Tableaux multidimensionnels [tm]
Support de Cours

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Version 18 mai 2018

Table des matières

C - Tableaux multidimensionnels (Cours)

Mots-Clés Tableaux multidimensionnels, Parcours d'un tableau ■
Requis Structures de base, Structures conditionnelles, Algorithmes paramétrés, Structures répétitives, Schéma itératif, Tableau unidimensionnel ■
Difficulté ●●○ (1 h) ■

Introduction
Ce module décrit des **tableaux multidimensionnels** puis expose les parcours ainsi que la vue linéaire d'un tableau bidimensionnel.

1 Tableaux multidimensionnels

1.1 Définitions

De nombreuses situations nécessitent l'usage de tableaux à deux dimensions. Ils vous sont déjà familiers par leur présence dans nombre de situations courantes : calendrier, grille horaire, grille de mots croisés, sudoku, jeux se déroulant sur un quadrillage (damier, échiquier, scrabble, ...). Cependant certaines situations complexes nécessitent l'usage de tableaux à trois voire plus de dimensions.

Tableau multidimensionnel
Tableau dont le type de base est un tableau : c'est un « tableau de tableaux ».

Dimension
Nombre d'indices utilisé pour faire référence à un des éléments d'un tableau multidimensionnel.

Taille et Dimension
Ne confondez pas :

- La taille (= nombre de cases).
- La dimension (= nombre d'indices).

Remarque
On peut faire une analogie avec les mathématiques :

- Un tableau à une dimension représente un vecteur.
- Un tableau à deux dimensions une matrice.
- Un tableau de plus de deux dimensions un tenseur.

1.2 Déclaration et initialisation d'un k-tableau

Déclaration d'un k-tableau

```
T nomTab[taille1][taille2][...][tailleK];
```

Explication

Déclare un tableau k-dimensionnel de nom `nomTab` d'éléments de type `T`. Les `tailleI` sont des valeurs entières positives (littérales ou expressions constantes).

Cas particulier d'un tableau bidimensionnel

```
T nomTab[taille1][taille2];
```

Déclaration et initialisation

```
T nomTab[taille1]...[tailleK] = {{{v111, ..., v11k}, ...}; // dim explicite
T nomTab[ ]...[ ] = {{{v111, ..., v11k}, ...}; // dim implicite
```

Explication

Déclare et initialise un tableau k-dimensionnel. Dans le cas (2), la dimension de la composante est la longueur de la liste.

1.3 Accès indiciel

Accès indiciel

```
tab[k1][k2][...]
```

Explication

Accède à la case indice `(k1, k2, ...)` d'un tableau multidimensionnel `tab`. Il faut spécifier autant d'indices qu'il y a de dimensions dans le tableau.

Crochet de crochets

Un tableau k-dimensionnel est vu comme un tableau à une dimension dont chacun des éléments est lui-même un tableau à $(k-1)$-dimension.

Rappel

Les indices sont relatifs à 0.

1.4 Exemples

C/C++ **Exemple C/C++**

Soit le tableau déclaré ainsi :

```
char tab[4][5];
```

Explication

Le tableau `tab` est d'abord décomposé à un premier niveau en quatre éléments auxquels on accède par le premier indice. Ensuite, chaque élément de premier niveau est décomposé en cinq éléments de deuxième niveau accessibles par le deuxième indice.

C/C++ **Exemple C/C++**

```
int mat[3][5]
double cube[10][20][15]
```

Explication

La variable `mat` est une matrice de 3 lignes et 5 colonnes d'`Entier`s et `cube` est un cube de 15 matrices de 10 lignes et 20 colonnes de `Réel`s.

C/C++ **Exemple C/C++**

```
mat[2][1]
```

Explication

Accède à l'élément ligne 2, colonne 1 de la matrice `mat`.

2 Parcours d'un tableau bidimensionnel

Comme nous l'avons fait pour les tableaux à une dimension, nous envisageons le parcours des tableaux à deux dimensions (`nligs` lignes et `ncols` colonnes) d'éléments de type `T`.

Nous commençons par le cas où on ne parcourt qu'une seule des dimensions (parcours d'une ligne, parcours d'une colonne, parcours d'une diagonale si le tableau est carré `nligs=ncols=n`) puis nous attaquons le cas général (parcours complet ligne par ligne, colonne par colonne, parcours partiel puis un parcours plus compliqué : le serpent).

Déclarations C

Soit donc un tableau `tab` déclaré ainsi :

```
enum {NLIGSMAX = ..., NCOLSMAX = ...};
typedef T Tableau2d[NLIGSMAX][NCOLSMAX] // avec T un Type quelconque
Tableau2d tab;
```

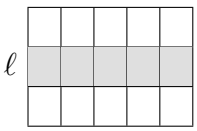
2.1 Parcours d'une ligne/colonne

Parcours ligne lig

```
int col;
for (col=0; col<ncols; ++col)
{
  traiter tab[lig][col];
}
```

Explication

Si on parcourt la ligne ℓ, on visite les cases $(\ell, 1)$, $(\ell, 2)$, $(\ell, 3)$... L'indice de ligne est constant et c'est l'indice de colonne qui varie.



Retenir

Pour parcourir une ligne, on utilise une boucle sur les colonnes.

Parcours colonne col

```
int lig;
for (lig=0; lig<nligs; ++lig)
{
  traiter tab[lig][col];
}
```

Explication

Le parcours d'une colonne est symétrique.

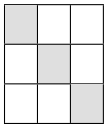
2.2 Parcours d'une diagonale

Parcours diagonale descendante

```
int j;
for (j=0; j<n; ++j)
{
  traiter tab[j][j];
}
```

Explication

Les éléments à visiter sont $(1,1)$, $(2,2)$, …, (n,n).
Une seule boucle suffit comme le montre l'algorithme ci-dessus.



Parcours diagonale montante, 2 indices

```
col = n-1;
int lig;
for (lig=0; lig<n; ++lig)
{
  traiter tab[lig][col];
  --col;
}
```

Parcours diagonale montante, 1 indice

```
int lig;
for (lig=0; lig<n; ++lig)
{
  traiter tab[lig][n-lig];
}
```

Explication

Pour la diagonale montante, on peut envisager deux solutions, avec deux indices ou un seul en se basant sur le fait que $i + j = n + 1$ d'où $j = n + 1 - i$.

2.3 Parcours complet

Parcours ligne par ligne

```
int lig,col;
for (lig=0; lig<nligs; ++lig)
{
  for (col=0; col<ncols; ++col)
  {
    traiter tab[lig][col];
  }
}
```

Explication

Chaque case est visitée comme suit :

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

Parcours colonne par colonne

```
int lig,col;
for (col=0; col<ncols; ++col)
{
  for (lig=0; lig<nligs; ++lig)
  {
    traiter tab[lig][col];
  }
}
```

Explication

Ici chaque case est visitée ainsi :

1	4	7	10	13
2	5	8	11	14
3	6	9	12	15

Remarque

On peut obtenir le même résultat avec une seule boucle si l'indice sert juste à compter le nombre de passages et que les indices de lignes et de colonnes sont gérés manuellement. L'algorithme ci-dessous donne le parcours ligne par ligne. La solution pour un parcours colonne par colonne est similaire et laissée en exercice.

Parcours avec une seule boucle

```
lig = 1;
col = 1;
ncases = nligs * ncols;
int k;
for (k=0; k<ncases; ++k)
{
  traiter tab[lig][col];
  ++col;
  if (col>=ncols)
  {
    col=0;
    ++lig;
  }
}
```

Avantage de cette solution
Il apparaitra quand nous verrons des parcours plus difficiles, tel le serpent.

2.4 Parcours partiel

Comme avec les tableaux à une dimension, nous envisageons l'arrêt prématuré lors de la rencontre d'une condition. Et, comme avec les tableaux à une dimension, on transforme d'abord les `Pour` en `TantQue`, puis on introduit le test : avec ou sans utiliser de booléen.

Parcours ligne par ligne, boucle TantQue

```
lig = 0;
while (lig<nligs)
{
  col = 0;
  while (col<ncols)
  {
    traiter tab[lig][col];
    ++col;
  }
  ++lig;
}
```

Parcours partiel, deux boucles et un booléen

```
trouve = false;
lig = 0;
while (lig<nligs && !trouve)
{
  col = 0;
  while (col<ncols && !trouve)
  {
    if (tab[lig][col] impose l'arrêt du parcours)
    {
      trouve=true;
    }
    else
    {
      ++col;
    }
  }
  if (!trouve)
  {
    ++lig;
  }
}
```

Parcours partiel, une seule boucle et un TantQue

```
lig = 0;
col = 0;
ncases = nligs * ncols;
k = 0;
while (k<ncases)
{
  traiter tab[lig][col];
  ++col;
```

```
  if (col>=ncols)
  {
    col=0;
    ++lig;
  }
  ++k;
}
```

Parcours partiel, une boucle et pas de booléen

```
lig = 0;
col = 0;
ncases = nligs * ncols;
k = 0;
while (k<ncases && tab[lig, col] n'impose pas l'arrêt)
{
  ++col;
  if (col>=ncols)
  {
    col=0;
    ++lig;
  }
  ++k;
}
//Arrêt prématuré Si k < ncases
```

2.5 Parcours du serpent

Envisageons maintenant un parcours plus difficile illustré par le tableau suivant :

1	2	3	4	5
10	9	8	7	6
11	12	13	14	15

Le plus simple est d'adapter l'algorithme de parcours avec une seule boucle en introduisant un sens de déplacement.

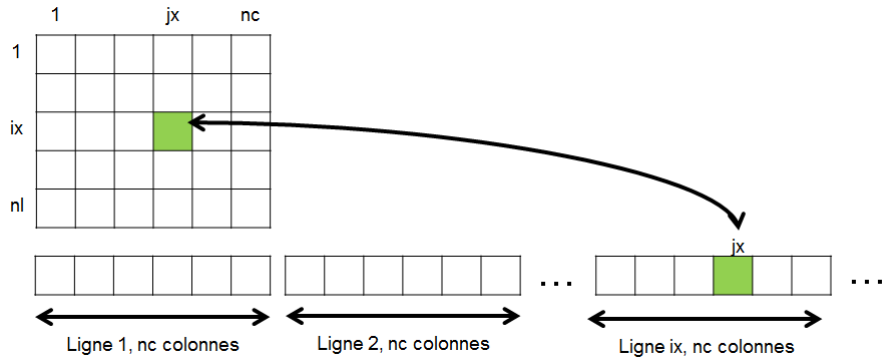
Parcours du serpent

```
lig = 0;
col = 0;
ncases = nligs * ncols;
depl = 1; // 1 Pour Avancer , -1 Pour reculer
while (k<ncases)
{
  traiter tab[lig][col];
  if (0 <= col + depl Et col + depl < ncols)
  {
    col += depl;
  }
  else
  {
    ++lig;
    depl = -depl;
  }
}
```

3 Vue linéaire

Stockage linéaire d'un tableau bidimensionnel

Le **stockage linéaire** d'un tableau bidimensionnel permet d'optimiser son espace mémoire. L'accès à l'élément en `(ix,jx)` du tableau bidimensionnel se fera via une fonction `indexTab2d(ix,jx,ncols)` qui donne l'index linéaire de la case en `(ix,jx)` pour un tableau à `ncols` colonnes.



Remarque

La vue linéaire d'un tableau bidimensionnel s'étend facilement (par le même principe) au cas des tableaux multidimensionnels.