Algorithmes et Structures de données | 2021

Journée 3 – Vendredi 26 février

Complément – Au sujet du pseudo-code

L'utilisation du pseudo-code sert à exprimer l'idée d'un algorithme, sans être contraint par la sémantique et la syntaxe d'un langage de programmation particulier. Du cou, il n'y a par définition pas de manière canonique d'exprimer les choses en pseudo-code, puisque le but est précisément de s'affranchir des contraintes strictes imposées par un compilateur ou par un interprète. Tout au plus, dans la plupart des livres d'algorithmique il est fait mention de certaines conventions de base.

Par exemple, dans l'algorithme du tri par sélection, on va écrire $swap(A[\mathrm{i}],A[\mathrm{j}])$ plutôt que :

$\mathrm{int}\ temp = A[\mathrm{i}]$

$A[\mathrm{i}] = A[\mathrm{j}]$

$A[\mathrm{j}] = temp$

De même, au lieu d'écrire $\mathrm{if\ a\ !=\ 2}$, on écrira $\mathrm{if\ a} \neq 2$ même si le caractère '≠' n'est généralement pas utilisé dans les langages de programmation. Encore une fois, l'objectif est de transmettre l'idée de l'algorithme de la manière la plus efficace que possible. Ainsi, l'expression $swap(A[\mathrm{i}],\ A[\mathrm{j}])$ est nettement plus claire que la suite d'instructions donnée ci-dessus. Et peu importe si dans le langage que vous allez finalement utiliser pour implémenter votre algorithme, il n'existe pas d'instruction ou de fonction swap, vous pouvez quand même l'utiliser dans votre pseudo-code.

Autre exemple : en admettant que S soit une collection (ensemble, liste, etc.) contenant des objets de type $Student$ ayant un champ $grade$, vous pouvez définir P l'ensemble des étudiants promus par :

$$P = \{\ s \in S\ \mid\ s.grade \geq 4\ \}$$

plutôt que d'écrire une boucle qui crée explicitement l'ensemble P.

Si dans votre implémentation, vous utilisez un langage de programmation qui offre la notion de liste en compréhension (Python par exemple), l'écriture de votre algorithme dans ce langage se rapprochera beaucoup de la notation ci-dessus. Dans le cas contraire, vous devrez peut-être écrire une boucle, soit directement dans la fonction qui implémente votre algorithme, soit dans une fonction annexe. Il se peut également qu'il existe dans ce langage une librairie qui implémente l'équivalent des listes en compréhension, mais avec une syntaxe relativement différente.

Concernant cette notation particulière (liste en compréhension), le langage Python offre une manière d'exprimer la création d'une collection de manière simple et élégante, se rapprochant fortement de la notion ensembliste mathématique. Néanmoins, si vous décidez d'exprimer systématiquement vos algorithmes en Python plutôt qu'en pseudo-code, vous allez forcément tomber sur des cas où la syntaxe Python sera lourde ou peu claire, comparée à celle d'un autre langage ou à du pseudo-code utilisant une notation mathématique.

Un autre problème avec la notation des listes en compréhension en Python tient au fait qu'elle passe par la création d'une nouvelle liste. Or dans certains cas, pour des raisons de gestion de mémoire, peut-être vaudra-t-il mieux simplement itérer sur la liste de base et ne traiter que certains éléments, dans l'exemple ci-dessus, les éléments s tels que $s.grade \geq 4$. Du coup, même si en pseudo-code vous avez écrit votre algorithme en utilisant la notation $\{\ s \in S\ \mid\ s.grade \geq 4\ \}$, peut-être que dans une implémentation particulière, vous allez quand même faire le choix de passer par une boucle (même en Python), pour des raisons de gestion mémoire.

Or l'utilisation d'un langage particulier et de certaines constructions particulières de ce langage (plutôt que d'autres constructions) a tendance à provoquer une confusion entre l'idée de votre algorithme et une implémentation possible de celle-ci. En exprimant votre algorithme en pseudo-code, vous dites d'une certaine façon : *"Voici l'idée de mon algorithme; pour son implémentation, je vous laisse choisir le langage et les constructions (structures de données + structures de contrôle) qui conviennent à votre contexte particulier."*