Tableaux : exemples (1)

- Pour le calcul du nombre d'étudiants ayant une note supérieure à 10 avec les tableaux, on peut écrire :

```
Variables       i ,nbre : entier
                tableau notes[30] : réel
Début
        nbre ← 0
        Pour i allant de 0 à 29
                Si (notes[i]  >10) alors
                        nbre ←nbre+1
                FinSi
        FinPour
        écrire ("le nombre de notes supérieures à 10 est : ", nbre)
Fin
```

Tableaux : saisie et affichage

- Procédures qui permettent de saisir et d'afficher les éléments d'un tableau :

```
Procédure SaisieTab(n : entier par valeur, tableau T : réel par référence )
 variable i: entier
   Pour i allant de 0 à n-1
      écrire ("Saisie de l'élément ",  i + 1)
      lire (T[i] )
   FinPour
Fin Procédure

Procédure AfficheTab(n : entier par valeur, tableau T : réel par valeur )
 variable i: entier
   Pour i allant de 0 à n-1
      écrire ("T[",i, "] =", T[i])
   FinPour
Fin Procédure
```

Tableaux : exemples d'appel

- Algorithme principale où on fait l'appel des procédures SaisieTab et AfficheTab :

Algorithme Tableaux
variable p : entier
tableau A[10] : réel
Début
p ← 10
SaisieTab(p, A)
AfficheTab(10,A)
Fin

Tableaux : fonction longueur

La plus part des langages offrent une fonction **longueur** qui donne la dimension du tableau. Les procédures Saisie et Affiche peuvent être réécrites comme suit :

```
Procédure SaisieTab( tableau T : réel par référence )
 variable i: entier
   Pour i allant de 0 à longueur(T)-1
      écrire ("Saisie de l'élément ",  i + 1)
      lire (T[i] )
   FinPour
Fin Procédure
Procédure AfficheTab(tableau T : réel par valeur )
 variable i: entier
   Pour i allant de 0 à longueur(T)-1
      écrire ("T[",i, "] =", T[i])
   FinPour
Fin Procédure
```

Tableaux : syntaxe Pascal

- En Maple, un tableau se définit en utilisant le type **array** comme suit :

  identificateur:= **array (a..b)**

  - Identificateur est le nom du tableau
  - a et b sont les bornes de l'indice du tableau

- Il est possible d'entrer directement toutes les valeurs d'un tableau.

  Exemple: **> A:=array(1..4,[5,8,1,7]);**

- Il est également possible de les entrer un par un comme suit :

  Exemple : **> T:=array(1..3);**
  > T[1]:=1: T[2]:=3: T[3]:=5:

- Pour afficher tous les éléments d'un tableau, il suffit d'utiliser la commande print **> print(T);** ➡ **[1, 3, 5]**

Tableaux en malpe : exemple

- Une procédure qui calcule la moyenne des éléments d'un tableau :

```
> moyenne:=proc(n,T)
> local i,s;
> s:=0;
> for i from 1 to n do
> s:=s+T[i];
> od;
> s/n;
> end;

> A:=array(1..4,[5,8,1,7]);
> moyenne(4,A);
```

résultat : 21/4

Tableaux à deux dimensions

- Les langages de programmation permettent de déclarer des tableaux dans lesquels les valeurs sont repérées par **deux indices**. Ceci est utile par exemple pour représenter des matrices

- En pseudo code, un tableau à deux dimensions se **déclare** ainsi :

  variable **tableau** identificateur**[dimension1] [dimension2] : type**

  - Exemple : une matrice A de 3 lignes et 4 colonnes dont les éléments sont réels

    variable **tableau** A**[3][4] : réel**

- **A[i][j]** permet d'accéder à l'élément de la matrice qui se trouve à l'intersection de la ligne i et de la colonne j

Exemples : lecture d'une matrice

- Procédure qui permet de saisir les éléments d'une matrice :

```
Procédure SaisieMatrice(n : entier par valeur, m : entier par valeur ,
                                                tableau A : réel par référence )
Début
 variables i,j : entier
   Pour i allant de 0 à n-1
        écrire ("saisie de la ligne ",  i + 1)
        Pour j allant de 0 à m-1
            écrire ("Entrez l'élément de la ligne ",  i + 1, " et de la colonne ", j+1)
            lire (A[i][j])
        FinPour
   FinPour
Fin Procédure
```

Exemples : affichage d'une matrice

- Procédure qui permet d'afficher les éléments d'une matrice :

Procédure AfficheMatrice(n : entier par valeur, m : entier par valeur ,**tableau** A : réel par valeur)
Début
variables i,j : entier
Pour i allant de 0 à n-1
Pour j allant de 0 à m-1
écrire ("A[",i, "] [",j,"]=", A[i][j])
FinPour
FinPour
Fin Procédure

Exemples : somme de deux matrices

- Procédure qui calcule la somme de deux matrices :

```
Procédure SommeMatrices(n, m : entier par valeur,
          tableau A, B : réel par valeur , tableau C : réel par référence )
Début
 variables i,j : entier
   Pour i allant de 0 à n-1
       Pour j allant de 0 à m-1
           C[i][j] ← A[i][j]+B[i][j]
       FinPour
   FinPour
Fin Procédure
```

Appel des procédures définies sur les matrices

Exemple d'algorithme principale où on fait l'appel des procédures définies précédemment pour la saisie, l'affichage et la somme des matrices :

Algorithme Matrices

variables **tableau** M1[3][4],M2 [3][4],M3 [3][4] : réel

Début

SaisieMatrice(3, 4, M1)

SaisieMatrice(3, 4, M2)

AfficheMatrice(3,4, M1)

AfficheMatrice(3,4, M2)

SommeMatrice(3, 4, M1,M2,M3)

AfficheMatrice(3,4, M3)

Fin

Matrices : syntaxe Maple

- Pour définir une matrice en Maple, on peut utiliser le type **array** ou le type **matrix** comme suit :

  identificateur:= **array (a1..b1, a2..b2)**

  identificateur:= **matrix(n, m)**

  - a1 et b1 sont les bornes du premier indice du tableau
  - a2 et b2 sont les bornes du deuxième indice du tableau
  - n est le nombre de lignes et m le nombre de colonnes

- Il est possible d'entrer directement toutes les valeurs d'une matrice

  Exemple: **> A:=matrix(2, 3, [[7,0,1], [2,4,3]]);**

- Le type **matrix** est disponible dans le **package linalg**. Il faut donc charger ce package avec la commande **with(linalg)** avant d'utiliser ce type

Tableaux : 2 problèmes classiques

- **Recherche d'un élément dans un tableau**
  - Recherche séquentielle
  - Recherche dichotomique
- **Tri d'un tableau**
  - Tri par sélection
  - Tri rapide

Recherche séquentielle

- Recherche de la valeur x dans un tableau T de N éléments :

```
Variables i: entier, Trouvé : booléen
...
 i←0 , Trouvé ← Faux
TantQue (i < N)  ET (Trouvé=Faux)
        Si (T[i]=x) alors
            Trouvé ← Vrai
        Sinon
             i←i+1
        FinSi
FinTantQue
Si Trouvé alors          // c'est équivalent à écrire Si Trouvé=Vrai alors
                  écrire ("x appartient au tableau")
Sinon             écrire ("x n'appartient pas au tableau")
FinSi
```

Recherche séquentielle (version 2)

- Une fonction Recherche qui retourne un booléen pour indiquer si une valeur x appartient à un tableau T de dimension N.

  x , N et T sont des paramètres de la fonction

```
Fonction Recherche(x : réel, N: entier, tableau T : réel ) : booléen
Variable i: entier
        Pour i allant de 0 à N-1
            Si (T[i]=x) alors
                retourne (Vrai)
            FinSi
        FinPour
retourne (Faux)
FinFonction
```

Notion de complexité d'un algorithme

- Pour évaluer l'**efficacité** d'un algorithme, on calcule sa **complexité**
- Mesurer la **complexité** revient à quantifier le **temps** d'exécution et l'espace **mémoire** nécessaire
- Le temps d'exécution est proportionnel au **nombre des opérations** effectuées. Pour mesurer la complexité en temps, on met en évidence certaines opérations fondamentales, puis on les compte
- Le nombre d'opérations dépend généralement du **nombre de données** à traiter. Ainsi, la complexité est une fonction de la taille des données. On s'intéresse souvent à son **ordre de grandeur** asymptotique
- En général, on s'intéresse à la **complexité** dans **le pire des cas** et à la **complexité moyenne**

Recherche séquentielle : complexité

- Pour évaluer l'efficacité de l'algorithme de recherche séquentielle, on va calculer sa complexité dans le pire des cas. Pour cela on va compter le nombre de tests effectués
- Le pire des cas pour cet algorithme correspond au cas où x n'est pas dans le tableau T
- Si x n'est pas dans le tableau, on effectue 3N tests : on répète N fois les tests (i < N), (Trouvé=Faux) et (T[i]=x)
- La **complexité** dans le pire des cas est **d'ordre N**, (on note **O(N)**)
- Pour un ordinateur qui effectue 10^6 tests par seconde on a :

N	10^3	10^6	10^9
temps	1ms	1s	16mn40s

Recherche dichotomique

- Dans le cas où le tableau est ordonné, on peut améliorer l'efficacité de la recherche en utilisant la méthode de recherche dichotomique

- **Principe :** diviser par 2 le nombre d'éléments dans lesquels on cherche la valeur x à chaque étape de la recherche. Pour cela on compare x avec T[milieu] :

  - Si x < T[milieu], il suffit de chercher x dans la 1ère moitié du tableau entre (T[0] et T[milieu-1])

  - Si x > T[milieu], il suffit de chercher x dans la 2ème moitié du tableau entre (T[milieu+1] et T[N-1])

Recherche dichotomique : algorithme

```
inf←0 , sup←N-1, Trouvé ← Faux
TantQue (inf <=sup)  ET (Trouvé=Faux)
    milieu←(inf+sup)div2
    Si (x=T[milieu]) alors
                    Trouvé ← Vrai
    SinonSi (x>T[milieu]) alors
                    inf←milieu+1
          Sinon sup←milieu-1
          FinSi
   FinSi
FinTantQue
Si Trouvé alors   écrire ("x appartient au tableau")
Sinon             écrire ("x n'appartient pas au tableau")
FinSi
```

Exemple d'exécution

- Considérons le tableau T :

4	6	10	15	17	18	24	27	30

- Si la valeur cherché est 20 alors les indices inf, sup et milieu vont évoluer comme suit :

inf	0	5	5	6
sup	8	8	5	5
milieu	4	6	5	

- Si la valeur cherché est 10 alors les indices inf, sup et milieu vont évoluer comme suit :

inf	0	0	2
sup	8	3	3
milieu	4	1	2

Recherche dichotomique : complexité

- La complexité dans le pire des cas est d'ordre $\log_2 N$

- L'écart de performances entre la recherche séquentielle et la recherche dichotomique est considérable pour les grandes valeurs de N

  - Exemple: au lieu de N=1milion ≈2^{20} opérations à effectuer avec une recherche séquentielle il suffit de 20 opérations avec une recherche dichotomique

Tri d'un tableau

- Le tri consiste à ordonner les éléments du tableau dans l'ordre croissant ou décroissant

- Il existe plusieurs algorithmes connus pour trier les éléments d'un tableau :

  - Le tri par sélection
  - Le tri par insertion
  - Le tri rapide
  - …

- Nous verrons dans la suite l'algorithme de tri par sélection et l'algorithme de tri rapide. Le tri sera effectué dans l'ordre croissant

Tri par sélection

- **Principe** : à l'étape i, on sélectionne le plus petit élément parmi les (n - i +1) éléments du tableau les plus à droite. On l'échange ensuite avec l'élément i du tableau
- **Exemple :**

9	4	1	7	3

  - **Étape 1:** on cherche le plus petit parmi les 5 éléments du tableau. On l'identifie en troisième position, et on l'échange alors avec l'élément 1 :

1	4	9	7	3

  - **Étape 2:** on cherche le plus petit élément, mais cette fois à partir du deuxième élément. On le trouve en dernière position, on l'échange avec le deuxième:

1	3	9	7	4

  - **Étape 3:**

1	3	4	7	9

Tri par sélection : algorithme

- Supposons que le tableau est noté T et sa taille N

```
Pour i  allant de 0 à N-2
        indice_ppe ← i
        Pour j allant de i + 1 à N-1
            Si T[j] <T[indice_ppe] alors
                indice_ppe  ← j
            Finsi
        FinPour

        temp ← T[indice_ppe]
        T[indice_ppe] ← T[i]
        T[i] ← temp
FinPour
```

Tri par sélection : complexité

- Quel que soit l'ordre du tableau initial, le nombre de tests et d'échanges reste le même

- On effectue N-1 tests pour trouver le premier élément du tableau trié, N-2 tests pour le deuxième, et ainsi de suite. Soit : (N-1)+(N-2)+…+1 = N(N-1)/2
  On effectue en plus (N-1) échanges.

- La **complexité** du tri par sélection est **d'ordre N²** à la fois dans le meilleur des cas, en moyenne et dans le pire des cas

- Pour un ordinateur qui effectue 10^6 tests par seconde on a :

N	10^3	10^6	10^9
temps	1s	11,5 jours	32000 ans

Tri rapide

- Le tri rapide est un tri récursif basé sur l'approche "diviser pour régner" (consiste à décomposer un problème d'une taille donnée à des sous problèmes similaires mais de taille inférieure faciles à résoudre)

- **Description du tri rapide :**

  - **1)** on considère un élément du tableau qu'on appelle pivot

  - **2)** on partitionne le tableau en 2 sous tableaux : les éléments inférieurs ou égaux à pivot et les éléments supérieurs à pivot. on peut placer ainsi la valeur du pivot à sa place définitive entre les deux sous tableaux

  - **3)** on répète récursivement ce partitionnement sur chacun des sous tableaux crées jusqu'à ce qu'ils soient réduits à un à un seul élément

Procédure Tri rapide

Procédure TriRapide(tableau **T** : réel par adresse, **p,r**: entier par valeur)

variable q: entier

Si p <r **alors**

Partition(T,p,r,q)

TriRapide(T,p,q-1)

TriRapide(T,q+1,r)

FinSi

Fin Procédure

A chaque étape de récursivité on partitionne un tableau T[p..r] en deux sous tableaux T[p..q-1] et T[q+1..r] tel que chaque élément de T[p..q-1] soit inférieur ou égal à chaque élément de A[q+1..r] . L'indice q est calculé pendant la procédure de partitionnement

Procédure de partition

```
Procédure Partition(tableau T : réel par adresse, p,r: entier par valeur,
                                                   q: entier par adresse )
  Variables  i, j: entier
             pivot: réel
    pivot← T[p], i←p+1, j ← r
    TantQue (i<=j)
          TantQue (i<=r et T[i] <=pivot)     i ← i+1  FinTantQue
          TantQue (j>=p et T[j] >pivot )  j ← j-1    FinTantQue
          Si  i <j  alors
                    Echanger(T[i], T[j]), i ← i+1, j ← j-1
          FinSi
    FinTantQue
     Echanger(T[j], T[p])
    q ← j
Fin Procédure
```

Tri rapide : complexité et remarques

- La complexité du tri rapide dans le pire des cas est en $O(N^2)$
- La complexité du tri rapide en moyenne est en O(N log N)
- Le choix du pivot influence largement les performances du tri rapide
- Le pire des cas correspond au cas où le pivot est à chaque choix le plus petit élément du tableau (tableau déjà trié)
- différentes versions du tri rapide sont proposés dans la littérature pour rendre le pire des cas le plus improbable possible, ce qui rend cette méthode la plus rapide en moyenne parmi toutes celles utilisées