Comment aborder l'écriture d'un algorithme ?

Comment organiser cette écriture ?

Jean-Cédric Chappelier (version 1.0), modifiée par Ronan Boulic

Version 1.4 – février 2019

Ce document donne quelques conseils sur la méthode de travail pour se lancer dans l'écriture d'un algorithme puis, dans un second temps, pour l'écrire de façon formelle dans le cadre du cours « Information, Calcul et Communication ».

Dans la première étape nous nous concentrons sur les moyens d'identifier les composantes de la solution d'un problème conduisant à l'*ébauche* de l'algorithme. Dans la seconde étape nous insistons sur le style et la *syntaxe* d'écriture du pseudocode.

1 De la donnée du problème à *l'ébauche* de l'algorithme

Le tout premier conseil est justement de ne pas commencer par la syntaxe (« *comment* écrire? ») mais, vraiment, de commencer par le fond/le but (« *quoi* écrire? ») : ne vous bloquez pas sur comment écrire votre algorithme si vous ne savez pas encore clairement ce que vous voulez écrire.

Le premier conseil est donc de réfléchir, faire un/des brouillon(s), schémas, organigramme, etc. Cette première étape est essentiellement une étape ***papier-crayon*** (ou tablette/stylet) dans laquelle on travaille la matière de la donnée de façon à identifier:

- l'idée d'une méthode systématique reformulant les éléments de la donnée

En l'absence d'une idée claire dès le départ, on cherchera à verbaliser :

- les sous-problèmes ou actions à effectuer

- la séquence temporelle dans lequel ces actions doivent se succéder

- les éventuelles conditions auxquelles une action, ou sa répétition, est soumise

Exemples : le cours fournit plusieurs illustrations de cette approche descendante en montrant, indépendamment de la donnée, des questions/réponses clarifiant le problème (M1.L2 slide 30), le principe de la solution (M1L2 slides 32, 34, 36, 39, M1.L4 slide 9, 39), une ébauche de haut-niveau (M1.L3 slide 53, M1.L4 slides 27, 29).

2 De l'ébauche de l'algorithme au pseudocode

Une fois au clair sur le « quoi », et seulement à ce moment là, préoccupez-vous de la mise en forme.

Commencez pour cela par écrire formellement (en français tout de même) la description la plus précise possible **des entrées fournies à l'algorithme** et **la sortie** obtenue.

Au niveau des **entrées**, pour avoir une correspondance plus directe avec les langages de programmation, on distingue deux familles pour lesquelles on adopte la définition suivante:

- **Entrée = Entrée non-modifiée**: les valeurs fournies à l'algorithme peuvent être modifiée dans l'algorithme MAIS *ces modifications n'ont aucune conséquence à l'extérieur de l'algorithme.* C'est le cas classique et recommandé pour la robustesse de l'algorithme mais il induit parfois un surcoût du fait de la copie des valeurs des paramètres.

- **Entrée modifiée** : les entités fournissant ces valeurs en entrées peuvent être modifiées dans l'algorithme et cela *se répercute sur les entités externes à l'algorithme qui sont ainsi modifiées.* Cela correspond au passage par référence dans certains langages.

L'ordre des paramètres dans le prototype de la fonction devrait toujours refléter le même ordre: **entrées non modifiées / entrées modifiées** (on privilégie cet ordre car il est le plus proche de l'ordre **entrées / sorties**).

Remarque : ne pas confondre un *affichage* avec une « sortie ». Il est rare qu'un algorithme mentionne l'action d'afficher sauf si c'est explicitement le problème à résoudre.

Exemple : algorithme de recherche d'*une* des valeurs maximales dans une liste :

Valeur maximale
entrée : *L une liste non vide de nombres* **sortie** : *la (ou une des) valeur(s) maximale(s) de la liste*

Par défaut, l'indication « **entrée** », utilisée seule, veut dire **entrée non-modifiée**
Par défaut, l'indication « **sortie** » doit être associée à un résultat transmis avec l'instruction **Sortir** indiquée explicitement dans l'algorithme (expliquée plus loin).

3 Les composantes du pseudocode

Utilisez ensuite les instructions suivantes pour écrire votre pseudocode :

— affectation : ←
p.ex. : $x \leftarrow 3$

— toutes les opérations mathématiques : notation usuelle
p.ex. : $x \geq 2$

— désignation d'un élément d'une liste : parenthèses rondes () ou carrées [], au choix
p.ex. : le i-ème élément de la liste L : $L(i)$ ou $L[i]$

3.1 Les structures de contrôle.

TRES important : décaler sur la droite les instructions contrôlées (**indentation**)
On peut ajouter une barre verticale pour mieux les distinguer

3.1.1 « **Si** ... alors ... Sinon …» :

Le mot « alors » n'est pas écrit pour alléger le pseudocode
La clause **Sinon** est optionnelle

```
Si condition
    Instructions
```

```
Si condition
    Instructions
Sinon
    Instructions
```

On remplacera l'écriture ci-dessous par celle-ci :

```
Si condition
    // ne rien faire
Sinon
    Instructions
```

```
Si négation de la condition
    Instructions
```

3.1.2 « **Tant que** ... » : la **boucle conditionnelle** possède deux formes

```
Tant que condition
    Instructions
```

```
Faire
    Instructions
Tant que condition
```

3.1.3 « **Pour …** » encore appelée **itération**

Les conventions d'écriture des boucles « **Pour** » incluent :

- que sur les nombres entiers l'incrément est implicitement de 1;

  Pour i allant de 1 à n

  Instructions

- sinon il faut le préciser;

  Pour i allant de 1 à n de 2 en 2

  Instructions

- autre exemple :

  Pour i allant de n à 1 en descendant

  Instructions

Si l'ensemble décrit par la boucle est l'ensemble vide, la boucle ne se déroule pas du tout; p.ex.

Pour i allant de 1 à n ne fera *rien* si n est inférieur ou égal à 0.

On utilisera la boucle conditionnelle **Tant que** si une *condition* supplémentaire doit être remplie pour poursuivre l'itération.

3.1.4 « **Pour tout** ... » :

Cette formulation de boucle peut être utilisée s'il n'est pas nécessaire de connaître la position/l'indice de l'élément dans l'ensemble *L* pour traiter cet élément :

Pour tout élément x de L

Instructions utilisant seulement *x*

3.1.5 Terminaison prématurée d'une boucle ou itération

La terminaison prématurée d'une boucle doit être associée au test d'une condition ; elle peut être exprimée avec l'instruction inconditionnelle suivante :

Pour …..

Instructions

Si condition

Sortir de la boucle

Instructions

3.1.6 Passage prématuré à l'itération suivante d'une boucle

Certains langages permettent de *passer prématurément à l'itération suivante d'une boucle* à l'aide d'un mot clef comme par exemple « continue » associé au test d'une expression logique. Cependant, il suffit d'inverser le test pour obtenir le même résultat. C'est pourquoi nous recommandons de remplacer la forme de gauche par celle de droite:

Pour …..
    Instructions
    Si *condition*
        Continue
    Instructions

Pour …
    Instructions
    Si ***négation de la condition***
        Instructions

3.1.7 Bonne pratique sur le point de sortie

Il est recommandé d'avoir un seul point de sortie d'une boucle, celui qui est exprimé au début avec le mot-clef **Tant que** ou implicitement avec la borne finale d'une itération **Pour**.

S'il y a des conditions supplémentaires à remplir il est préférable de les rassembler en ce point de sortie unique. La boucle **Tant que** est plus adaptée que l'itération **Pour** pour exprimer cette souplesse supplémentaire de l'exécution.

L'approche 3.1.5 est néanmoins acceptée pour traiter des conditions additionnelles.

3.1.8 Terminaison de l'algorithme ou d'une fonction : « **Sortir** : »

Sortir : x

Notez que l'instruction « Sortir : » met fin à l'algorithme (même s'il y a encore des lignes en dessous). S'il s'agit d'une fonction cela quitte la fonction en retournant éventuellement une valeur utilisée au niveau supérieur de l'algorithme.

3.2 Actions de plus haut niveau.

— si nécessaire, pour afficher une valeur/expression, utilisez simplement « afficher »; p.ex.:
afficher x.

Sauf mention contraire dans la donnée, vous pouvez également utiliser tout algorithme *vu en cours* (taille, tri, recherche, plus court chemin) en le désignant par un nom suffisamment clair; par exemple :

— $n \leftarrow \text{taille}(L)$

— $L^0 \leftarrow \text{trier}(L)$ ou $L^0 \leftarrow \text{tri}(L)$

Note : au niveau formel, il est préférable de considérer que les algorithmes ne modifient pas leur entrée mais produisent un nouvel objet (comme une fonction mathématique). Par exemple ci-dessus, la liste *L* n'est pas modifiée par l'algorithme de tri, mais celui-ci retourne une nouvelle liste (triée). Le cours M1.L4 illustre ce point.

4 Exemple

Voici l'exemple complet, la recherche d'*une* des valeurs maximales dans une liste :

Valeur maximale
entrée : *L une liste non vide de nombres* **sortie** : *la (ou une des) valeur(s) maximale(s) de la liste*

n ← taille(L)
x_{max} ← L(1)
Pour i allant de 2 à n
  Si L(i) > x_{max}
  x_{max} ← L(i)
Sortir : x_{max}

Notez que l'algorithme ci-dessus est correct dans tous les cas en raison des conventions :

— la boucle « **Pour** » ne fait rien si n vaut 1 (et donc, dans ce cas, on retourne finalement $L(1)$);

— la description de l'entrée est toujours vraie : ci-dessus la liste L ne peut (axiomatiquement) pas être vide; il est donc important de bien préciser les hypothèses de départ. Par exemple l'algorithme suivant n'est pas correct :

Valeur maximale
entrée : *L une liste de nombres* **sortie** : *la (ou une des) valeur(s) maximale(s) de la liste*
$n \leftarrow taille(L)$ $x_{max} \leftarrow L(1)$ *etc.*

car $L(1)$ n'est pas défini pour une liste vide (et que l'on n'a pas empêché cette possibilité *a priori*). Il faut donc l'écrire comme donné plus haut, et pas autrement, car il n'y a de toutes façons pas de définition de « la valeur maximale » pour une liste vide.