Parcours de tableau 1

Objectifs

— Savoir définir et appliquer les parcours fondamentaux de tableau.
— Vérifier si un tableau est trié.
— Chercher un extremum dans un tableau.

Introduction

Définition

Un tableau est une structure de données permettant de stocker des valeurs. Un tableau est itérable (c'est à dire qu'il peut être utilisé dans les boucles de type for).

Attention : Indexation

En Python, le premier indice d'un tableau est 0.
En algorithmique, on commence à 1.
Représentation d'un tableau :

indice	1	2	3	4	5	...
valeurs						

Beaucoup de traitements algorithmiques nécessitent de parcourir des tableaux pour chercher, filtrer ou calculer. Écrire des algorithmes de parcours de tableau est à l'algorithmique ce que faire des gammes est à la musique.

Parcours séquentiel

> Définition
>
> On appelle **parcours séquentiel** d'un tableau, le fait d'accéder successivement aux cases d'un tableau à l'aide d'une boucle.

Pseudo-code générique :

```
DEBUT
  i ← 1
  TQ i   n FAIRE
    Traitement(T[i])
    i ← i+1
  FTQ
FIN
```

Si la complexité de la fonction `Traitement(T[i])` est en $\Theta(1)$ alors la complexité de ce type d'algorithme est en $\Theta(n)$ (on visite toutes les cases une seule fois).

> Exercices
>
> 1. Écrire un algorithme qui affiche les éléments contenus dans les cases d'indice pair.
> 2. Écrire un algorithme qui affiche tous les éléments d'un tableau en commençant par l'indice $i \in [1, n]$ de manière cyclique (après $T[n]$, il affichera $T[1]$).

Vérification et recherche

Vérification

Principe : On parcourt le tableau et on vérifie que chaque élément vérifie une condition établie au préalable.

Par exemple, on peut vérifier si le tableau est trié dans l'ordre croissant. Il faut donc que chaque élément soit inférieur ou égal au suivant.

Pseudo-code :

Avec variable booléenne

```
DEBUT
  i ← 1
  juste ← VRAI
  TQ i < n FAIRE
    SI T[i] > T[i+1] ALORS
      juste ← FAUX
    FSI
    i ← i+1
  FTQ
  RENVOYER juste
FIN
```

Sans variable booléenne

```
DEBUT
  i ← 1
  TQ i < n ET T[i]   T[i+1] FAIRE
    i ← i+1
  FTQ
  RENVOYER i = n
FIN
```

Analyse de complexité :

Dans le premier algorithme, on teste toutes les cases du tableau donc la complexité est en $\Theta(n)$.

Dans le second algorithme :

- **Meilleur cas :** le premier élément testé ne vérifie pas la condition et $\check{C}(n) = \Theta(1)$
- **Pire cas :** le tableau est trié et il faut tester tout le tableau donc $\hat{C}(n) = \Theta(n)$
- **Complexité moyenne :** $\bar{C}(n) = O(n)$

Chercher le minimum ou maximum d'un tableau

Principe : On cherche une valeur particulière dans un tableau, donc il faut le parcourir et stocker la valeur cherchée (si besoin).

Pseudo-code :

```
Algorithme de recherche du minimum
DEBUT
  i ← 2
  temp ← T[1]
  TQ i   n FAIRE
    SI T[i] < temp ALORS
      temp ← T[i]
    FSI
    i ← i+1
  FTQ
  RENVOYER temp
FIN
```

On va tester toutes les cases une fois dans cette recherche, la complexité est en $\Theta(n)$.

Question : Que doit-on changer pour chercher le maximum du tableau ?

Exemple : l'addition en base b

On prendra ici $b = 10$, mais l'algorithme sera le même en changeant de base.

On considérera deux nombres entiers N_1 et N_2 avec le même nombre de chiffres. Les nombres N_1 et N_2 seront représentés par un tableau de chiffres, dans l'ordre des **puissances croissantes** :

Par exemple : Si $N_1 = 4325$ alors le tableau correspondant sera :

5	2	3	4

Spécification du problème

Problème	Addition entière
Entrée	Deux entiers donnés sous la forme de tableau de nombres entre 0 et 9 de taille n
Sortie	Un tableau de taille $n + 1$ correspondant à la somme des deux nombres

Exemple du fonctionnement

Addition de $5618 + 7364 = 12802$

			i=4	i=3	i=2	i=1
N_1	=		8	1	6	5
N_2	=		4	6	3	7
retenue	=		0	1	1	·
$N_1 + N_2$	=	1	2	8	0	2

Pseudo-code :

```
ALGORITHME Addition(N1, N2)
DONNEES:
  N1, N2 : tableaux d'entiers de taille n
VARIABLES:
  i, retenue, somme : entiers
  N3 : tableau de taille n+1 initialisé à 0
DEBUT
  i ← 1
  retenue ← 0
  TQ i   n FAIRE
    somme ← N1[i] + N2[i] + retenue
    N3[i] ← (somme mod 10)
    retenue ←  somme/10
    i ← i+1
  FTQ
  N3[n+1] ← retenue
  RENVOYER N3
FIN
```

Complexité :

Boucle principale : la variable de boucle i est initialisée à 1, elle est incrémentée de 1 et va jusqu'à $n+1$. Il y aura donc n passages dans la boucle qui est constituée de 4 opérations de base. Le nombre d'opérations total est donc de :

$$3 + \sum_{i=1}^{n} 4 = 3 + 4n$$

La complexité est donc en $\Theta(n)$.

Question : Que faire si N_1 et N_2 n'ont pas la même taille ?