TD1 – Exercices en lien avec le cours

Killian Reine

Rappels de cours.

Un ***algorithme*** est une suite finie et ordonnée d'instructions précises, non ambiguës et exécutables mécaniquement, qui, à partir de données d'entrée, produit un résultat en un nombre fini d'étapes.

Exercice 1

Reconnaître un algorithmique

1 **Une recette de cuisine.**

Une recette de cuisine peut être considérée comme un algorithme **si** elle décrit une suite finie d'instructions précises, ordonnées, et non ambiguës permettant d'obtenir un résultat à partir d'ingrédients donnés. Cependant, dans la pratique, de nombreuses recettes font appel à des notions vagues ("cuire jusqu'à ce que ce soit prêt", "ajouter selon le goût"), ce qui empêche leur exécution mécanique. Ainsi, une recette de cuisine **n'est algorithmique que si elle est suffisamment formalisée**.

2 **Réfléchir jusqu'à avoir une bonne idée.**

Cette situation ne définit pas un algorithme. En effet, aucune suite finie d'instructions précises n'est donnée, et la condition d'arrêt (avoir une bonne idée) n'est ni objective ni vérifiable automatiquement. Il n'y a donc ni déterminisme, ni garantie de terminaison. Cette situation n'est **pas algorithmique**.

3 **Un programme qui affiche les nombres de 1 à 10.**

Ce programme est algorithmique. Il décrit une suite finie d'instructions bien définies, exécutables mécaniquement, et dont la terminaison est garantie après un nombre fini d'étapes. Le résultat attendu est clairement spécifié. Il s'agit donc bien d'un algorithme.

4 **Un programme qui affiche les nombres tant que l'utilisateur n'appuie pas sur stop.**

Ce programme ne définit pas un algorithme au sens strict. En effet, la condition d'arrêt dépend d'une action extérieure imprévisible (l'utilisateur), et il n'existe aucune garantie de terminaison. Un algorithme doit terminer pour toute entrée valide. Ce programme n'est donc **pas algorithmique**.

Exercice 2

Définir les entrées et sorties d'un algorithme

1 **Algorithme qui calcule la somme de deux entiers.**

- **Entrées :** deux entiers $a, b \in \mathbb{Z}$.
- **Sortie :** un entier $s \in \mathbb{Z}$ tel que $s = a + b$.

2 **Algorithme qui cherche le minimum dans un tableau.**

- **Entrées :** un tableau T de taille $n \geq 1$, contenant des éléments d'un ensemble totalement ordonné (par exemple des entiers ou des réels).
- **Sortie :** une valeur m appartenant à T telle que

$$m = \min_{0 \leq i < n} T[i].$$

3 **Algorithme qui dit si un nombre est pair.**

- **Entrée :** un entier $n \in \mathbb{Z}$.
- **Sortie :** une valeur booléenne appartenant à $\{\text{Vrai}, \text{Faux}\}$, indiquant si n est pair.

4 **Algorithme de recherche dichotomique.**

— **Entrées :**
  — un tableau T de taille $n \geq 1$, trié dans l'ordre croissant ;
  — un élément x à rechercher, appartenant au même ensemble que les éléments de T.
— **Sortie :**
  — soit un indice i tel que $T[i] = x$ si l'élément est présent ;
  — soit une valeur spéciale (par exemple -1 ou `Faux`) indiquant que x n'appartient pas au tableau.

Exercice 3

Construction d'un algorithme et étude

1 **Algorithme permettant de récupérer le minimum d'un tableau.**
On suppose que le tableau T contient $n \geq 1$ éléments comparables.

```
min ← T[1]
pour i de 2 à n faire
    si T[i] < min alors
        min ← T[i]
    fin si
fin pour
retourner min
```

2 **Étude et fonctionnement de l'algorithme.**
L'algorithme initialise la variable `min` avec le premier élément du tableau. Il parcourt ensuite le tableau élément par élément. À chaque itération, l'élément courant est comparé à la valeur stockée dans `min`. Si l'élément courant est plus petit, `min` est mis à jour.
À la fin du parcours, `min` contient la plus petite valeur du tableau.

3 **Nombre de comparaisons et d'affectations.**

— **Comparaisons :** La condition `T[i] < min` est évaluée pour chaque élément à partir du deuxième. Il y a donc exactement $n - 1$ comparaisons.
— **Affectations :**
  — 1 affectation lors de l'initialisation : `min ← T[1]` ;
  — une affectation supplémentaire chaque fois qu'un nouvel élément minimum est rencontré.

Le nombre total d'affectations dépend donc de la disposition des valeurs dans le tableau.

4 **Cas du tableau $T = \{7, 3, 9, 2, 5\}$.**
Déroulons l'algorithme :

— Initialisation : `min` $= 7$;
— $3 < 7$: affectation, `min` $= 3$;
— $9 < 3$: faux, aucune affectation ;
— $2 < 3$: affectation, `min` $= 2$;
— $5 < 2$: faux.

Il y a donc :
— 4 comparaisons ;
— 3 affectations (1 initiale + 2 mises à jour).

Ce cas n'est ni le meilleur ni le pire cas.

a **Un tableau possible dans le meilleur cas.**
Le meilleur cas correspond à une situation où le minimum est le premier élément du tableau, par exemple :

$$T = \{1, 3, 5, 7, 9\}.$$

Il n'y a alors qu'une seule affectation (l'initialisation).

b **Un tableau possible dans le pire cas.**
Le pire cas correspond à un tableau strictement décroissant, par exemple :

$$T = \{9, 7, 5, 3, 1\}.$$

À chaque itération, une nouvelle affectation est effectuée. Il y a alors n affectations au total (initialisation incluse).

5 **Le nombre de comparaisons dépend-il du nombre d'éléments ?**
Oui. Le nombre de comparaisons est exactement égal à $n - 1$, où n est le nombre d'éléments du tableau. Il dépend uniquement de la taille du tableau et non de l'ordre des éléments.

Exercice 4

Boucles et contrôles
On considère l'algorithme suivant :

```
for i de 1 à N
    afficher(i)
```

1 **Combien de fois la condition $i \leq N$ est-elle testée ?**
Dans une boucle `for`, la condition de poursuite de la boucle est testée :

— une première fois avant la première itération ;
— une fois après chaque itération.

La condition $i \leq N$ est donc testée exactement $N + 1$ fois.

2 **Combien de fois l'instruction `afficher(i)` est-elle exécutée ?**
L'instruction `afficher(i)` est exécutée une fois à chaque itération de la boucle. Comme la boucle s'exécute pour $i = 1, 2, \ldots, N$, cette instruction est exécutée exactement N fois.

3 **Que devient ce nombre lorsque $N = 1$?**
Si $N = 1$:

— la condition $i \leq N$ est testée 2 fois ;
— l'instruction `afficher(i)` est exécutée 1 fois.