Parcours de tableau 2

Objectifs

- Compter des occurrences.
- Filtrer un tableau.
- Parcourir plusieurs tableaux ou sous-tableaux.

Rappel

Les algorithmes de parcours simples de tableau à n éléments ont une complexité en $O(n)$, on dit que la complexité est **linéaire**. Nous allons voir dans cette partie des parcours qui nécessitent davantage de traitements et nous verrons leur complexité.

Comptage d'occurrences

Principe : Je veux compter le nombre d'occurrences d'un élément cible dans un tableau. Il faut donc parcourir le tableau et garder en mémoire le nombre de fois où la cible a été rencontrée.

Pseudo-code générique :

```
DEBUT
  compteur ← 0
  i ← 1
  TQ i   n FAIRE
    SI T[i] = cible ALORS
      compteur ← compteur + 1
    FSI
    i ← i+1
  FTQ
  RENVOYER compteur
FIN
```

La boucle principale parcourt toutes les cases du tableau et à chaque passage de boucle le nombre d'opérations est en $\Theta(1)$. Donc la complexité de cet algorithme est en $\Theta(n)$.

> **Exercice**
>
> Adapter cet algorithme pour compter le nombre de fois qu'apparaît chaque élément dans le cas où le tableau parcouru est composé d'entiers strictement positifs inférieurs ou égaux à N.

Filtrage de tableau

Principe : On va parcourir le tableau et traiter les éléments qui vérifient une certaine condition.

Par exemple, on peut extraire les éléments pairs d'un tableau.

Pseudo-code :

```
VARIABLES :
  i : entier
  Reponse : tableau vide
DEBUT
  i ← 1
  TQ i   n FAIRE
    SI T[i] mod 2 = 0 ALORS
      Ajouter T[i] à Reponse
    FSI
    i ← i+1
  FTQ
  RENVOYER Reponse
FIN
```

De la même façon que dans la recherche d'occurrences, la complexité est en $\Theta(n)$.

> **Exercice**
>
> Écrire un algorithme qui additionne tous les nombres impairs d'un tableau.

Parcours double

Principe : On a deux tableaux (ou des sous-tableaux), et on veut les parcourir avec deux indices différents. Il faut donc deux boucles pour le faire.

Par exemple, on peut compter le nombre de paires (i, j) telles que $T[i] < T[j]$ avec $i < j$.

Pseudo-code :

```
DEBUT
  compteur ← 0
  i ← 1
  TQ i   n FAIRE
    j ← i+1
    TQ j   n FAIRE
      SI T[i] < T[j] ALORS
        compteur ← compteur + 1
      FSI
      j ← j+1
    FTQ
    i ← i+1
  FTQ
  RENVOYER compteur
FIN
```

Analyse de complexité :

Pour chaque valeur de i de 1 à n, on parcourt les indices j de $i+1$ à n. Le nombre total d'opérations est :

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} \Theta(1) = \sum_{i=1}^{n} (n-i) = \sum_{k=0}^{n-1} k = \frac{n(n-1)}{2}$$

La complexité est donc en $\Theta(n^2)$, on dit que c'est une complexité **quadratique**.

> ⚠ Attention
>
> Les boucles imbriquées conduisent souvent à une complexité quadratique ou pire. Il est important d'analyser si cette complexité est nécessaire ou si on peut optimiser l'algorithme.

Exercice

1. Écrire un algorithme qui compte le nombre de paires d'éléments égaux dans un tableau.
2. Écrire un algorithme qui vérifie si un tableau contient des doublons.
3. Quelle est la complexité de ces algorithmes ?

Comparaison de complexités

Voici un tableau récapitulatif des complexités vues :

Type de parcours	Complexité	Exemple
Parcours simple	$\Theta(n)$	Recherche d'un élément, comptage
Filtrage	$\Theta(n)$	Extraction d'éléments pairs
Parcours double	$\Theta(n^2)$	Recherche de paires, doublons

Principe général

— Un parcours simple avec traitement en $\Theta(1) \rightarrow$ complexité $\Theta(n)$
— Deux boucles imbriquées avec traitement en $\Theta(1) \rightarrow$ complexité $\Theta(n^2)$
— Trois boucles imbriquées avec traitement en $\Theta(1) \rightarrow$ complexité $\Theta(n^3)$