Les tableaux

Gilles Trombettoni

IUT MPL-Sète, département info

Développement initiatique

Octobre 2021

Définition de tableau

Définition (wikipedia)

Un tableau est une séquence finie d'éléments auxquels on peut accéder efficacement par leur position, ou **indice**, dans la séquence.

Un tableau « de base » contient des valeurs qui sont toutes du même type.

3.14	2.72	1.62	-1.655	-10.54
0	1	2	3	4

Les tableaux dans le langage maison

Déclaration (dans la partie `Variables`)

- `<nom de tableau> : tableau de <taille> <type des éléments>`

  ⇒ définit un tableau de `<taille>` éléments de type `<type des éléments>` indicés de 0 à `<taille>`−1.

- Exemple de déclaration d'un tableau de 10 caractères (dans un algo `machin`) :

  ```
  tab : tableau de 10 caractères
  ```

- Exemple de déclaration, dans une fonction `truc`, d'un tableau de caractères, de taille quelconque :

  ```
  fonction truc (t : tableau de caractères)...
  ```

  Si l'algo `machin` appelle la fonction `truc(tab)`, cela copie :

  - les éléments (valeurs) de `tab` dans le tableau `t` en paramètre de `truc` et
  - la valeur 10 dans le paramètre `t.longueur`.

Accès aux cases d'un tableau

- Accès à une case :
  `<nom de tableau>[<indice>]`
  Exemple : `t[2]`

- Accès en écriture (affectation) :
  `<nom de tableau>[<indice>]` ← `<valeur>`
  Exemple : `t[2]` ← `'z'`

- Accès en lecture (dans un test) :
  Exemple : `si t[2] mod 2 == 0 alors ...`

Les tableaux en mémoire

- Un tableau est stocké en mémoire vive dans une zone contiguë.
- Un tableau est du coup déterminé par l'adresse de sa première case.
- L'accès à chaque case prend le même temps de calcul (premier élément ou 1000^e élément), très rapide : une multiplication, une addition, un accès à une adresse mémoire !

Principe de parcours partiel

Définition

- Définition : parcours possible de seulement quelques éléments du tableau.
- But : savoir s'il existe au moins un élément vérifiant une condition.
- Deux conditions d'arrêt : vérification de la condition **ou** parcours de tout le tableau (sans vérifier la condition).

Pseudo-code générique

```
conditionVerifiee <- faux
tantQue <il existe un elt du tableau non encore visité> ET
        non conditionVerifiee
faire
   si <condition vérifiée> alors
      conditionVerifiee <- vrai
   sinon
      <passer à l'élément suivant>
   finSi
fin tantQue
/* conditionVerifiee indique si un elt vérifie la condition */
```

Exemple de parcours partiel

Une fonction `contient` qui recherche un élément réel donné `elt` dans un tableau `t`. `contient` retourne vrai ssi l'élément est trouvé.

Fonction `contient`

```
fonction contient (elt : reel, t : tableau de reels)
                  retourne booleen
Variables
   i : entier ; trouve : booleen
Debut
   i <- 0 ; trouve <- faux
   tantQue i < t.longueur ET non trouve faire
      si t[i] == elt alors // elt trouvé
         trouve <- vrai
      sinon
         i <- i+1
      finSi
   finTantQue
   retourne trouve
Fin contient
```

Qu'est-ce qu'un tableau dynamique ?

Un **tableau dynamique** est une structure de données concrète (en mémoire) qui implante à la fois les types abstraits « tableau » et « liste ».

(Les listes seront vues plus tard cette année. Elles permettent un ajout ou une suppression rapide d'un élément, au début et/ou en fin de liste.)

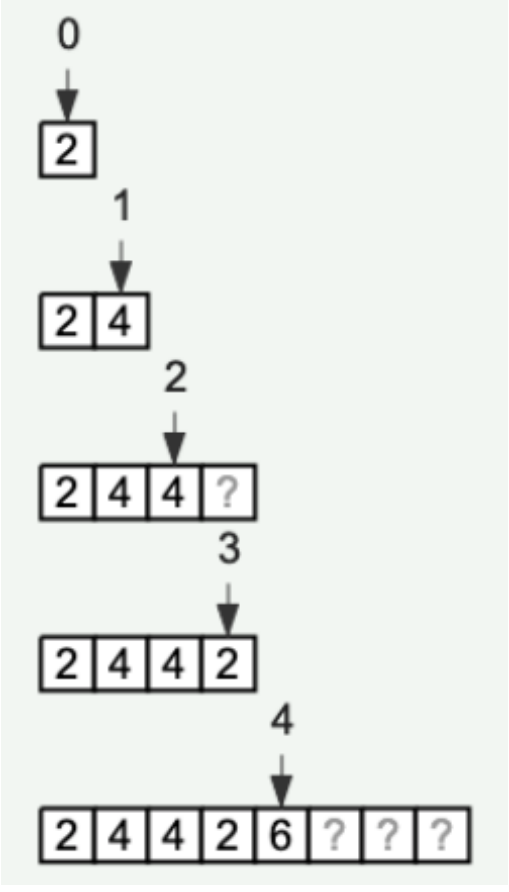
Tableaux dynamiques dans quelques langages de programmation

C++	`vector`
Java	`ArrayList`
Python	« liste »

Un tableau dynamique est presque un tableau

- Comme un tableau, un vecteur est implanté par des éléments contigüs en mémoire.
- Comme un tableau, l'accès à n'importe quel élément par son indice se fait en temps rapide, le même temps pour chaque élément.
- (Dans de nombreux langages, un vecteur peut contenir des objets de types différents, mais en fait ce sont les *références* des éléments (adresses, pointeurs) qui sont stockés...)
- Pour gérer un ajout rapide d'un élément en fin de vecteur, un vecteur est défini par la taille `nMax` du tableau sous-jacent et un indice supplémentaire `iMax` :
  - Les données du vecteur sont comprises entre les indices 0 et `iMax`.
  - Les cases d'indices `iMax` et `nMax` sont utilisées en cas d'ajout de nouveaux éléments.
  - En cas de « dépassement », le tableau est copié dans un tableau plus grand de taille $a \times$ `nMax` (ex : $a = 2$ ou 1.25).

Un tableau dynamique est un tableau nomade



Pour en savoir plus :

- `http://mpechaud.fr/scripts/donnees/listestableaux.html`
- `https://wiki.python.org/moin/TimeComplexity`

Tableaux de tableaux

- Pour un tableau donné, chaque case peut contenir un autre tableau ! On parle de tableau multi-dimensionnel.
- Si les tableaux dans chaque case sont tous du même type et de même taille, on décrit une matrice.
- Déclaration :

  ```
  mat : tableau de n (tableaux de m entiers)
  ```

  Ou bien :

  ```
  mat : matrice de n lignes et m colonnes entières
  ```

- Convention : `mat` est un tableau de n « lignes » contenant chacune m « colonnes ».
- Sémantique : allocations de $n * m$ éléments entiers.
- On peut généraliser, en dimension 3 par exemple :

  ```
  mat3d : tableau de n (tableaux de m (tableaux
  de p entiers))
  ```

- Représentation mémoire ?

Parcours total d'une matrice (2D)

Parcours des $n \times m$ éléments d'une matrice `mat` avec deux boucles imbriquées

```
pour i dans 0..(n-1) faire  // parcours des lignes
   pour j dans 0..(m-1) faire
       // parcours des colonnes j d'une ligne i :
       afficher mat[i][j]
   finPour
   sautLigne()
finPour
```

Parcours des $n \times m$ éléments avec une seule boucle

```
i <-0 ; j <- 0
tantQue i < n faire
   afficher mat[i][j]
   si j < (m-1) alors
       j <- j+1 // colonne suivante
   sinon // j == m-1
      sautLigne()
      i <- i+1  // ligne suivante
      j <- 0    // on revient à la première colonne
   finSi
fin TantQue
```

Parcours partiel d'une matrice (2D)

Exemple : recherche d'un élément donné dans une matrice

```
fonction contient (elt: reel, mat: tableau de tableaux de reels)
                  retourne booleen
Variables
   i,j : entier ; trouve : booleen
Debut
  i <- 0 ; trouve <- faux
  tantQue i < mat.longueur et non trouve faire
     // parcours des lignes :
     j <- 0
     tantQue j < mat[i].longueur et non trouve faire
        // parcours des colonnes j de la ligne i :
        si mat[i][j] == elt alors
           trouve <- vrai
        finSi
        j <- j+1
     fin tantQue
     i <- i+1
  fin tantQue
  retourne trouve
Fin contient
```

Parcours partiel dans une matrice (2D) : carré magique

Exemple : vérification qu'un carré est magique

```
fonction estMagique (mat : tableau de tableaux d'entiers)
                    retourne booleen
Variables
  magique                : booleen
  i, j, sommeReference : entier
Début
  sommeReference <- sommeLigne(mat, 0) // somme premiere ligne
  i <- 1 ; magique <- vrai
  tantQue i < mat.longueur et magique faire // lignes magiques ?
     magique <- sommeReference == sommeLigne(mat, i)
     si magique alors i <- i+1 finSi
  fin tantQue
  j <- 0
  tantQue j < mat.longueur et magique faire // colonnes magiques?
     magique <- sommeReference == sommeColonne(mat, j)
     si magique alors j <- j+1 finSi
  fin tantQue
  retourne magique etAlors verifDiagonales(mat, sommeReference)
                   etAlors verifPremiersEntiers(mat)
Fin estMagique
```

Parcours partiel dans une matrice : carré magique

Fonction sommeLigne

```
fonction sommeLigne(matr: tableau de tableaux d'entiers,
                    ligne: entier) retourne entier
Variables
  j, somme : entier
Début
  somme <- 0
  pour j dans 0..(matr[ligne].longueur-1) faire
    somme <- somme + matr[ligne][j]
  fin Pour
  retourne somme
Fin sommeLigne
```

Parcours partiel dans une matrice : carré magique

Fonction sommeColonne

```
fonction sommeColonne(matr: tableau de tableaux d'entiers,
                      col: entier) retourne entier
Variables
  i, somme : entier
Début
  somme <- 0
  pour i dans 0..(matr.longueur-1) faire
    somme <- somme + matr[i][col]
  fin Pour
  retourne somme
Fin sommeLigne
```