

Pseudo-code – Langage de Description d’Algorithmes

2 novembre 2024

Ce document donne quelques conseils sur la façon formelle d’écrire un algorithme dans le cours “ Information, Calcul et Communication ”. Il se focalise donc sur le style, la *syntaxe*.

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, etc.

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 le problème (en français ou en anglais) par la description la plus précise possible des entrées fournies à l’algorithme et la sortie obtenue.

Par exemple, pour l’algorithme de recherche d’une des valeurs maximales dans une liste, on écrit (en français ou en anglais) :

Valeur maximale
entrée : <i>liste L (non-vide) de nombres entiers</i> sortie : <i>la valeur maximale de la liste</i>
(instructions)

Maximal Value
input : <i>(non-empty) liste L of integers</i> output : <i>the largest value in the list</i>
(instructions)

Utilisez ensuite les instructions suivantes :

— affectation : $\leftarrow$

p.ex. : $x \leftarrow 3$

— les opérations mathématiques suivantes :

$+$, $-$, $\cdot$, $/$, $\bmod$, $<$, $\leq$, $=$, $\neq$, $\geq$, $>$, $\lfloor \rfloor$ (la partie entière)

p.ex. : $x \geq 2$

— les opérations logiques : et (and), ou (or)

p.ex. : $x > 0$ ou $y > 0$

p.ex. : $x > 0$ or $y > 0$

— 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 $L[i]$

— désignation d’un sous-ensemble d’éléments d’une liste : à nouveau, avec des parenthèses rondes ou carrées, au choix, et la notation “ $a : b$ ” pour désigner un intervalle

p.ex. : $L(1 : m)$ désigne les m premiers éléments de la liste L (qui peut en contenir plus que m).

— ajout d’un élément à la fin d’une liste

p.ex. : $\text{append}(L, e)$, $\text{ajouter}(L, e)$

— création d’une liste vide

p.ex. : $L \leftarrow \text{empty}$, $L \leftarrow \text{vide}$, $L \leftarrow \{\}$

- référence à un autre algorithme que vous avez écrit ou que vous avez vu en cours ; p.ex. :
 - $n \leftarrow \text{taille}(L)$
 - $L' \leftarrow \text{tri par insertion}(L)$
 - $n \leftarrow \text{algorithm1}(L, n)$
 - $\text{sorted}L \leftarrow \text{sort}(L, n)$
- les **trois structures de contrôle** :
 - les tests :

Si condition
 | instructions
Sinon
 | instructions

if condition
 | instructions
else
 | instructions

- les boucles simples :

Pour i allant de 1 à n
 | instructions

for i from 1 to n
 | instructions

- les boucles conditionnelles :

Tant que condition
 | instructions

while condition
 | instructions

Veuillez noter que dans la boucle simple ci-dessus (celle commençant par **Pour** (ou **for**)), la valeur de i est automatiquement incrémentée de 1 à chaque exécution du coeur de la boucle, tandis que dans une boucle conditionnelle (celle commençant par **Tant que** (ou **while**)), c'est à vous de changer quelque chose dans le coeur de la boucle pour que l'algorithme en sorte une fois. Par exemple, si vous écrivez :

$i \leftarrow 1$
Tant que $i \leq n$
 | **Afficher** : i

$i \leftarrow 1$
while $i \leq n$
 | **display** : i

L'algorithme affichera une suite infinie de 1 sans jamais s'arrêter. Pour éviter ça, il faut manuellement rajouter l'instruction $i \leftarrow i + 1$ dans le coeur de la boucle.

- Voici deux autres exemples de boucles simples avec une autre incrémentation :

Pour i allant de 1 à n de 2 en 2
 | instructions

for i from 1 to n in increments of 2
 | instructions

et

Pour i allant de n à 1 en descendant
 | instructions

for i from n to 1 going down
 | 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.

Remarque : Pensez à indenter (décaler à droite) et même marquer par une barre verticale, les instructions contrôlées par une structure de contrôle.

— La **terminaison** de l'algorithme : “ **Sortir :** ” ou “ **return :** ”

p.ex. : **Sortir :** x

p.ex. : **return :** x

notez que l'instruction “ **Sortir :** ” (ou “ **return :** ”) met fin à l'algorithme (même s'il y a encore des lignes en dessous) ;

— si nécessaire (rare dans des algorithmes formels), pour afficher une valeur/expression, utilisez simplement “ **Afficher :** ” ou “ **display :** ”

p.ex. : **Afficher :** x .

p.ex. : **display :** x .

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).

Terminons par un exemple complet : l'algorithme de recherche d'une des valeurs maximales dans une liste :

Valeur maximale
entrée : liste L (<i>non-vide</i>) de nombres avec n éléments sortie : la valeur maximale de la liste
$x_{\max} \leftarrow L[1]$ Pour i allant de 2 à n Si $L[i] > x_{\max}$ $x_{\max} \leftarrow L[i]$ Sortir : $x_{\max}$

Maximal Value
input : (<i>non-empty</i>) liste L with n integers output : the largest value in the list
$x_{\max} \leftarrow L[1]$ for i from 2 to n if $L[i] > x_{\max}$ $x_{\max} \leftarrow L[i]$ return : $x_{\max}$

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

- la boucle simple “**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.