[PROF] Introduction à l'Algorithmique

Fiche 1

Ce document a pour objectif de guider les enseignants pour le cours d'algorithmique. Il est déconseillé de le fournir aux étudiants, car il vise surtout à vous permettre de guider la séance. Vous n'êtes évidemment pas obligé de le suivre à la lettre (c'est même déconseillé, car cela va autant vous perturber vous que la classe : suivez votre chemin de pensée et/ou celui de la classe, et ensuite vérifiez que vous n'avez rien oublié).

La première séance vise à faire découvrir les fondements de l'algorithmique aux personnes n'en ayant absolument jamais faite. Les algorithmes étudiés sont donc littéralement ce qui est enseigné en mathématiques dans l'enseignement primaire ou secondaire. Aucun ordinateur n'est nécessaire : vous pouvez les interdire pendant cette séance. Il faut uniquement du papier et de quoi écrire (et cela permet de ne pas perdre la capacité d'écrire).

1 Problèmes, Solutions, et Types de données

1.1 Définition algorithme

— Demandez à la classe comment les étudiants définiraient ce qu'est un "algorithme"

— Rappelez enfin la définition suivante :

Définition informelle d'un algorithme [1] : « *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* ».

— Insistez sur les mots clés : *procédure*, *valeur d'entrée*, *valeur de sortie*, *séquence d'étapes*, *calculs*, *transformer l'entrée en sortie*

1.2 Bien spécifier le problème

— Rappeler qu'un algorithme vise à résoudre un problème (même si cela peut simplement être d'exécuter un traitement simple)

— Pour résoudre le problème, on va devoir bien analyser les choses et phénomènes pour pouvoir les représenter

— 3 questions à constamment se poser dans cet ordre précis (les mêmes qu'en recherche) :

1. **Pourquoi** ?
2. **Quoi** ?
3. **Comment** ?

Exemple : une usine de retraitement de déchets récupère des gravats et doit les séparer pour les réutiliser.

— Pourquoi ? Trier des gravats

— Quoi ? Pierres de différentes tailles

— Comment ? Tamis de différentes tailles pour séparer les pierres, les cailloux, et le gravier

1. Introduction à l'Algorithmique. 2001 (2^e édition) T.Cormen et al.

1.2.1 Pourquoi ?

— **Objectif** ou **but** à atteindre
— Souvent c'est un verbe : *Trier*, *Filtrer*, *Retrouver*, *Calculer*, ...
— Il s'agit, en général, du nom de la fonction

1.2.2 Quoi ?

— **Objets** manipulés, **qualités** observées, **quantités** mesurées
— Souvent ce sont des noms : *Personne*, *Fruits*, *Distance*, *Taille*, *État d'un système*, ...
— Il s'agit des variables et des paramètres de chaque algorithme

1.2.3 Comment ?

— Combinaison des mesures et objets précédents permettant de répondre au problème (ou formules mathématiques), ou combinaison d'outils existants
— Il s'agit de la suite d'instructions/la formule appliquée OU la/les fonction(s) d'une bibliothèque

Bien expliciter que le *pourquoi* est le plus important et permet de ne pas partir vers la mauvaise direction : lorsque l'on est embrouillé dans l'écriture d'un algo ou de code... il faut s'arrêter 2 minutes et se reposer la question :

— *Pourquoi je fais cela ?*
— *Quel problème est-ce que je cherche à régler ?*
— *Quel est l'objectif du code que j'écris ?*

1.3 Types de données

Une fois le *pourquoi* intégré, on peut passer au *quoi* et donc aux types de données :

entier	**integer**	Entiers relatifs (+ et -)	**`42`**
flottant	**float** **double**	Nombres à virgule (attention aux comparaisons)	**`13.37`**
caractère	**char**	Lettres / Caractères (un seul à la fois)	**`'b'`**
chaîne de caractères	**string**	Suite de caractères	**`"lol"`**
booléen	**bool**	Valeur booléenne (Vrai/Faux)	**`True`**

Donnez des exemples de données et demandez à quel type ils appartiennent. Rappelez que le type *bool* n'existe pas dans tous les langages et qu'il est parfois un entier : « 0 » indiquant *faux*, et « tout sauf 0 » indiquant *vrai*.

Attention : indiquez bien qu'en C et dans la plupart des langages de programmation :

— les caractères sont encadrées par des *simple quotes* (apostrophes simples) [**`'`**]
— les chaînes de caractères sont encadrées par des *double quotes* (apostrophes doubles) [**`"`**]

(La *back quote* ou *backtick* (accent grave) [**`` ` ``**] (Alt Gr + 7 sur les claviers français) est utilisée dans le développement, mais pas pour les types)

Attention : de même, les flottants s'écrivent généralement avec la notation anglaise, à savoir un *point* entre les unités et les décimales, et très rarement avec une virgule comme en français.

Demandez comment mettre **`"`** entre des doubles apostrophes ou comment manipuler le caractère **`'`**.
Expliquez l'usage du *backslash* (anti-slash) [**`\`**] : **`\'`** **`\"`** **`` \` ``** **`\n`** **`\r`** **`\r\n`** ...
Exemples : **`'\''`** **`"\"Texte\""`** ... mais : **`'"'`** **`"Tex't'e"`**

2 Exécution pas à pas

2.1 Somme des N premiers entiers

— Projettez le PDF au tableau pour n'avoir qu'à écrire dans les cases, tout en permettant à la classe de voir l'algorithme

— N'hésitez pas à créer des cases autour au feutre pour montrer les variables qui évoluent

— ...attention à ne pas écrire sur le mur avec le feutre...

— Chaque case du tableau permet de connaître l'état des variables au moment d'effectuer la condition du *while*

— « État Initial » sert juste à indiquer l'état des variable au moment d'entrer dans la boucle

— Les cases suivantes indiquent l'état des variables à la fin de chaque tour

— (La case « 1 » contient donc l'état des variables à la fin du 1er tour de boucle/juste après avoir exécuté une fois chaque instruction de la boucle)

2.2 Multiplication égyptienne

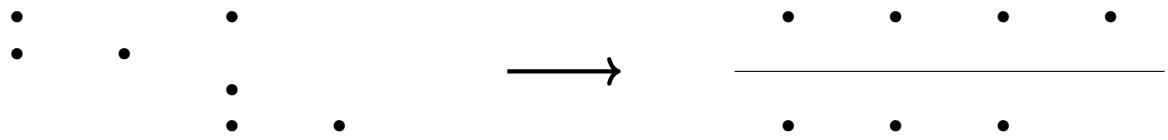
— Projetez l'algorithme au tableau, mais ne l'expliquez pas

— Laissez les étudiants exécuter l'algorithme, et demandez-leur après d'expliquer le fonctionnement

— Demandez comment cela peut être possible de savoir diviser par 2 et détecter des nombres pairs/impaires sans savoir faire de multiplication

— Une fois des solutions présentées par les étudiants, montrez cette technique au tableau :

Comment diviser 7 par 2 quand on ne sait que dénombrer ? (Voire, sans même avoir la notion de dénombrement)



1. On prend les objets deux à deux, on les aligne, et on répète jusqu'à ne plus avoir que un ou zéro objets.
2. — S'il ne reste aucun objet : on avait une quantité paire d'objets
   — S'il reste un objet : alors on avait une quantité impaire.
3. Le dividende/La moitié "entière" est le côté avec le moins de points (ici la zone du bas avec 3 objhets)

C'est un algorithme purement visuel qui ne nécessite aucune connaissance mathématique théorique, ni même de savoir lire, écrire, ou compter : on a divisé par 2 le tas d'objets sans même savoir combien il y en avait.

— L'algorithmique correspond à peu près à cela :

— Trouver une méthode pour résoudre un problème

— (et pouvoir réutiliser cette méthode si le problème est rencontré de nouveau)

— Ainsi qu'optimiser au mieux la méthode

2.3 Division euclidienne

— L'algorithme a un problème : les étudiants doivent le corriger
— C'est plus simple de corriger/débugger avec des valeurs

— Présentez les effets de > et >= sur la boucle
— Demandez comment gérer le cas 0...

3 Écriture d'algorithmes simples

3.1 Multiplication classique

— Faire écrire un premier algorithme aux étudiants
— Insister sur le fait de faire avec une addition au milieu de l'algorithme
— *Nous on sait faire la multiplication, mais comment le processeur a "appris" à le faire ?*
— Expliquez la différence entre `i = i + 1` et `i += 1` (et pour les bons : `i++` et `++i`)

— Demandez comment gérer le cas 0...
— ...Puis le cas des nombres négatifs
— Présentez le système de *fonction chapeau* (fonction filtrant les paramètres avant l'exécution de l'algorithme principal)
— Faites leur écrire une fonction chapeau filtrant le cas 0 et permettant de gérer les nombres négatifs

— Exercice de calligraphie : apprenez aux étudiants à écrire le symbole « & » (et « { » « } »)
(Quelques-uns savent le faire, mais certains bloquent jusqu'à la fin de l'année)

3.2 Puissance

— Même chose : insister sur le fait qu'il faille utiliser des multiplications, et particulièrement la fonction réalisée juste avant
— Demandez aux étudiants à quoi doit ressembler la *pile d'appels* et combien d'additions ont réellement été effectuées
— Indiquez que le calcul d'une puissance n'est pas du tout une opération facile pour un CPU : il n'y a pas toujours d'instruction POW dans les processeurs... ce sont souvent des fonctions dans les logiciels/bibliothèques, mais pas internes aux processeurs
[*Chez Intel x86/x86-64 il n'existe que l'instruction F2XM1 qui s'en rapproche en calculant* $2^x - 1$
Donnez un point à celui qui arrivera à la citer sans que vous ne parliez de cette instruction]

WARNING : indiquez aux étudiants que s'ils ont besoin de calculer une puissance lors des examens ou dans les projets... ils devront réécrire eux-mêmes la fonction *pow* (elle ne sera *jamais* autorisée durant toute l'année, sauf s'ils l'écrivent au moins une fois à chaque examen sur leur copie). L'accent circonflexe ne sera PAS accepté dans les examens en tant qu'opérateur calculant la puissance.

3.3 Parité

–RAS–

4 Réécriture de boucles

— Rappelez les différences entre boucles For et While
— Précisez que la boucle For peut être optimisée de différentes manières par le compilateur (même si les étudiants ne comprennent pas encore cela, ils entendent le terme une première fois)
— À l'inverse, la boucle While ne peut pas être optimisée aussi facilement

5 Fonctions, Procédures, et Effets de bord

— Rappelez la différence entre *algorithmique* et *programmation*
— Précisez succinctement qu'un ordinateur contient un ou des processeurs qui sont les vrais exécutants du code

5.1 Portée des Variables

— Expliquez les termes *variables* et *paramètres*
— Expliquez la durée de vie des variables et leurs portées (fonction, boucles, ...)
— Indiquez oralement que certaines normes de programmation autorise ou interdisent certaines pratiques
Exemple : en C89, il est impossible de déclarer des variables dont la portée se limite à une boucle, mais en C99 oui.
— Illustrez le terme *scope* à l'aide de l'indentation dans les codes exemples
— Précisez qu'en Python c'est justement l'indentation qui explicite ces scopes, alors qu'en C, ce sont les accolades

5.2 Fonctions et Procédures

— Une *fonction* retourne une valeur
— Une *procédure* ne retourne aucune valeur

5.3 Effets de bord

— Reprendre les remarques sur la portée des variables
— Écrivez une fonction simple au tableau, et écrivez une deuxième fonction simple qui appelle la première (au hasard addition et multiplication)
— Demandez quelle variable peut agir et où
— Retirez le "retourne" final pour y mettre un "print" : demandez la différence avant/après
— Explicitez qu'afficher à l'écran est beaucoup plus complexe que ce que l'on croit...
— Indiquez que "print" produit donc des effets de bords

6 Propriétés des fonctions

L'art de la programmation réside en partie dans l'écriture de fonctions possédant certaines propriétés :
— *Déterminisme* : fonction renvoyant toujours le même résultat pour les mêmes paramètres
— *Pure* : une fonction est dite pure si elle est déterministe *ET* ne produit aucun effet de bord
(Citez succinctement les autres propriétés sans les détailler : fonctions réentrantes, idempotentes, ...)

Ce document et ses illustrations ont été réalisés par Fabrice BOISSIER en septembre 2024
(dernière mise à jour octobre 2024)