

Comment écrire proprement un algorithme?

Jean-Cédric Chappelier

Version 1.3 – oct. 2026

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*, mais donne aussi quelques conseils méthodologiques.

1 Méthode générale

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 tout de même) par la description la plus précise possible des entrées fournies à l'algorithme et la sortie obtenue.

Par exemple, pour l'algorithme de recherche de la *valeur* maximale dans une liste, on écrit :

Valeur maximale
entrée : <i>L une liste non vide de nombres</i>
sortie : <i>La valeur maximale de la liste L</i>
...

2 « Boite à outils » (instructions de base)

Utilisez ensuite les instructions suivantes :

- affectation : $\leftarrow$
- p.ex. : $x \leftarrow 3, L \leftarrow (3, 5)$

- toutes les opérations mathématiques : notation usuelle
p.ex. : $x \geq 2$, $\log(x)$
- 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 $L[i]$
- les trois structures de contrôle :
 - branchements conditionnels :

```

Si condition
  |
  | instructions
Sinon
  |
  | instructions

```

- boucles conditionnelles :

<pre> Tant que condition instructions </pre>	<pre> Répéter instructions tant que condition </pre>
---	--

- itérations :

```

Pour tout élément  $x$  de  $L$ 
  |
  | instructions
ou
Pour  $i$  allant de 1 à  $n$ 
  |
  | instructions

```

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

- que sur les nombres entiers l'incrément est de 1 ; sinon il faut le préciser ;
p.ex. :

```

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.

Attention aussi : i et n (ou L dans le premier cas) ne doivent pas être modifiés dans la boucle, sinon le comportement n'est pas défini. Utiliser une boucle conditionnelle dans de tels cas.

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** : » ;
p.ex. : **Sortir** : x ;

Notez que l'instruction « **Sortir** : » 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** » ;
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 :

- $n \leftarrow \text{taille}(L)$
- $L' \leftarrow \text{trier}(L)$ ou $L' \leftarrow \text{tri}(L)$

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).

3 Exemple

Donnons un exemple complet : un algorithme de recherche de la *valeur* maximale dans une liste non vide :

Valeur maximale
entrée : L une liste non vide de nombres
sortie : La valeur maximale de la liste L
$n \leftarrow \text{taille}(L)$ $x_{\max} \leftarrow L[1]$ Pour i allant de 2 à n Si $L[i] > x_{\max}$ $x_{\max} \leftarrow L[i]$ Sortir : $x_{\max}$

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

- l'itération « **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 valeur maximale de L
$n \leftarrow \text{taille}(L)$ $x_{\max} \leftarrow L[1]$ 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.

4 Plus de notations

Donnons maintenant quelques notations usuelles d'opérations sur des listes.

Pour ajouter des éléments à une liste, ou aussi concaténer (fusionner l'une derrière l'autre) deux listes, vous pouvez, si vous le souhaitez, utiliser la notation $\oplus$: $L_1 \oplus L_2$ crée la liste de tous les éléments de L_1 suivis de tous les éléments de L_2 . Notez que c'est une opération sur des *listes*.

Pour un élément à une liste, on peut alors simplement écrire :

pour un ajout en fin : $L \leftarrow L \oplus (x)$

pour un ajout au début : $L \leftarrow (x) \oplus L$

Comment ajouter un élément x à une position i dans une liste ?

Le plus simple est de l'écrire explicitement (la taille de L ayant au préalable été affectée à n) :

$$L \leftarrow (L[1], \dots, L[i-1], x, L[i], \dots, L[n])$$

avec la convention que si i vaut 1, alors x est ajouté au début (mais préférez si possible la notation ci-dessus avec $\oplus$) et que si i vaut $n+1$, x est ajouté en fin (mais, là aussi, préférez si possible la notation ci-dessus avec $\oplus$).

Comment vider une liste ?

Il suffit simplement de lui affecter la liste vide : $L \leftarrow ()$

Comment supprimer un élément d'une liste ?

En écrivant explicitement la liste sans cet élément :

$$L \leftarrow (L[1], \dots, L[i-1], L[\textcolor{red}{i} + 1], \dots, L[n])$$

Mais pourquoi ne peut-on pas noter $L \ominus L[i]$? (ne **jamais** écrire ça !)

Parce que cette notation est *ambiguë* : évalué à droite d'une expression (« *r-value* »), l'expression $L[i]$ représente une **valeur**. Par exemple $a \leftarrow L[i]$ veut bien dire $a \leftarrow 4$ si $L[i]$

vaut 4.

Que voudrait alors dire $L \ominus 4$ si L contient plusieurs fois la valeur 4 ? Lequel supprime-t-on ? Ou les supprime-t-on tous ?

On pourrait à la limite tolérer une notation non ambiguë telle que $L \ominus i$ (où i désigne alors une position/un indice), mais je pense que ce ne serait pas très bon pédagogiquement (encore une notation de plus ; risque de confusion avec la notation critiquée ci-dessus ; ...).

Autre remarque concernant les écritures ambiguës à éviter :

il ne faut pas écrire de condition comme « $a = b = c$ » ; et cela pour deux raisons :

— premièrement en raison du « sinon » : que veut dire le « sinon » ?

Est-ce « $(a \neq b)$ **OU** $(b \neq c)$ » (b est différent de l'un ou l'autre),
ou alors est-ce « $(a \neq b)$ **ET** $(b \neq c)$ **ET** $(a \neq c)$ » (les trois sont différents) ?

Autrement dit, est-ce « $a = b = c$ » signifie

« $(a = b)$ **ET** $(b = c)$ »

ou

« $(a = b)$ **OU** $(b = c)$ **OU** $(a = c)$ »

?

Il y a en fait quatre situations différentes possibles pour la « négation de » xxx :
 xyx , xyx , yxx et $xyz...$

Lesquelles veut-on (toutes ou seulement certaines) ?

— de plus, si l'on autorise « $a = b = c$ », alors c'est la porte ouverte à « $a \neq b \neq c$ »...
...qui est clairement ambiguë (quid de la relation entre a et c ?).

5 Algorithmes récursifs

Pour concevoir un algorithme récursif, il vous faut *toujours* trois choses :

- les conditions d'arrêt ;
- voir « le schéma qui se répète » ;
- utiliser le résultat de l'appel récursif.

Pour « le schéma qui se répète », vous avez essentiellement¹ deux approches :

- soit « réduire d'un » (enlever un élément d'une liste, soustraire une petite valeur à un nombre, ...)
- soit « couper en deux » (travailler sur une ou les deux moitié(s) d'une liste, diviser un nombre par une petite valeur, ...)

Si vous avez du mal à trouver « le schéma qui se répète » d'une façon générale, essayez alors avec des cas particuliers à peine plus compliqués (trois ou quatre de plus) que les conditions d'arrêt.

1. Il y en a d'autres, mais ces deux là sont de loin les plus fréquents.

Faites aussi bien attention à ne pas oublier le troisième point : *utiliser* le résultat de l'appel récursif ; souvent les débutant(e)s oublient ce point car ils/elles imaginent que « c'est automatique », que les appels récursifs « se souviennent » des appels précédents. Ce n'est pas le cas.

Prenons un exemple : recherche de la *valeur* maximale dans une liste (non vide) :

— Quelles sont, ou quelle est, les/la condition(s) d'arrêt ?

☞ la liste réduite à un seul élément : la valeur maximale est alors simplement la valeur de cet élément.

— Quel est « le schéma qui se répète » ? enlever un ? ou couper en deux ?

Faisons les deux pour illustrer :

— **enlever un** : lequel ? le premier ? le dernier ?

☞ ici peu importe, ça ne change rien ; disons : « le premier ».

Si l'on a donc d'un côté le premier élément (celui qu'on va enlever) et de l'autre une **solution** (peu importe comment) au problème « trouver la valeur maximale » pour la sous-liste à partir du deuxième élément, peut-on résoudre le problème de départ ?

☞ oui : c'est le plus grand de ces deux.

— **couper en deux** :

Si l'on a une **solution** au problème « trouver la valeur maximale » pour la première moitié de la liste et une **solution** pour la seconde moitié de la liste, peut-on résoudre le problème de départ ?

☞ oui : c'est le plus grand de ces deux.

— utiliser le résultat de l'appel récursif : ici c'est simple :

— dans la version « **enlever un** » :

si l'on note S la **solution** au problème « trouver la valeur maximale » pour la sous-liste à partir du deuxième élément, alors la solution que l'on cherche est : $\max(L[1], S)$;

— dans la version « **couper en deux** » :

si l'on note S_1 la **solution** au problème « trouver la valeur maximale » pour la première moitié de la liste et S_2 la **solution** pour la seconde moitié de la liste, alors la solution que l'on cherche est : $\max(S_1, S_2)$.

Ce qui nous donne les deux algorithmes suivants :

Valeur maximale V1
entrée : L une liste non vide de nombres sortie : La valeur maximale de la liste L
$n \leftarrow \text{taille}(L)$ Si $n = 1$ Sortir : $L[1]$ $S \leftarrow \text{Valeur maximale V1} (L[2], \dots, L[n])$ Sortir : $\max(L[1], S)$

Valeur maximale V2
entrée : L une liste non vide de nombres sortie : La valeur maximale de la liste L
$n \leftarrow \text{taille}(L)$ Si $n = 1$ Sortir : $L[1]$ $S_1 \leftarrow \text{Valeur maximale V2} (L[1], \dots, L[\lfloor \frac{n}{2} \rfloor])$ $S_2 \leftarrow \text{Valeur maximale V2} (L[\lfloor \frac{n}{2} \rfloor + 1], \dots, L[n])$ Sortir : $\max(S_1, S_2)$