Les tableaux à plusieurs dimensions

Christian Lasou, Nour-Eddine Oussous, Éric Wegrzynowski

Licence ST-A, USTL - API1

26 mars 2007

Motivations

Dans différentes applications, on utilise des tableaux à deux dimensions (deux entrées). Par exemple :

Motivations

Dans différentes applications, on utilise des tableaux à deux dimensions (deux entrées). Par exemple :

1. les matrices (en algèbre linéaire)

Motivations

Dans différentes applications, on utilise des tableaux à deux dimensions (deux entrées). Par exemple :

1. les matrices (en algèbre linéaire)
2. la tabulation des fonctions à deux variables

Motivations

Dans différentes applications, on utilise des tableaux à deux dimensions (deux entrées). Par exemple :

1. les matrices (en algèbre linéaire)
2. la tabulation des fonctions à deux variables
3. les mots croisés

Motivations

Dans différentes applications, on utilise des tableaux à deux dimensions (deux entrées). Par exemple :

1. les matrices (en algèbre linéaire)
2. la tabulation des fonctions à deux variables
3. les mots croisés
4. le soduku

Motivations

Dans différentes applications, on utilise des tableaux à deux dimensions (deux entrées). Par exemple :

1. les matrices (en algèbre linéaire)
2. la tabulation des fonctions à deux variables
3. les mots croisés
4. le soduku
5. les carrés magiques
6. ...

Déclaration de tableaux à deux dimensions

Le type d'un tableau à deux dimensions

Déclaration de tableaux à deux dimensions

Le type d'un tableau à deux dimensions

```
array[INDICE1] of array[INDICE2] of ELEMENT
```

Déclaration de tableaux à deux dimensions

Le type d'un tableau à deux dimensions

```
array[INDICE1] of array[INDICE2] of ELEMENT
```

ou plus simplement

```
array[INDICE1,INDICE2] of ELEMENT
```

Déclaration de tableaux à deux dimensions

Le type d'un tableau à deux dimensions

```
array[INDICE1] of array[INDICE2] of ELEMENT
```

ou plus simplement

```
array[INDICE1,INDICE2] of ELEMENT
```

les deux déclarations sont équivalentes.

Exemple

Un tableau à deux dimensions

```
array['A'..'C', 1..5] of CARDINAL
```

Exemple

Un tableau à deux dimensions

```
array['A'..'C', 1..5] of CARDINAL
```

	1	2	3	4	5
A					
B					
C					

Tab.: Représentation d'un tableau à deux dimensions

Exemple

Un tableau à deux dimensions

```
array['A'..'C', 1..5] of CARDINAL
```

	1	2	3	4	5
A					
B					
C					

Tab.: Représentation d'un tableau à deux dimensions

Par convention, on conviendra que

- le premier indice désigne la ligne ;
- le second désigne la colonne.

Notation indicielle

Avec la déclaration

```
var
   t : array[INDICE1,INDICE2] of ELEMENT;
```

Notation indicielle

Avec la déclaration

```
var
   t : array[INDICE1,INDICE2] of ELEMENT;
```

1. `t[i,j]` désigne le terme de la ligne *i* colonne *j*. C'est un `ELEMENT`.

Notation indicielle

Avec la déclaration

```
var
   t : array[INDICE1,INDICE2] of ELEMENT;
```

1. `t[i,j]` désigne le terme de la ligne *i* colonne *j*. C'est un `ELEMENT`.
2. `t[i][j]` est une notation équivalente à la précédente.

Notation indicielle

Avec la déclaration

```
var
   t : array[INDICE1,INDICE2] of ELEMENT;
```

1. `t[i,j]` désigne le terme de la ligne i colonne j. C'est un `ELEMENT`.
2. `t[i][j]` est une notation équivalente à la précédente.
3. `t[i]` désigne la ligne i de t. C'est une valeur de type `array[INDICE2] of ELEMENT`.

Notation indicielle

Avec la déclaration

```
var
   t : array[INDICE1,INDICE2] of ELEMENT;
```

1. `t[i,j]` désigne le terme de la ligne i colonne j. C'est un `ELEMENT`.
2. `t[i][j]` est une notation équivalente à la précédente.
3. `t[i]` désigne la ligne i de t. C'est une valeur de type `array[INDICE2] of ELEMENT`.
4. `length(t)` = nbre d'indices dans l'intervalle `INDICE1`.

Notation indicielle

Avec la déclaration

```
var
   t : array[INDICE1,INDICE2] of ELEMENT;
```

1. `t[i,j]` désigne le terme de la ligne i colonne j. C'est un `ELEMENT`.
2. `t[i][j]` est une notation équivalente à la précédente.
3. `t[i]` désigne la ligne i de t. C'est une valeur de type `array[INDICE2] of ELEMENT`.
4. `length(t)` = nbre d'indices dans l'intervalle `INDICE1`.
5. `length(t[i])` = nbre d'indices dans l'intervalle `INDICE2`.

Notation indicielle

Avec la déclaration

```
var
   t : array[INDICE1,INDICE2] of ELEMENT;
```

1. `t[i,j]` désigne le terme de la ligne i colonne j. C'est un `ELEMENT`.
2. `t[i][j]` est une notation équivalente à la précédente.
3. `t[i]` désigne la ligne i de t. C'est une valeur de type `array[INDICE2] of ELEMENT`.
4. `length(t)` = nbre d'indices dans l'intervalle `INDICE1`.
5. `length(t[i])` = nbre d'indices dans l'intervalle `INDICE2`.
6. il en va de même pour les fonctions `low` et `high`.

Un programme exemple

Listing

```
program exemple;
type
   TABLEAU = array['A'..'C',1..5] of CARDINAL;
const
   t : TABLEAU = ((1,2,3,4,5),
                  (6,7,8,9,10),
                  (11,12,13,14,15));
begin
   writeln(t['A',1]);          // affiche 1
   writeln(t['A'][1]);         // idem
   writeln(length(t));         // affiche 3
   writeln(length(t['A']));    // affiche 5
   writeln(low(t));            // affiche A
   writeln(low(t['A']));       // affiche 1
   writeln(high(t));           // affiche C
   writeln(high(t['A']));      // affiche 5
end.
```

Les schémas d'algorithmes qui suivent s'appliquent à tout tableau bidimensionnel de la forme

```
t : array[a..b,c..d] of ELEMENT;
```

Parcourir une ligne

Schéma de parcours séquentiel complet (de gauche à droite) de la ligne i d'un tableau bidimensionnel.

```
pour j variant de c à d faire
    traiter t[i,j]
fin pour
```

Parcourir une colonne

Schéma de parcours séquentiel complet (de haut en bas) de la colonne j d'un tableau bidimensionnel.

```
pour i variant de a à b faire
    traiter t[i,j]
fin pour
```

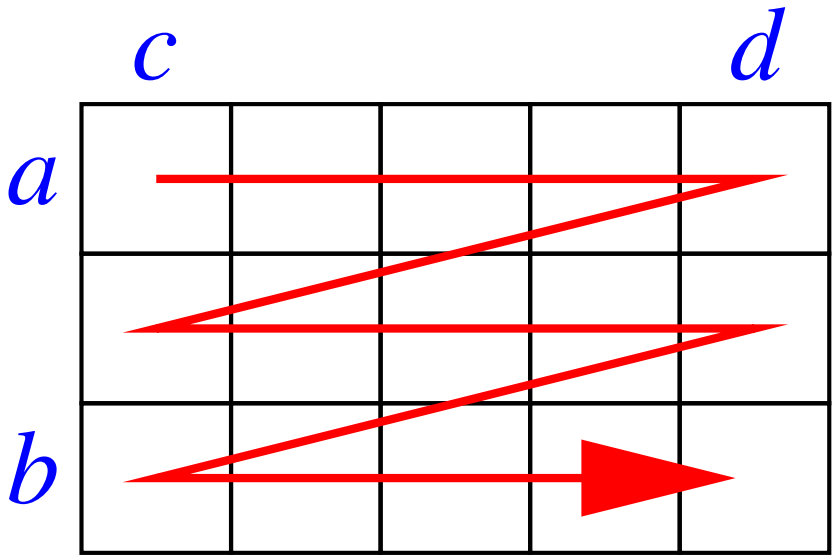
Parcourir complet (1/2)

Schéma de parcours complet d'un tableau bidimensionnel, ligne par ligne.

```
pour i variant de a à b faire
    pour j variant de c à d faire
        traiter t[i,j]
    fin pour
fin pour
```

(deux boucles imbriquées)

Parcourir complet : illustration



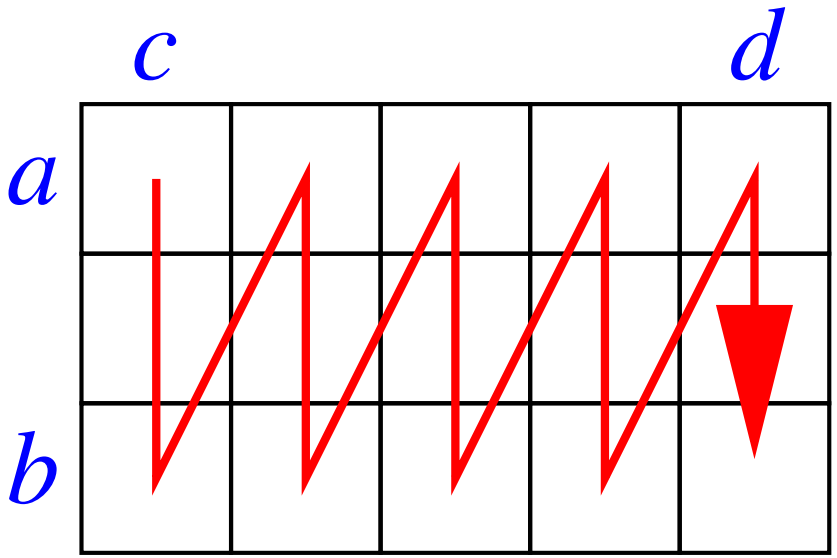
Parcourir complet (2/2)

Schéma de parcours complet d'un tableau bidimensionnel, colonne par colonne.

```
pour j variant de c à d faire
    pour i variant de a à b faire
        traiter t[i,j]
    fin pour
fin pour
```

(deux boucles imbriquées)

Parcourir complet : illustration



Généralisation

Tableaux à plusieurs dimensions !

On peut définir un tableau à n dimensions comme étant un tableau à une dimension dont les éléments sont des tableaux à $n-1$ dimensions.

Généralisation

Tableaux à plusieurs dimensions !

On peut définir un tableau à n dimensions comme étant un tableau à une dimension dont les éléments sont des tableaux à $n-1$ dimensions.

Tableau à 3 dimensions

```
type
    TABLEAUX3DIM = array[INDICE] of
                       array[INDICE,INDICE] of
                           ELEMENT ;
```

Le problème et sa représentation

Définition

Un carré magique d'ordre n est un tableau carré de n lignes et n colonnes, comportant n^2 entiers naturels et tel que

- la somme des n entiers de chaque ligne est égale à ...
- ... la somme des n entiers de chaque colonne et à
- ... la somme des n entiers de chaque diagonale

Le problème et sa représentation

Définition

Un carré magique d'ordre n est un tableau carré de n lignes et n colonnes, comportant n^2 entiers naturels et tel que

- la somme des n entiers de chaque ligne est égale à ...
- ... la somme des n entiers de chaque colonne et à
- ... la somme des n entiers de chaque diagonale

La somme obtenue sur chaque ligne, colonne ou diagonale est appelée constante magique.

Le problème et sa représentation

Définition

Un carré magique d'ordre n est un tableau carré de n lignes et n colonnes, comportant n^2 entiers naturels et tel que

- la somme des n entiers de chaque ligne est égale à ...
- ... la somme des n entiers de chaque colonne et à
- ... la somme des n entiers de chaque diagonale

La somme obtenue sur chaque ligne, colonne ou diagonale est appelée constante magique.

But

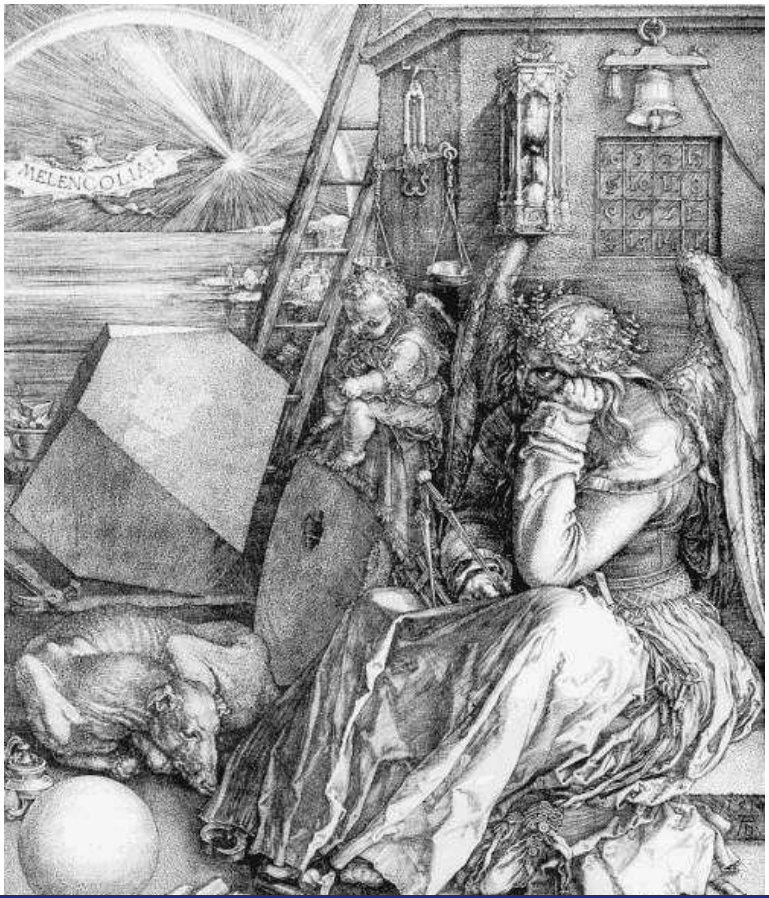
Écrire des procédures de lecture et d'affichage des carrés magiques et un prédicat qui teste si le carré passé en argument est un carré magique ou non.

Exemple de carrés magiques (1/2)

La carré magique de la gravure de Dürer «La mélancolie»

Exemple de carrés magiques (1/2)

La carré magique de la gravure de Dürer «La mélancolie»

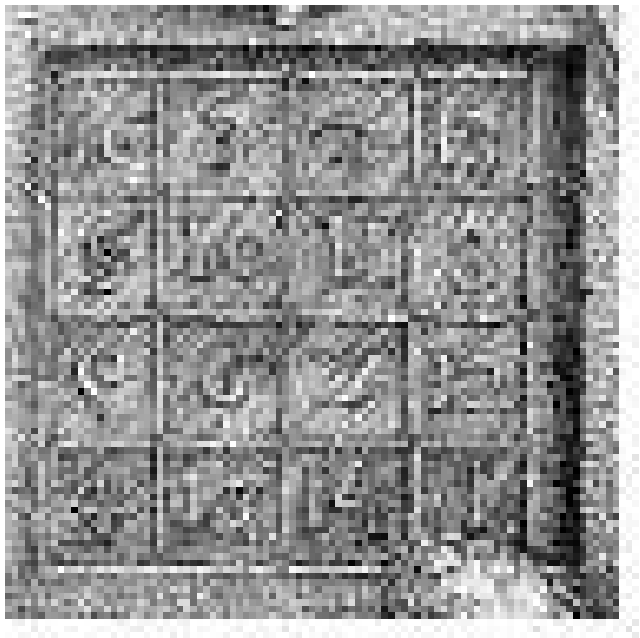


Exemple de carrés magiques (1/2)

La carré magique de la gravure de Dürer «La mélancolie»

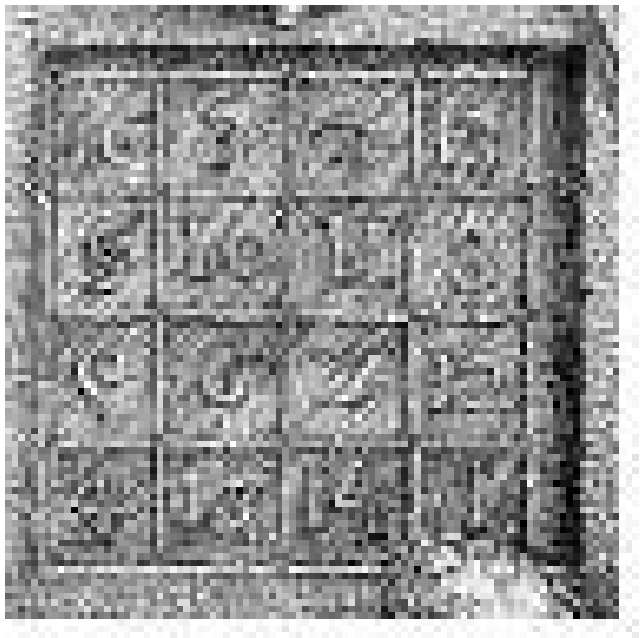
Exemple de carrés magiques (1/2)

La carré magique de la gravure de Dürer «La mélancolie»



Exemple de carrés magiques (1/2)

La carré magique de la gravure de Dürer «La mélancolie»



16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

Le problème et sa représentation

On va utiliser un tableau à deux dimensions pour représenter un carré.

Le problème et sa représentation

On va utiliser un tableau à deux dimensions pour représenter un carré.

Listing

```
const
  MAX = 31 ;    // taille max du carre
  FORMAT = 4 ; // format pour l'affichage
type
  INDICE = 0..MAX-1 ;
  INDICE_ETENDU = 0..MAX ;
  TAILLE = 3..MAX ;
  CARRE = record
     taille  : TAILLE ;
     contenu : array[INDICE,INDICE] of CARDINAL ;
  end {CARRE} ;
```

Lecture d'un carré

Listing

```
// lireCarre(c,t) lit le contenu du carre c
// de taille t
procedure lireCarre(out c: CARRE;
                    const t: TAILLE);
var
  i,j : INDICE;
begin
  c.taille := t;
  for i := low(INDICE) to t-1 do begin
    for j := low(INDICE) to t-1 do
      read(c.contenu[i,j]);
  end {for};
end {lireCarre};
```

Lecture d'un carré

Remarque

Noter l'utilisation de la procédure `read` au lieu de `readln`. Cela permet de lire les valeurs des données dans des fichiers de texte dans lesquels les éléments d'un carré sont disposés sous forme de carré.

Affichage d'un carré

Listing

```
// ecrireCarre(c) affiche le contenu du carre
procedure ecrireCarre(const c: CARRE);
var
  i,j : INDICE;
begin
  for i := low(INDICE) to c.taille-1 do begin
    for j := low(INDICE) to c.taille-1 do
      write(c.contenu[i,j]:FORMAT);
    writeln();
  end {for};
end {ecrireCarre};
```

Somme d'une ligne

On va écrire une fonction `sommeLigne` qui calcule la somme des éléments de la ligne `i` du carré `C`.

Somme d'une ligne

On va écrire une fonction sommeLigne qui calcule la somme des éléments de la ligne i du carré C.

Listing

```
function sommeLigne(const C: CARRE ;
                    const i: INDICE): CARDINAL ;
var
   j: INDICE ;
   s: CARDINAL ;
begin
   s := 0 ;
   for j := low(INDICE) to C.taille-1 do
      s := s + C.contenu[i,j] ;
   sommeLigne := s
end {sommeLigne} ;
```

Somme d'une colonne

On va écrire une fonction `sommeColonne` qui calcule la somme des éléments de la colone `j` du carré `C`.

Somme d'une colonne

On va écrire une fonction sommeColonne qui calcule la somme des éléments de la colone j du carré C.

Listing

```
function sommeColonne(const C: CARRE ;
                      const j: INDICE): CARDINAL ;
var
   i: INDICE ;
   s: CARDINAL ;
begin
   s := 0 ;
   for i := low(INDICE) to C.taille-1 do
      s := s + C.contenu[i,j] ;
   sommeColonne := s
end {sommeColonne} ;
```

Somme de la diagonale

On va écrire une fonction `sommeDiagonale` qui calcule la somme des éléments de la diagonale principale du carré `C`.

Somme de la diagonale

On va écrire une fonction sommeDiagonale qui calcule la somme des éléments de la diagonale principale du carré C.

Listing

```
function sommeDiagonale(const C: CARRE): CARDINAL ;
var
    i: INDICE ;
    s: CARDINAL ;
begin
    s := 0 ;
    for i := low(INDICE) to C.taille-1 do
        s := s + C.contenu[i,i] ;
    sommeDiagonale := s
end {sommeDiagonale} ;
```

Somme de la deuxième diagonale

On va écrire une fonction `sommeDiagonale2` qui calcule la somme des éléments de la deuxième diagonale du carré `C`.

Somme de la deuxième diagonale

On va écrire une fonction sommeDiagonale2 qui calcule la somme des éléments de la deuxième diagonale du carré C.

Listing

```
function sommeDiagonale2(const C: CARRE): CARDINAL ;
var
   i: INDICE ;
   s: CARDINAL ;
begin
   s := 0 ;
   for i := low(INDICE) to C.taille-1 do
      s := s + C.contenu[i,C.taille-1-i] ;
   sommeDiagonale2 := s
end {sommeDiagonale2} ;
```

Tester toutes les lignes

Écrire une fonction `lignesCorrectes` qui vérifie que toutes les lignes ont la même valeur v

Tester toutes les lignes

Écrire une fonction `lignesCorrectes` qui vérifie que toutes les lignes ont la même valeur *v*

Listing

```
function lignesCorrectes(const C: CARRE,
                         const v: CARDINAL): BOOLEAN;
var
   i: INDICE ;
   b: BOOLEAN ;
begin
   b := true ;
   for i := low(INDICE) to C.taille-1 do
      b := b and (sommeLigne(C,i)=v) ;
   lignesCorrectes := b
end {lignesCorrectes} ;
```

Tester toutes les lignes

Écrire une fonction `lignesCorrectes` qui vérifie que toutes les lignes ont la même valeur *v*

Listing

```
function lignesCorrectes(const C: CARRE,
                         const v: CARDINAL): BOOLEAN;
var
   i: INDICE ;
   b: BOOLEAN ;
begin
   b := true ;
   for i := low(INDICE) to C.taille-1 do
      b := b and (sommeLigne(C,i)=v) ;
   lignesCorrectes := b
end {lignesCorrectes} ;
```

On peut faire de même pour les colonnes : `colonnesCorrectes`

La fonction estCarreMagique

Listing

```
// estCarreMagique(C)= vrai si C est un carre magique
//                   = faux sinon
function estCarreMagique(const C: CARRE): CARDINAL ;
var
   i: INDICE ;
   v: CARDINAL ;
begin
   v := sommeDiagonale(C) ;
   estCarreMagique :=      (v=sommeDiagonale2(C))
                       and lignesCorrectes(C,v)
                       and colonnesCorrectes(C,v)
end {estCarreMagique} ;
```