.

1. **tant que** $n \neq 1$ **faire**
2. **si** n est pair **alors**
3. $n \leftarrow \frac{n}{2}$
4. **sinon**
5. $n \leftarrow n \times 3 + 1$
6. **fin si**
7. **fin tant que**

Pour cet algorithme, il n'a pas encore été prouvé que le nombre d'étapes était fini. En effet, nous n'avons aucunement la certitude que la variable n va, à un moment donné, atteindre la valeur 1 (pour nous permettre de sortir de la boucle) puisque, dès que la valeur de n est un nombre impair, la nouvelle valeur de n, après affectation, est plus grande que la valeur initiale de n, et donc encore plus éloignée de 1.

C.3.2 Correction

Prouver la correction d'un algorithme revient à montrer que l'algorithme donne le résultat escompté quelles que soient les valeurs initiales (appartenant aux entrées) reçues par ce dernier.

Algorithme 8 :

L'algorithme fournit bien le maximum entre les valeurs x et y, quels que soient $x \in \mathbb{R}$ et $y \in \mathbb{R}$.

Pour montrer cela, il faut distinguer trois cas distincts :

1. Le maximum entre x et y est x.
2. Le maximum entre x et y est y.
3. x et y ont même valeur.

Ce sont bien les trois seuls cas possibles car, lorsque nous comparons deux valeurs, soit elles sont égales, soit différentes, et alors l'une d'elles est plus grande que l'autre.

Étudions ces trois cas séparément pour montrer que, dans tous les cas, l'algorithme fournit le résultat attendu.

1. Si le maximum entre x et y est x, alors $x > y$. Dès lors, le test « $x \geqslant y$? » est vrai et c'est donc l'opération 2 qui est exécutée : *valeur* $\leftarrow x$.
   Après, il faut passer à la dernière opération, « **retourner** x ».
   L'algorithme nous donne donc bien le maximum entre x et y.
2. Si le maximum entre x et y est y, alors $y > x$. Dès lors, le test « $x \geqslant y$? » est faux et c'est donc l'opération située après le « **sinon** » (ligne 3) qui est exécutée : *valeur* $\leftarrow y$.
   Après, nous passons à la dernière opération, « **retourner** y ».
   L'algorithme nous fournit donc bien le maximum entre x et y.
3. Si x et y ont tous deux la même valeur ($x = y$), alors le maximum vaut x ou y indifféremment. Dès lors, le test « $x \geqslant y$? » est évalué à vrai et c'est donc l'opération 2 qui est exécutée : *valeur* $\leftarrow x$.
   En passant à la dernière opération, l'algorithme retourne la valeur x.
   Une fois de plus, l'algorithme détermine bien le maximum entre x et y.

Conclusion : en étudiant les différents cas possibles, nous avons pu constater que l'algorithme nous fournissait toujours le résultat escompté. Nous avons

donc prouvé la correction de cet algorithme.

Algorithme 9 :

L'algorithme nous donne bien x^n, quels que soient $x \in \mathbb{R}_0$ et $n \in \mathbb{N}$.

Pour le montrer, il n'est pas possible, comme dans l'exemple précédent, de répertorier tous les cas possibles car pour cet algorithme, ils sont trop nombreux.
Nous allons donc employer une technique de démonstration couramment utilisée en mathématiques : la **démonstration par récurrence**.

Principe de la démonstration par récurrence :

La démonstration par récurrence permet de démontrer des propriétés relatives aux nombres naturels. Elle permet de démontrer des propriétés du type $\forall k \in \mathbb{N} : P(k)$.
Les étapes d'une démonstration par récurrence sont les suivantes :

1. Pas initial : montrer que $P(0)$ est vraie.
2. Pas de récurrence : nous supposons que la propriété P est vraie pour $k-1$, c'est-à-dire $P(k-1)$ est vraie. Nous appelons $P(k-1)$ l'*hypothèse de récurrence*. Sur la base de cette hypothèse de récurrence, nous devons montrer que la propriété P est également vraie pour k ($P(k)$ est vraie).

Pour prouver la correction de cet algorithme, nous allons prouver, au moyen d'une preuve par récurrence, qu'au $k^{\text{ème}}$ passage dans la boucle, nous allons avoir $y = x^k$ (pour $0 \leqslant k \leqslant n$).

Pas initial ($k = 0$)

Pour le cas où k vaut 0, nous devons donc montrer que $y = 1 (= x^0)$.

Si $k = 0$, c'est que nous ne sommes pas encore entrés dans la boucle « **pour tout** » (car dans la boucle, k varie de 1 à n).
Dans ce cas, seule l'opération 1 est exécutée, c'est-à-dire $y \leftarrow 1$.

Nous avons donc bien montré qu'avant d'entrer dans la boucle, la valeur 1 est attribuée à la variable y.

Pas de récurrence

Supposons qu'au $(k-1)^{\text{ème}}$ passage dans la boucle, nous avons bien $y = x^{k-1}$ (hypothèse de récurrence).
Nous devrons montrer qu'au $k^{\text{ème}}$ passage dans la boucle, nous aurons $y = x^k$.

Lors du $(k-1)^{\text{ème}}$ passage dans la boucle, par hypothèse de récurrence, nous avons $y = x^{k-1}$. Au passage suivant dans la boucle (le $k^{\text{ème}}$), nous avons l'affectation suivante :

$$\begin{aligned} y &\leftarrow y \times x \\ y &\leftarrow x^{k-1} \times x \\ y &\leftarrow x^k \end{aligned}$$

Au $k^{\text{ème}}$ passage dans la boucle, la valeur x^k est attribuée à la variable y.

Nous venons donc de montrer que si, au $(k-1)^{\text{ème}}$ passage dans la boucle, la valeur x^{k-1} était attribuée à la variable y, alors au $k^{\text{ème}}$ passage, c'est la valeur x^k qui est attribuée à y.

Conclusion : comme nous avons $y = x^k$ au $k^{\text{ème}}$ passage dans la boucle pour $0 \leqslant k \leqslant n$, en particulier, pour $k = n$ (dernier passage dans la boucle), nous avons $y = x^n$. C'est ce que nous voulions montrer.

Algorithme 10 :

L'algorithme nous donne bien le plus grand commun diviseur à a et à b, avec $a \in \mathbb{N}_0$ et $b \in \mathbb{N}_0$. Nous allons le prouver.

Pour cet algorithme, nous ne pouvons pas utiliser la preuve par récurrence car nous ne connaissons pas à l'avance les valeurs que va prendre la variable v.
Pour l'algorithme précédent, nous savions que k allait prendre successivement les valeurs $1, 2, ...$ et n et nous pouvions donc déterminer une propriété qui devait être vraie pour chaque valeur de k. Ici, ne connaissant pas les différentes valeurs de v, nous ne pouvons pas extraire une telle propriété vraie après chaque passage dans la boucle. Il n'est donc pas possible de prouver la

correction de cet algorithme au moyen d'une preuve par récurrence sur les valeurs de v.
Nous devons donc procéder autrement.

Notons $D(x, y)$ l'ensemble des diviseurs naturels communs à x et à y.

Après chaque passage dans la boucle, nous avons $D(a, b) = D(u, v)$. En effet :
- Avant d'entrer pour la toute première fois dans la boucle, cette égalité est évidente car u a la même valeur que a (affectation de la ligne 1) et v a la même valeur que b (affectation de la ligne 2). Les valeurs coïncidant, leurs ensembles de diviseurs communs coïncident également.
- Après chaque passage dans la boucle, le couple de variables (u, v) va prendre les valeurs $(v, u - q.v)$.
  Montrons que $D(u, v) = D(v, u - q.v)$.
  - ⋆ Soit $\alpha \in \mathbb{N}_0$ tel que α divise u et v. Dire que α divise u et v signifie qu'il existe l et $k \in \mathbb{N}_0$ tels que $u = l.\alpha$ et $v = k.\alpha$.
    De ce fait, α divise v et $u - q.v$. Nous avons bien que α divise $u - q.v$ car $u - q.v = l.\alpha - q.k.\alpha = (l - q.k).\alpha$.
  - ⋆ Réciproquement, si $\beta \in \mathbb{N}_0$ divise v et $u - q.v$, alors β divise également u (car $u = u - q.v + q.v$) et v.

  Nous avons donc montré l'égalité $D(u, v) = D(v, u - q.v)$. Or, nous avons les affectations $u \leftarrow v$ et $v \leftarrow u - q.v$. De ce fait, nous avons bien l'égalité $D(a, b) = D(u, v)$ après chaque passage dans la boucle.

Au moment où nous sortons de la boucle, la condition $v \neq 0$ n'est plus vérifiée et nous avons donc $v = 0$. Par l'égalité trouvée juste avant, nous avons alors l'égalité suivante : $D(a, b) = D(u, 0)$.
Or, $D(u, 0)$ possède un plus grand élément qui est u car le plus grand diviseur commun de u et de 0 est u. De ce fait, le plus grand commun diviseur de a et de b est u. C'est exactement ce que nous renvoie l'algorithme.
Nous venons donc de prouver la correction de cet algorithme.

C.4 Complexité

La complexité est une notion propre à l'algorithmique. Elle sert à estimer l'efficacité d'un algorithme ou à comparer différents algorithmes résolvant le même problème.
Il existe deux types de complexités : la complexité en mémoire (pour le

stockage des données) et la complexité en temps de calcul (temps d'exécution de l'algorithme).
La complexité que nous allons étudier est la complexité en temps de calcul car elle est plus intéressante pour les mathématiciens (la complexité en mémoire relevant, quant à elle, davantage du domaine de l'informatique).

Le temps d'exécution d'un algorithme peut être évalué au moyen d'un ordinateur. En implémentant un programme construit à partir de l'algorithme, l'ordinateur peut calculer le temps qu'a mis le programme pour s'exécuter. Cependant, cette méthode ne nous donne pas le temps réel d'exécution d'un programme car le temps dépend de la vitesse de l'ordinateur utilisé (le nombre d'instructions qu'il sait exécuter par seconde), du compilateur utilisé, du langage de programmation dans lequel l'algorithme a été traduit,... Il est donc plus commode d'évaluer la complexité d'un algorithme de manière théorique pour avoir une évaluation du temps de calcul indépendante de l'ordinateur, du compilateur,...

Pour déterminer la complexité en temps de calcul, nous devons supposer que toute opération élémentaire (affectation, addition,...) s'exécute en une unité de temps. Le calcul de la complexité se fait donc en déterminant le nombre d'opérations élémentaires à exécuter.

À titre d'exemple, nous allons déterminer la complexité de deux algorithmes permettant d'évaluer un polynôme en une valeur donnée.

Grâce à cette étude de la complexité, nous comprendrons l'intérêt d'introduire l'algorithme d'Hörner pour résoudre ce problème, alors qu'un autre algorithme, auquel nous sommes plus accoutumés, existe pour résoudre ce même problème.

Le premier algorithme, l'**Algorithme 11**, comporte des affectations (lignes 1, 3 et 4 ; lignes 1 et 3 de l'**Algorithme 9**), des multiplications (ligne 3 ; ligne 3 de l'**Algorithme 9**) et des additions (ligne 4). Pour déterminer la complexité de cet algorithme, nous allons donc compter le nombre d'affectations, de multiplications et d'additions. Ces nombres dépendront bien évidemment de n. La complexité est alors notée $T(n)$.

De même, le deuxième algorithme, l'**Algorithme 12**, comporte des affectations (lignes 1, 3 et 4), des multiplications (ligne 3) et des additions (ligne 4). Une fois encore, ces nombres dépendront de la valeur de n.

Algorithme 11

Entrée : n, un naturel ; a_0, a_1,...,a_n, nombres réels ; t, un réel.
Sortie : la valeur du polynôme $a_0 + a_1.x + ... + a_n.x^n$ évalué en $x = t$.

1. $c \leftarrow 0$
2. **pour tout** i allant de 0 à n **faire**
3. $\quad b \leftarrow a_i \times t^i$
4. $\quad c \leftarrow c + b$
5. **fin pour tout**
6. **retourner** c

Remarque : pour calculer t^i, on utilise l'**Algorithme 9**.

Cet algorithme effectue le calcul suivant :

$$c = a_0 + a_1.t + ... + a_{n-1}.t^{n-1} + a_n.t^n.$$

Algorithme 12

Entrée : n, un naturel ; a_0, a_1,...,a_n, nombres réels ; t, un réel.
Sortie : la valeur du polynôme $a_0 + a_1.x + ... + a_n.x^n$ évalué en $x = t$.

1. $c \leftarrow a_n$
2. **pour tout** i allant de 0 à $n - 1$ **faire**
3. $\quad b \leftarrow c \times t$
4. $\quad c \leftarrow b + a_{n-i-1}$
5. **fin pour tout**
6. **retourner** c

Cet algorithme effectue le calcul suivant :

$$c = \Big(\big((a_n.t + a_{n-1}).t + a_{n-2}\big).t + ... + a_1\Big).t + a_0.$$

Il s'agit de l'**algorithme d'Hörner**.

Complexité de l'**Algorithme 11** :

Remarque : n'oubliez pas que t^i est obtenu à partir d'un autre algorithme !

– Nombre d'affectations :

	1	*ligne 1*
$+$	$n+1$	*ligne 3*
$+$	$n+1$	*ligne 4*
$+$	$n+1$	*ligne 1* ***Algo 9***
$+$	0	*ligne 3* ***Algo 9***, $i=0$
$+$	1	$i=1$
$+$	$...$	
$+$	n	$i=n$
$=$	$3n+4+(0+1+2+...+n)$	
$=$	$3n+4+\frac{n.(n+1)}{2}$	
$=$	$\frac{n^2}{2}+\frac{7n}{2}+4$	

– Nombre de multiplications :

	$n+1$	*ligne 3*
$+$	0	*ligne 3* ***Algo 9***, $i=0$
$+$	1	$i=1$
$+$	$...$	
$+$	n	$i=n$
$=$	$(n+1)+(0+1+2+...+n)$	
$=$	$n+1+\frac{n.(n+1)}{2}$	
$=$	$\frac{n^2}{2}+\frac{3n}{2}+1$	

– Nombre d'additions : $n+1$ (*ligne 4*)

– Nombre total d'opérations : $T(n) = \frac{n^2}{2} + \frac{7n}{2} + 4 + \frac{n^2}{2} + \frac{3n}{2} + 1 + n + 1$

$$= n^2 + 6n + 6$$

Complexité de l'**Algorithme 12** :

– Nombre d'affectations :

	1	*ligne 1*
+	n	*ligne 3*
+	n	*ligne 4*
=	$2n+1$	

– Nombre de multiplications : n *(ligne 3)*

– Nombre d'additions : n *(ligne 4)*

– Nombre total d'opérations : $T(n) = 2n + 1 + n + n$

$$= 4n + 1$$

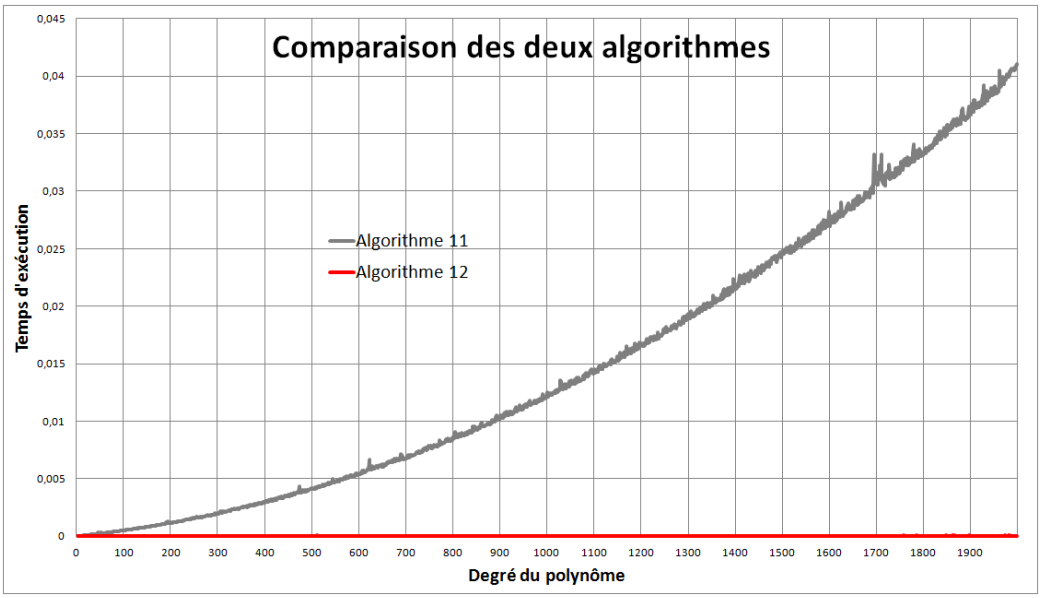
Pour une étude de la complexité, ce qui est intéressant, c'est le coût des algorithmes lorsque la taille des données (n dans les deux exemples) devient de plus en pus grande.

Les complexités obtenues avec les deux exemples sont des approximations car nous n'avons pas tenu compte, par exemple, du fait que la boucle « **pour tout** » assigne, à chaque nouveau passage dans la boucle, une nouvelle valeur à la variable i. Ceci dit, ce qui nous importe, c'est la croissance de $T(n)$ et non les constantes.
Comme nous nous intéressons au coût d'un algorithme pour une taille de

données de plus en plus grande, c'est le terme de plus haut degré de $T(n)$ qui nous intéresse (car lorsque n est grand, les termes de plus hauts degrés l'emportent sur ceux de degrés moins élevés).

Par exemple, pour l'**Algorithme 11**, le terme de plus haut degré de $T(n)$ est n^2. La complexité est donc **de l'ordre de** n^2, ce qui signifie que, si nous représentons le graphique du temps mis pour l'exécution de l'algorithme en fonction de la taille des données (n), le graphique sera une parabole.
De même, pour l'**Algorithme 12**, le terme de plus haut degré de $T(n)$ est n. Sa complexité est donc **de l'ordre de** n. En représentant le temps d'exécution par rapport à la taille des données, nous devrions obtenir une droite.

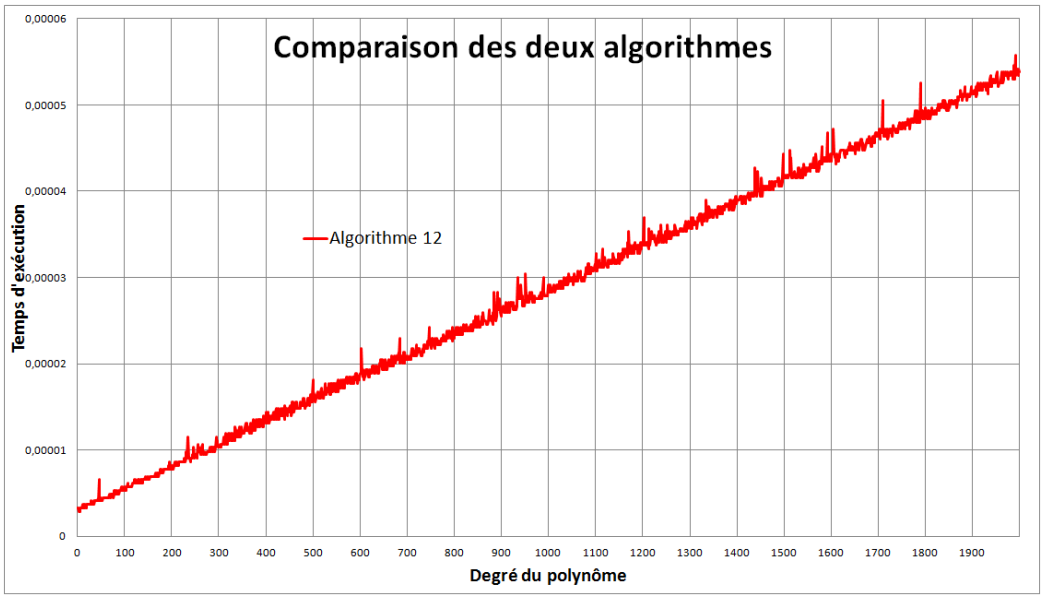
Au moyen d'un ordinateur, les deux algorithmes ont été exécutés et les temps d'exécutions en fonction de la taille des données sont repris sur le graphique suivant.



Comme nous pouvions nous y attendre, le graphique du temps en fonction de la taille des données pour l'**Algorithme 11** décrit une parabole. Celui pour l'**Algorithme 12** est bien linéaire.

Nous constatons également que l'**Algorithme 12** est plus rapide que l'**Algorithme 11** pour une taille de données tendant vers l'infini (car le graphique correspondant à la complexité de l'**Algorithme 12** se trouve en

dessous de celui correspondant à la complexité de l'**Algorithme 11**).
Nous disons que l'**Algorithme 12** a une **meilleure complexité** que l'**Algorithme 11**.



Comme nous avons pu le constater, pour un problème donné, il existe plusieurs algorithmes permettant de le résoudre.
Souvent, lorsque nous avons besoin de travailler avec une taille de données assez grande, nous aurons tendance à privilégier un algorithme avec une meilleure complexité car il s'exécute plus rapidement qu'un autre. Cependant, en fonction des données, il se peut que l'algorithme avec la meilleure complexité théorique ne soit pas le plus rapide et il sera alors préférable d'utiliser un autre algorithme.

C.5 Le problème du site de rencontres

Autant d'hommes que de femmes se sont inscrits sur un site de rencontres. Le gestionnaire de ce site doit former des couples en tenant compte des préférences de chaque personne. Pour ce faire, chaque personne a établi un classement de ses préférences. Dans ce classement, nous retrouvons toutes les personnes de l'autre sexe classées par ordre de préférence (de la préférée à la moins désirée).

C.5.1 Premier exemple

Dans un premier temps, le nombre d'inscrits sur le site est de six personnes : trois hommes (Alain, Benjamin et Cyrille) et trois femmes (Delphine, Éline et Florine).
Chaque personne a établi son classement de préférences parmi les personnes du sexe opposé.

Ci-dessous se trouvent les tableaux de préférences pour les trois hommes et pour les trois femmes (dans les tableaux, les noms sont donnés par leurs initiales) :

Alain	D	E	F
Benjamin	E	D	F
Cyrille	D	E	F

Delphine	B	A	C
Éline	A	B	C
Florine	A	B	C

Ces tableaux se lisent horizontalement. Pour chaque personne, horizontalement se trouvent toutes les personnes du sexe opposé. La première personne sur sa ligne est la personne avec laquelle elle désirerait être en priorité. La dernière personne sur sa ligne est la personne avec laquelle elle a le moins envie d'être.

Par exemple, ces tableaux nous indiquent que Cyrille aimerait par dessus tout être avec Delphine, ensuite avec Éline et finalement, avec Florine.
De même, l'homme préféré de Delphine est Benjamin et celui avec lequel elle a le moins envie d'être est Cyrille.

Dès lors, comment former les trois couples de façon à respecter les préférences de chaque individu ?

Plusieurs critères sont envisageables pour former les couples.
Par exemple, nous pourrions former les couples en essayant de maximiser le nombre de personnes mises en couple avec leurs premiers choix, ou encore en essayant de minimiser le nombre de personnes mises avec leurs derniers choix.
Il est possible d'imaginer d'autres critères, mais le critère que nous allons

choisir est celui de la **stabilité**.

Un ensemble de couples est dit *instable* s'il existe un homme et une femme qui préfèreraient être ensemble plutôt que d'être chacun avec la personne qui lui a été attribuée.
Par exemple, l'ensemble de couples {(Alain, Florine), (Benjamin, Éline), (Cyrille, Delphine)} est instable car Alain préfère Éline à Florine et Éline préfère Alain à Benjamin (Alain et Éline préféreraient être ensemble plutôt que d'être avec les personnes qui leur ont été attribuées).

Il est possible de montrer qu'il existe au moins un ensemble de couples stable [1].
L'algorithme suivant permet de retourner un ensemble stable de couples :

Algorithme 13

Entrée : n, nombre naturel non nul et $2n$ listes de préférences.
Sortie : un ensemble de n couples stable.

1. **tant qu'**il reste au moins un homme seul **faire**
2. Chaque homme se propose à la $1^{\text{ère}}$ femme non barrée de sa liste.
3. Chaque femme dit « peut-être » à l'homme qu'elle préfère parmi ceux qui se sont proposés à elle.
4. Chaque homme rejeté barre de sa liste la femme à laquelle il s'est proposé.
5. **fin tant que**
6. **retourner** couples

Appliquons donc cet algorithme à notre exemple afin d'obtenir un ensemble de couples stable.

1. Papadimitriou & Vazirani (2006). *Stable Marriage - Application of Proof Techniques for Algorithmic Analysis.*

⋆ Première étape : Tous sont seuls.

2. A se propose à D.
   B se propose à E.
   C se propose à D.

3. D dit « peut-être » à A
   (elle préfère A à C).
   E dit « peut-être » à B.
   F ne dit rien.

4. C barre D de sa liste.

⋆ Deuxième étape : C est seul.

2. A se propose à D.
   B se propose à E.
   C se propose à E.

3. D dit « peut-être » à A.
   E dit « peut-être » à B.
   (elle préfère B à C).
   F ne dit rien.

4. C barre E de sa liste.

⋆ Troisième étape : C est seul.

2. A se propose à D.
   B se propose à E.
   C se propose à F.

3. D dit « peut-être » à A.
   E dit « peut-être » à B.
   F dit « peut-être » à C.

4. Aucun homme n'est rejeté.

⋆ Quatrième étape :
aucun homme n'est seul.

Les couples sont retournés : $\Big\{$(Alain, Delphine), (Benjamin, Éline), (Cyrille, Florine)$\Big\}$.

Nous pouvons vérifier qu'il s'agit bien d'un ensemble de couples stable.

C.5.2 Deuxième exemple

Maintenant, le nombre d'inscrits sur le site de rencontres est passé de 6 à 8 personnes : 4 hommes (Kevin, Luc, Marc et Nicolas) et 4 femmes (Gaëlle, Hélène, Isabelle et Jeanine). Comme précédemment, chaque personne a établi son classement de préférences pour les personnes du sexe opposé.

Les préférences de chaque individu sont reprises dans les deux tableaux ci-dessous (à nouveau, les prénoms sont désignés par leurs initiales).

Kevin	G	I	H	J
Luc	G	H	I	J
Marc	H	G	I	J
Nicolas	I	H	G	J

Gaëlle	N	K	L	M
Hélène	K	L	M	N
Isabelle	L	M	N	K
Jeanine	N	L	M	K

Ici aussi, nous désirons former un ensemble stable de couples. Pour ce faire, appliquons une fois de plus l'**Algorithme 13**.

⋆ Première étape : Tous sont seuls.

2. K se propose à G.
   L se propose à G.
   M se propose à H.
   N se propose à I.

3. G dit « peut-être » à K
   (elle préfère K à L).
   H dit « peut-être » à M.
   I dit « peut-être » à N.
   J ne dit rien.

4. L barre G de sa liste.

⋆ Deuxième étape : L est seul.

2. K se propose à G.
   L se propose à H.
   M se propose à H.
   N se propose à I.

3. G dit « peut-être » à K.
   H dit « peut-être » à L
   (elle préfère L à M).
   I dit « peut-être » à N.
   J ne dit rien.

4. M barre H de sa liste.

⋆ Troisième étape : M est seul.

2. K se propose à G.
   L se propose à H.
   M se propose à G.
   N se propose à I.

⋆ Quatrième étape : M est seul.

2. K se propose à G.
   L se propose à H.
   M se propose à I.
   N se propose à I.

3. G dit « peut-être » à K.
   H dit « peut-être » à L.
   I dit « peut-être » à N.
   J ne dit rien.

4. M barre G de sa liste.

3. G dit « peut-être » à K.
   H dit « peut-être » à L.
   I dit « peut-être » à M.
   J ne dit rien.

4. N barre I de sa liste.

⋆ Cinquième étape : N est seul.

2. K se propose à G.
   L se propose à H.
   M se propose à I.
   N se propose à H.

3. G dit « peut-être » à K.
   H dit « peut-être » à L.
   I dit « peut-être » à M.
   J ne dit rien.

4. N barre H de sa liste.

⋆ Sixième étape : N est seul.

2. K se propose à G.
   L se propose à H.
   M se propose à I.
   N se propose à G.

3. G dit « peut-être » à N.
   H dit « peut-être » à L.
   I dit « peut-être » à M.
   J ne dit rien.

4. K barre G de sa liste.

⋆ Septième étape : K est seul.

2. K se propose à I.
   L se propose à H.
   M se propose à I.
   N se propose à G.

3. G dit « peut-être » à N.
   H dit « peut-être » à L.
   I dit « peut-être » à M.
   J ne dit rien.

4. K barre I de sa liste.

⋆ Huitième étape : K est seul.

2. K se propose à H.
   L se propose à H.
   M se propose à I.
   N se propose à G.

3. G dit « peut-être » à N.
   H dit « peut-être » à K.
   I dit « peut-être » à M.
   J ne dit rien.

4. L barre H de sa liste.

⋆ Neuvième étape : L est seul.

2. K se propose à H.
   L se propose à I.
   M se propose à I.
   N se propose à G.

3. G dit « peut-être » à N.
   H dit « peut-être » à K.
   I dit « peut-être » à L.
   J ne dit rien.

4. M barre I de sa liste.

⋆ Dixième étape : M est seul.

2. K se propose à H.
   L se propose à I.
   M se propose à J.
   N se propose à G.

3. G dit « peut-être » à N.
   H dit « peut-être » à K.
   I dit « peut-être » à L.
   J dit « peut-être » à M.

4. Aucun homme n'est rejeté.

⋆ Onzième étape : aucun homme n'est seul.

Les couples sont retournés : $\Big\{$(Kevin, Hélène), (Luc, Isabelle), (Marc, Jeanine), (Nicolas, Gaëlle)$\Big\}$.

Une fois encore, nous pouvons montrer qu'il s'agit bien d'un ensemble de couples stable.

C.5.3 Terminaison

Pour que l'**Algorithme 13** soit bel et bien un algorithme, il faut vérifier que le nombre d'étapes exécutées est fini.
Les étapes dans la boucle sont en nombre fini car la ligne 2 est exécutée n fois (il y a n hommes), la ligne 3 est exécutée au maximum n fois (dans le cas où toutes les femmes auraient à parler) et la ligne 4 est exécutée au maximum n fois (car, dans le pire des cas, tous les hommes sont rejetés).
Il faut donc vérifier que la boucle se termine, c'est-à-dire vérifier que tous les hommes sont avec une femme à un moment donné.

Supposons donc, par l'absurde, qu'un homme H ait barré toutes les femmes de sa liste. Comme il y a le même nombre d'hommes que de femmes, si H est seul, il doit également rester une femme F seule. Comme H a barré toutes les femmes de sa liste, il a également barré F après s'être proposé à elle. S'il

a barré F, cela signifie qu'elle lui a préféré un autre homme. Ce qui crée une contradiction puisqu'alors, F ne devrait pas être seule.
La terminaison est donc prouvée.

Déterminons le nombre maximal de passages dans la boucle. À chaque passage dans la boucle - sauf le dernier -, au moins un homme barre une femme de sa liste (car si aucun ne devait barrer de femme, cela voudrait dire qu'aucun n'était seul).
Comme il y a n hommes et n femmes et que chaque homme a le choix des n femmes, le nombre maximal de passages dans la boucle est $n \times n = n^2$.

C.6 Conclusion

Lors de cette introduction à l'algorithmique, nous avons exécuté plusieurs algorithmes, nous en avons prouvé certains et nous avons fait des études de la complexité.
Au final, nous n'avons pas écrit beaucoup d'algorithmes nous-mêmes, nous avons plutôt essayé de les comprendre.

Il existe des cours d'algorithmique dans l'enseignement supérieur, dispensés notamment dans des études de mathématiques, d'informatique, de physique, d'ingénieur civil,...

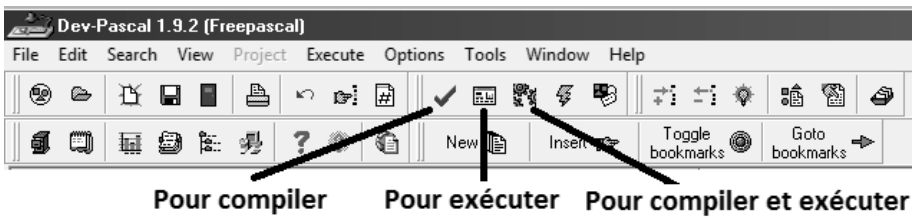
Annexe D

Séquence d'enseignement : introduction à la programmation

Durant la dernière heure de notre séquence d'enseignement, nous proposons aux élèves une introduction à la programmation. Pour cela, nous leur présentons quatre langages de programmation : le « Pascal », le « C », le « Python » et le « Matlab ». Nous leur donnons des logiciels permettant de programmer dans ces différents langages.
Nous traduisons les algorithmes 5, 6 et 7 de notre séquence de cours dans ces langages afin de les illustrer.

D.1 Le langage « Pascal »

Un logiciel pour programmer en Pascal : *Dev-Pascal*.



Algorithme 5

```
program algo5;

var
x,absolu,a:real;

begin
writeln('Donnez un reel, nous en donnerons la valeur absolue.');
readln(x);

if x>=0 then
   absolu:=x
else
   absolu:=-x;

writeln('La valeur absolue de ',x,'est ',absolu,'.');
readln(a);
end.
```

Algorithme 6

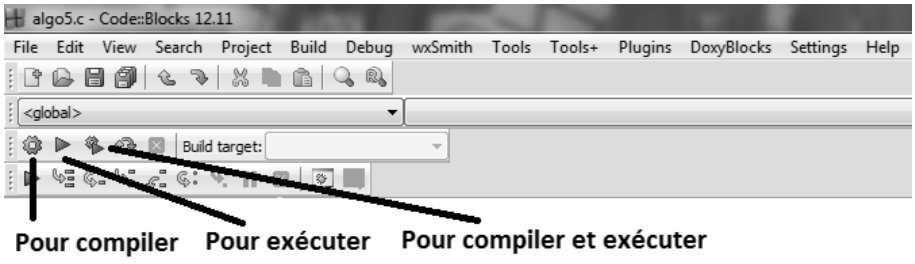
```
program algo6;

var
n,i,somme,a:integer;

begin
writeln('Donnez un naturel n, nous donnerons la somme de 1 a n');
readln(n);

somme:=0;
for i:=1 to n do
    somme:=somme+i;

writeln('La somme de 1 jusque ',n,' vaut ',somme,'.');
readln(a);
end.
```

Algorithme 7

```
program algo7;

var
n,i,somme,a:integer;

begin
writeln('Donnez un naturel n, nous donnerons la somme de 1 a n.');
readln(n);

somme:=0;
i:=1;
while i<>n+1 do
     begin
     somme:=somme+i;
     i:=i+1;
     end;

writeln('La somme de 1 jusque ',n,' vaut ',somme,'.');
readln(a);
end.
```

D.2 Le langage « C »

Un logiciel pour programmer en C : *Code : :Blocks*.



Algorithme 5

```
#include <stdio.h>

float x;
float absolu;

int main()
{
    printf("Donnez un reel.");
    printf("Nous en donnerons la valeur absolue. \n");
    scanf("%f",&x);
    if (x>=0){
        absolu=x;
        }
    else{
        absolu=-x;
        }
    printf("La valeur absolue de %f est %f.\n", x,absolu);
}
```

Algorithme 6

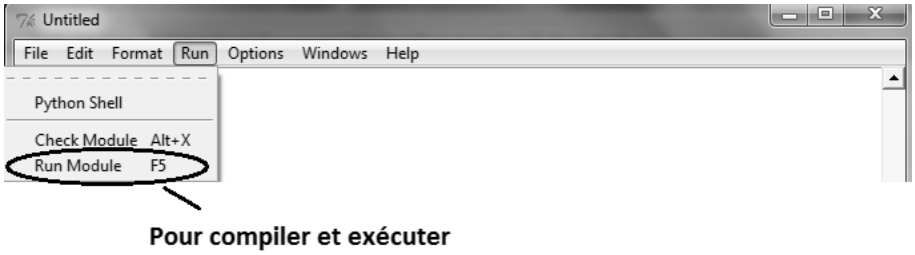
```
#include <stdio.h>

int n,somme,i;

int main()
{
    somme=0;
    printf("Donnez n, un naturel.");
    printf("Nous donnerons la somme de 1 a n. \n");
    scanf("%d",&n);
    for (i=1;i<=n;i++){
    somme=somme+i;
    }
    printf("La somme de 1 jusque %d vaut %d. \n",n,somme);
}
```

Algorithme 7

```
#include <stdio.h>

int n,somme,i;

int main()
{
    somme=0;
    i=1;
    printf("Donnez n, un naturel.");
    printf("Nous donnerons la somme de 1 a n. \n");
    printf("\n");
    scanf("%d",&n);
    while(i<=n){
    somme=somme+i;
    i++;
    }
    printf("La somme de 1 jusque %d vaut %d. \n",n,somme);
}
```

D.3 Le langage « Python »

Pour installer Python, aller sur le site *www.python.org* et installer une version commençant par 2.7.
Pour écrire un code, ouvrir l'IDLE de Python et appuyer sur F5 pour compiler et exécuter le code.



Algorithme 5

```
x=input('Donnez un reel, nous en donnerons la valeur absolue. \n')
if x>=0:
    absolu=x
else:
    absolu=-x
print 'La valeur absolu de', x,' est', absolu,'.'
```

Algorithme 6

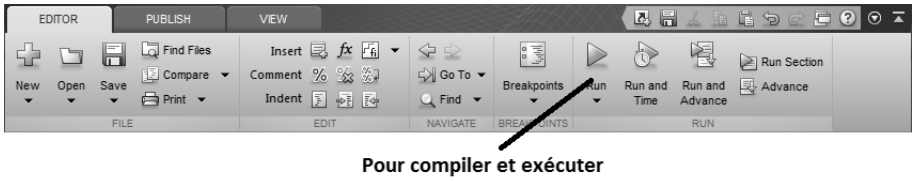
```
somme=0
n=input('Donnez n, naturel, nous donnerons la somme de 1 à n. \n')

for i in range(1,n+1):
    somme=somme+i

print 'La somme de 1 jusque', n,' vaut', somme,'.'
```

Algorithme 7

```
somme=0
i=1
n=input('Donnez n, naturel, nous donnerons la somme de 1 à n. \n')

while i<=n:
    somme=somme+i
    i=i+1

print 'La somme de 1 jusque', n,' vaut', somme,'.'
```

D.4 Le langage « Matlab »

Matlab est un logiciel payant.



Algorithme 5

```
clear all
clc

x=input('Donnez un reel, nous en donnerons la valeur absolue.');
if x>=0
    absolu=x;
else
    absolu=-x;
end;
fprintf('La valeur absolue de %f est %f. \n \n', x,absolu)
```

Algorithme 6

```
clear all
clc

somme=0;
n=input('Donnez n, naturel, nous donnerons la somme de 1 a n.');

for i=1:1:n
    somme=somme+i;
end;

fprintf('La somme de 1 jusque %d vaut %d. \n \n',n,somme);
```

Algorithme 7

```
clear all
clc

somme=0;
i=1;
n=input('Donnez n, naturel, nous donnerons la somme de 1 a n.');

while i<=n
    somme=somme+i;
    i=i+1;
end;

fprintf('La somme de 1 jusque %d vaut %d. \n \n',n,somme);
```