A1 - Introduction à l'algorithmique

I Introduction

Motivation : `https://www.hbrfrance.fr/chroniques-experts/2017/07/16236-entreprises-doivent-developper-algorithmes-ethiques/`

I.1 Définitions

I.1.1 Informatique

Source : `http://www.enib.fr/~tisseau/pdf/course/info-S1.pdf`

Le terme informatique est un néologisme proposé en 1962 par Philippe Dreyfus pour caractériser le traitement automatique de l'information : il est construit sur la contraction de l'expression «information automatique». Ce terme a été accepté par l'Académie française en avril 1966, et l'informatique devint alors officiellement la science du traitement automatique de l'information, où l'information est considérée comme le support des connaissances humaines et des communications dans les domaines techniques, économiques et sociaux (figure 1.1). Le mot informatique n'a pas vraiment d'équivalent en anglais. Aux Etats-Unis où l'on parle de *Computing Science* (science du calcul, souvent traduit inexactement en *science des ordinateurs*. En effet, *compute* veut dire calculer, et *computer* se traduit par calculateur ou ordinateur selon le contexte) alors que le terme *Informatics* est acceptés par les Britanniques.

> Définition I.1 — L'informatique est la science du traitement automatique de l'information.

I.1.2 Algorithmique

Plusieurs définitions du terme algorithme existent. Ce ne sera pas important pour nous de rentrer dans le débat de savoir quelle définition est la meilleure. Je donne ici la définition de Cormen et al. (*Introduction à l'algorithmique*[1]).

> Définition I.2 — Un algorithme est une procédure de calcul bien définie qui prend en entrée une valeur, ou un ensemble de valeurs, et qui donne en sortie une valeur, ou un ensemble de valeurs. Un algorithme est donc une séquence d'étapes de calcul qui transforment l'entrée en sortie.

En pratique :

- J'ai un problème (de calcul) bien spécifié.
- Ce problème définie une relation entre l'entrée et la sortie.
- L'algorithme décrit une procédure de calcul spécifique permettant d'obtenir cette relation entrée/sortie.

Exemple I — Extrait d'un dialogue entre un touriste égaré et un autochtone.

- *Pourriez-vous m'indiquer le chemin de la gare, s'il vous plaît ?*
- *Oui bien sûr : vous allez tout droit jusqu'au prochain carrefour, vous prenez à gauche au carrefour et ensuite la troisième à droite, et vous verrez la gare juste en face de vous.*
- *Merci.*

La réponse de l'autochtone est une suite d'instruction qui permet au touriste de résoudre son problème (trouver la gare). C'est donc un algorithme.

Exemple II — Sur booking.com, je cherche les hôtels à Annecy pour le nouvel an. Il m'affiche une liste d'hôtel disponible. Je veux les afficher par ordre croissant de prix. On va donc utiliser un algorithme de tri.

- Entrée : une liste de n nombres réels.
- Sortie : la liste ordonnée par ordre croissant.

1. Bouquin très largement utilisé dans l'enseignement *undergraduate* (Licence) aux Etats-Unis

Appelons l la liste d'entrée (non triée) et A l'algorithme. Alors on peut noter $A(l)$ la sortie de l'algorithme : c'est la liste triée dans l'ordre croissant.

Par exemple, si la liste d'entrée est $l =< 80; 65; 125; 75; 90 >$, alors $A(l) =< 65; 75; 80; 90; 125 >$.

Définition I.3 — Un algorithme est dit *correct* si, pour chaque instance en entrée, il se termine en produisant la bonne sortie. L'on dit qu'un algorithme correct *résout* le problème donné.

- Un algorithme *incorrect* risque de ne pas se terminer pour certaines instances en entrée, voire de se terminer sur une réponse autre que celle désirée.
- Contrairement à ce que l'on pourrait croire, un algorithme incorrect peut s'avérer utile dans certains cas, si son taux d'erreur est susceptible d'être contrôlé. Néanmoins en seconde, on ne s'intéressera uniquement aux algorithmes corrects.

I.2 Des exemples de problèmes qui peuvent être résolus par des algorithmes

Exemple III — Dans la vie de tous les jours :

- Une recette de cuisine est un algorithme (beaucoup de professeurs utilisent cet exemple archi-classique. Néanmoins, je pense que c'est un très mauvais exemple, car la suite des opérations peut ne pas être respectée à la lettre : la cuisson d'un aliment n'est jamais minutée à la seconde près, de même que la quantité d'eau pour faire bouillir les pâtes est approximative).

En maths :

- L'algorithme d'Euclide (ou l'algo des soustractions successives) pour le calcul du PGCD de deux nombres.
- Le théorème de Pythagore permet de calculer en un nombre fini de calculs la longueur de l'hypoténuse à partir des longueurs des deux autres côtés. Ou permet de dire si un triangle est rectangle ou non en connaissant les longueurs des trois côtés.

Exemple IV — Donnons quelques exemples en vrac d'algorithmes.

- Correcteur orthographique :
  - Entrée : texte (tapé dans Word).
  - Sortie : souligne les mots mal orthographiés
  - Pour détecter les fautes d'orthographe, il se base sur les règle grammaticales.
- Google Translate
  - Entrées : texte à traduire, langue de départ, langue d'arrivée.
  - Sortie : texte traduit.
  - Pour traduire le texte, l'algorithme se base sur un dictionnaire de langue (entre autre). Il est intéressant de noter que le texte traduit est souvent approximatif : l'algorithme est *incorrect* (mais quand même extrêmement utile).
- Connexion à Facebook
  - Entrées : adresse e-mail, mot de passe
  - Sortie : Vrai/Faux (Vrai = connexion réussie, Faux = connexion ratée)
  - Pour décider, l'algorithme vérifie si le mot de passe donné correspond à celui de la base de donnée.
- Séquençage du génome humain.
- Internet (moteur de recherche, auto-complétion, commerce électronique).
- Reconnaissance vocale.
- Calcul du PGCD
  - Entrée : deux nombres entiers x et y.
  - Sortie : PGCD(a, b)
  - Pour le calcul, on utilise l'algorithme d'Euclide, ou l'algorithme des soustractions successives.
- Déterminer si ABC est un triangle rectangle
  - Entrée : les trois longueurs AB, AC, BC
  - Sortie : Oui/Non
  - On utilise le théorème de Pythagore.
- Déterminer la longueur de l'hypoténuse
  - Entrée : les deux longueurs AB et AC
  - Sortie : longueur de l'hypoténuse BC
  - Pour la calculer, on utilise le théorème de Pythagore.

I.3 Un algorithme, ça ressemble à quoi ?

Sur la planète Mars, des astronautes ont découvert l'existence de créatures vivantes. Ces créatures sont très douées en calcul. On a donc décidé de les ramener sur Terre pour pour utiliser leur capacité de calcul. Bon, vu qu'on est des êtres humains, on a mis ces créatures en esclavage, c'était plus simple. Deux problèmes se posent à nous :

— Ces créatures ne parlent aucun de nos langages, mais plutôt une langue super bizarre avec des 0 et des 1. A part von Neumann, aucun être humain ne peut parler le langage des créatures, et aucune créature ne peut comprendre les êtres humains.

— Les créatures sont des esclaves récalcitrants. Elles feront tout pour ne pas obéir aux ordres.

Pour résoudre ces deux problèmes, on a deux solutions :

— D'abord, on a essayé de traduire le français vers le binaire. Mais cela donnait de très mauvais résultats, car les créatures ne comprennent que des ordres simples, et la langue française est bien trop complexe.

— Donc on utilise ce que l'on appelle un *langage de programmation*. C'est un langage entre le français et le binaire, mais qu'un être humain peut comprendre. Ensuite, ce langage est traduit en binaire et la créature peut le comprendre (on parle de *compilation* ou d'*interprétation*).

Le gros problème, c'est que les êtres humains ne se sont pas mis d'accord sur un langage de programmation unique : il en existe plein ! Certains sont très proches du binaire (on parle de *langage de faible niveau*, comme le LISP, Assembly, et dans une moindre mesure C/C++), d'autres plus éloignés (*langages de haut niveau*, comme le Python).

Nous utiliserons trois langages :

— Le langage naturel (sorte mixe entre le français et un langage de programmation) ; mais aucune créature ne peut le comprendre, donc nous ne l'utiliserons qu'au début pour que la transition soit plus facile.

— Le langage de la calculatrice (TI). Comme son nom l'indique, il nous sera utile pour programmer à la calculatrice. A noter que Casio et TI ont deux langages différents (ça serait trop simple sinon...)

— Le langage Python : un vrai langage (vrai dans le sens vraiment utilisé massivement dans l'industrie, contrairement à Scratch ou Algobox qui ne sont utilisés qu'au collège/lycée, et nul part ailleurs) de haut niveau (donc proche de l'anglais). Instagram, Pinterest ou Mozilla utilisent un framework Python (Django en l'occurrence).

I.4 Algorithmes de calcul simples

Exemple V — Considérons la fonction $f(x) = x^2 + 3$. On veut écrire un algorithme qui, partant d'un nombre réel x, calcule $f(x)$.

Regardons étape par étape.

Langage naturel	Langage de la TI	Python
Saisir A	Input A	def f(x)
Afficher $A^2 + 3$	Disp $A^2 + 3$	return $x^2 + 3$

Exercice I — Considérons maintenant la fonction $g(x) = 2x^2 + 3x + 9$. Écrire un algorithme en TI et en Python qui, en partant d'un nombre réel x, calcule $f(x)$.

Correction

Langage naturel	Langage de la TI	Python
Saisir A	Input A	def f(x)
Afficher $2 \times A^2 + 3 \times A + 9$	Disp $2 \times A^2 + 3 \times A + 9$	return $2x^2 + 3x + 9$

□

II Variable

II.1 Définition

Durant l'exécution de l'algorithme, on va avoir besoin de stocker des données, voire des résultats de calcul. Pour cela, on utilise ce que l'on appelle une variable.

> Définition II.1 — Une variable est un espace mémoire dans lequel il est possible de mettre une valeur.
> - Par exemple, si en français je dis x est égal à 1, j'utilise la variable dont le nom est x pour lui fixer la valeur 1. Pour faire la même chose en Python, je note simplement : x = 1.
> - Cette opération est appelée "affectation" (ou "assignation"), et consiste à stocker une valeur en mémoire vive de l'ordinateur. On dit donc que l'on procède à l'affectation de la variable "x" avec la valeur "1".

Une variable est donc un objet qui a un nom, derrière lequel se cache une valeur, qui peut changer (varier) au cours de l'exécution du programme.

II.1.1 Bien nommer ses variables

Pour nommer une variable, on utilisera les règles suivantes :

- A la calculatrice, c'est simple, le nom d'une variable ne peut être qu'une lettre majuscule.
- En Python :
  - Les noms de variables ne peuvent contenir que des lettres, des chiffres, et des underscore.
  - Le nom de la variable ne peut pas commencer par un chiffre.
  - Le langage Python est sensible à la casse, ce qui signifie que des lettres majuscules et minuscules ne constituent pas la même variable (la variable *maVariable* n'est pas la même que *mavariable*, qui n'est pas la même que *MAVARIABLE*).
  - Par convention, on évite d'utiliser le underscore : donc on nommera *maVariable* et non *ma_variable*
  - Certains mot clés sont réservés et ne peuvent être utilisé comme nom de variable (en Python : and, or, for, if, def, with, raise, return, break, etc... Il y en a en tout 30 ; inutile de les apprendre par coeur (pour l'instant)).

Remarque importante Il est important de bien nommer ses variables, c'est à dire *trouver un nom pour cette variable qui décrive le mieux possible ce qu'elle représente*, quitte à ce que ce nom soit un peu long.

Par exemple : longueurFeuille ; nbPiècesJaunes ; maxHauteurPiquets sont des bons noms de variables.

Au contraire, maVariable ; variableAuxilliaire ; variable1 sont des mauvais noms de variables (on ne sait pas ce qu'elles représentent).

II.1.2 Affecter une valeur

Maintenant que l'on a vu comment nommer une variable, on va voir comment lui affecter une valeur.

- A la calculatrice : $2 \rightarrow A$ veut dire que l'on affecte la valeur 2 à la variable A.
- En Python : maVariable = 2 veut dire que l'on affecte la valeur 2 à maVariable.

II.1.3 Qu'est ce qu'on peut faire avec une variable ?

Considérons le programme suivant :

```
monAge = 24
monAge = monAge +1
```

Combien vaut monAge à la fin du programme ?

Réponse : 25.

II.2 Introduire une variable auxiliaire

Pour chacun des deux algorithmes suivants, explicitez les valeurs des variables à chaque étape.

Exercice II — A prend la valeur 1
B prend la valeur A+3
C prend la valeur 2A-B
Afficher C

Correction A vaut 1, puis B vaut 3+1 = 4, et enfin C vaut 2*1-4 = -2. Donc le programme affiche 2. □

Exercice III — A prend la valeur 5
B prend la valeur A+4
A prend la valeur A+B
Afficher A

Correction A vaut 5, puis B vaut 5+4=9. Ensuite, on change la valeur de la variable A en 9+5=14. Donc à la fin, le programme affiche la dernière valeur de A, c'est à dire 14. □

Exercice IV — Un apprenti informaticien a qui on demandait d'échanger (*swap*) les valeurs de 2 variables x et y et proposa la suite d'instructions suivante :

x = y
y = x

Il eut la désagréable surprise de constater que les valeurs des variables n' étaient pas permutées après cette séquence d'affectations. Expliquer pourquoi.

Correction Prenons un exemple. Essayons l'algorithme avec $x = 2$ et $y = 3$.

Lors que la première étape ($x = y$), j'affecte à la variable x la valeur de y. Autrement dit, x devient 3 (et y ne change pas, et reste 3).

Ensuite, lors de la deuxième étape ($y = x$), j'affecte à la variable y la valeur de x. Comme x vaut 3, y devient 3.

Donc à la fin, j'ai $x = 3$ et $y = 3$. Autrement dit, je n'ai pas échangé les deux variables (l'échange aurait été effectif si $x = 3$ et $y = 2$). La raison est que j'ai perdu la valeur initiale de x ($x = 2$ au départ) lors de la première étape, et derrière je n'ai aucune manière de la récupérer. □

Exercice V — Que fait cet algorithme :

Saisir A
Saisir B
C prend la valeur B
B prend la valeur A
A prend la valeur C
Afficher A
Afficher B.

Correction Essayons l'algorithme en partant de $A = 4$ et $B = 5$ (ce sont les deux premières lignes "Saisir A" et "Saisir B").

Ensuite, on introduit une nouvelle variable C, qui prend la valeur de B. Autrement dit, C=5.

Ensuite B prend la valeur de A, donc B=4.

Puis A prend la valeur de C, donc A=5.

Finalement, on a : A=5 et B=4. Les valeurs de A et B ont bien été échangée. Cette fois, on a du passer par une variable auxiliaire C, qui a permis de stocker temporairement la valeur de B (et donc de ne pas la perdre, contrairement à l'algorithme précédent !)

II.3 Différents types de données en Python

Jusqu'ici, nous n'avons travaillé uniquement avec des nombres (et pour cause, on ne fait rarement un programme sans aucun nombre !). Néanmoins, on peut être amené à travailler avec autre chose (par exemple des mots, des phrases). De plus, il existe plusieurs sortes de nombres (entiers, à virgule). En classe de Seconde, on rencontrera les types de données suivants :

- Les nombres entiers (se dit *integer* en anglais, que l'on abrège en *int*) : ce sont les éléments de $\mathbb{Z}$ (entiers positifs ou négatifs). En Python, on ne peut stocker dans un *int* des nombres entiers compris entre -2^{31} et 2^{31} (soit entre -2 147 483 648 et 2 147 483 648). Honnêtement cela nous suffira largement.
- Les nombres à virgules, appelé nombres flottant (*float*). Sans rentrer dans les détails[2], on peut représenter les nombres allant de 10^{-324} à 10^{308}, avec environ 16 chiffres significatifs, ce qui là aussi sera suffisant pour nous.
- Les chaînes de caractère (*str*, car en anglais, chaîne se dit *string*) : c'est du texte (une phrase, un mot). On écrit entre guillemets (ou entre apostrophe, mais c'est à éviter, car vous aurez vite des problèmes si vous écrivez une phrase avec des apostrophes).

Il est important de savoir à quelle type de variable on a à faire (car si on travaille avec des *string*, le signe + n'aura pas la même signification qu'avec des nombres entiers ou flottants !).

II.4 Problèmes courants

II.4.1 Division d'entiers

Une erreur courante arrive lorsque l'on divise des entiers. Par exemple, pour Python, *4/3 = 1*. En effet, quand Python divise l'entier 4 par l'entier 3, le résultat devrait être 1,3333...., mais ce n'est pas un nombre entier : donc Python tronque et renvoie l'entier 1. Pour s'en sortir, il faut forcer Python a travailler avec des *float* :

- par exemple en faisant float(4) / float(3)
- ou bien 4.0/3.0 (pour Python, 4.0 et 3.0 ne sont pas des entiers mais des float, car il y a une virgule !)

2. Les détails sont ici : `https://fr.wikipedia.org/wiki/IEEE_754`

II.4.2 Calculs avec les nombres décimaux

L'ordinateur compte en binaire, alors que nous utilisons une base 10. Pour les calculs avec des nombres entiers, cela ne pose pas de problème (on peut passer d'une base à l'autre sans soucis). Là où cela coince, c'est pour les calculs avec des chiffres à virgule.

Exemple :

— 0.1 s'écrit en binaire : 0.0001100110011001100110011001100110011001100110011... (c'est un développement périodique avec une infinité de chiffres après la virgule)

— Comme Python ne peut pas stocker une infinité de chiffre après la virgule, il va arrondir, et dire que 0.1 en base 10 vaut 0.0001100110011001100110011001100110011001100110011010.

— Or ce nombre n'est pas exactement 0.1, mais 0.1000000000000000055511151231257827021181583404541015625 (il y a une différence à partir du 18ème chiffre après la virgule).

— Une des conséquences, c'est que en Python, le calcul 0.1+0.2 va donner 0.30000000000000004.

Dans l'énorme majorité des cas, on s'en fiche car on n'a pas besoin d'une telle précision dans nos calculs. Si on a besoin de plus de précision, on peut toujours utiliser des modules additionnels (le module *decimal* par exemple), mais les calculs s'effectuent alors environ 100 fois plus lentement.