

Test de magie [tb04] - Exercice résolu

Karine Zampieri, Stéphane Rivière

Unisciel

sciel

algotrog

UNIVERSITÉ
HAUTE-ALSACE

Version 19 mai 2018

Table des matières

1 Énoncé	2
2 Algorithmique, Programmation	3
2.1 Stockage linéaire (tableau bidimensionnel)	3
2.2 Procédure afficherCarre (affichage d'un carré)	5
2.3 Fonction permutationTab (test de permutation)	5
2.4 Test de magie	7
2.5 Algorithme principal	9
3 Références générales	10

C++ - Test de magie (Solution)



Mots-Clés Tableau multidimensionnel, Carré magique ■

Requis Structures de base, Structures conditionnelles, Algorithmes paramétrés, Structures répétitives, Schéma itératif, Tableaux ■

Difficulté ●●○ (45 min) ■



Objectif

Cet exercice teste la « magie » d'un carré d'ordre n .

1 Énoncé

Définitions

Un **carré d'ordre** n est un tableau bidimensionnel $n \times n$ d'entiers.

Il est dit **semi-magique** si la somme de chaque ligne, la somme de chaque colonne et la somme de chaque diagonale principale (montante et descendante) est la même valeur.

Enfin, il est dit **magique**¹ s'il est semi-magique et s'il est une permutation des entiers $[1..n^2]$. Dans ce cas, la somme théorique vaut $n(n^2 + 1)/2$.

Exemple

Le carré $[4, 5, 6, 1, 2, 3, 7, 8, 9]$ d'ordre 3 est une 9-permutation mais n'est pas semi-magique.

$$\begin{array}{ccc|c} 4 & 5 & 6 & =15 \\ 1 & 2 & 3 & =6 \\ 7 & 8 & 9 & =24 \end{array}$$

Exemple

Le carré $[8, 1, 6, 3, 5, 7, 4, 9, 2]$ d'ordre 3 est magique (puisque 9-permutation et semi-magique).

$$\begin{array}{ccc|c} & & & =15(\text{diag}) \\ 8 & 1 & 6 & =15 \\ 3 & 5 & 7 & =15 \\ 4 & 9 & 2 & =15 \\ =15 & =15 & =15 & =15(\text{diag}) \end{array}$$

Exemple

Le carré suivant d'ordre 4 est un carré magique.

$$\begin{array}{cccc|c} & & & & =34(\text{diag}) \\ 16 & 3 & 2 & 13 & =34 \\ 5 & 10 & 11 & 8 & =34 \\ 9 & 6 & 7 & 12 & =34 \\ 4 & 15 & 14 & 1 & =34 \\ =34 & =34 & =34 & =34 & =34(\text{diag}) \end{array}$$

Objectif

Représenter un carré sous sa forme linéaire, saisir un carré et tester sa magie.

1. Les carrés magiques sont très anciens, puisqu'on trouve leur trace, il y a plus de 3000 ans, sur la carapace d'une tortue chinoise de Lo SHU. En Europe, le premier carré magique apparaît en 1514 sur une gravure du peintre allemand A. DÜRER. Si durant de nombreux siècles ces carrés étaient attachés à des superstitions divines, à partir du XVII^e siècle, ils ont fait l'objet de nombreuses études mathématiques.

2 Algorithmique, Programmation



Définissez la constante `TMAX=100` (nombre maximal de cases d'un tableau), le type `ITableau` comme étant un tableau de `TMAX` entiers, puis le typ `TCarre` comme étant un `ITableau`.



Validez vos définitions avec la solution.

Solution C++ @[pgtmagicite.cpp]

```
/**
 * Taille maximale du tableau
 */
const int TMAX = 100;

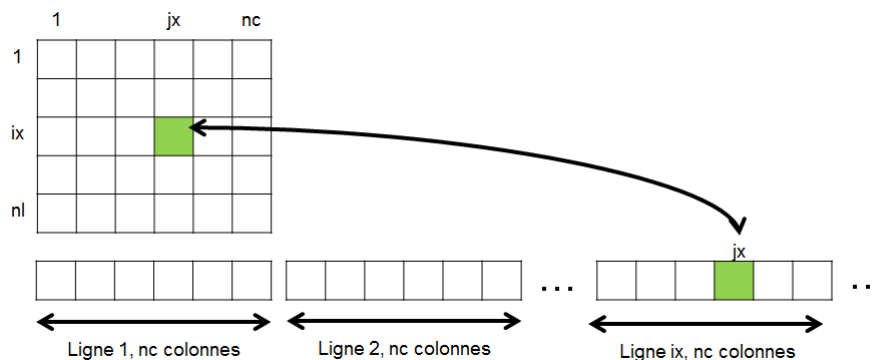
/**
 * Tableau d'entiers
 */
using ITableau = int[TMAX];
```

2.1 Stockage linéaire (tableau bidimensionnel)



Stockage linéaire d'un tableau bidimensionnel

Il permet d'optimiser son espace mémoire et correspond à effectuer la transformation suivante :



Écrivez une fonction `indexTab2d(j,k,ncols)` qui renvoie l'index linéaire de la case en (j,k) pour un `ITableau` à `ncols` colonnes. **Attention**, les indices sont numérotés à partir de 1.



Écrivez une fonction `evalCase(t,j,k,n)` qui renvoie la valeur de l'élément en (j,k) d'un `ITableau t` de `n` colonnes.





Écrivez une procédure `fixerCase(t,j,k,n,valeur)` qui fixe l'élément en (j,k) d'un `ITableau` `t` de `n` colonnes à la valeur `valeur`.



Validez vos fonctions et procédure avec la solution.

Solution C++ @[UtilsTM.cpp]

```

/**
 * Index linéaire d'un tableau bi-dimensionnel
 * @param[in] j - index-ligne à partir de 1
 * @param[in] k - index-colonne à partir de 1
 * @param[in] ncols - nombre de colonnes du tableau
 * @return l'index linéaire de (j,k)
 */
int indexTab2d(int j, int k, int ncols)
{
    return ((j - 1) * ncols + k - 1);
}

/**
 * Valeur de l'élément d'un TCarre
 * @param[in] t - un ITableau
 * @param[in] j - index-ligne à partir de 1
 * @param[in] k - index-colonne à partir de 1
 * @param[in] n - ordre de t
 * @return la valeur de l'élément en (j,k) de t
 */
int evalCase(const ITableau& t, int j, int k, int n)
{
    return t[indexTab2d(j,k,n)];
}

/**
 * Fixe l'élément d'un TCarre
 * @param[in,out] t - un ITableau
 * @param[in] j - index-ligne à partir de 1
 * @param[in] k - index-colonne à partir de 1
 * @param[in] n - ordre de t
 * @param[in] valeur - valeur à fixer
 */
void fixerCase(ITableau& t, int j, int k, int n, int valeur)
{
    t[indexTab2d(j,k,n)] = valeur;
}
  
```

2.2 Procédure afficherCarre (affichage d'un carré)



Écrivez une procédure `afficherCarre(t,n)` qui affiche un `ITableau t` d'ordre `n` sous sa forme bidimensionnelle. Exemple d'un carré d'ordre 4 :

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



Validez votre procédure avec la solution.

Solution C++ @[UtilsTM.cpp]

```
/**
 * Affiche un TCarre
 * @param[in] t - un ITableau
 * @param[in] n - ordre de t
 */
void afficherCarre(const ITableau& t, int n)
{
    for (int j = 1; j <= n; ++j)
    {
        for (int k = 1; k <= n; ++k)
        {
            cout<<setw(3)<<evalCase(t,j,k,n);
        }
        cout<<endl;
    }
}
```

2.3 Fonction permutationTab (test de permutation)



Définition

Une n -permutation est l'ensemble des entiers $\{1, 2, \dots, n\}$.



Écrivez le **profil** d'une fonction `permutationTab(t,n)` qui renvoie `Vrai` si les `n` premières valeurs d'un `ITableau t` est une n -permutation, `Faux` sinon.

Solution Paramètres

Entrants : Un `Tableau t` et un entier `n`

Résultat de la fonction : Un booléen

Solution Profil C++

```
bool permutationTab(const ITableau& t, int n)
```

**Analyse**

Pour tester **efficacement** que chaque entier de `t` est dans `[1..n]` et qu'il n'y est qu'une unique fois, on utilise un tableau de booléens `vu` (« entier déjà vu ») et un booléen `rs` qui est le résultat de la fonction.

On effectue alors les trois étapes :

1. Initialement, aucun entier a été vu (chaque case de `vu` vaut `Faux`) et `t` est une n -permutation (`rs` vaut `Vrai`).
2. On traverse `t` et on vérifie que `t[j]` est dans `[1..n]` et qu'il n'a pas été « entier déjà vu ». Dans le cas où cela n'est pas vérifié, `t` ne peut pas être une n -permutation : on fixe `rs` à `Faux`.
3. On renvoie la valeur de `rs`.

De cette analyse,



Écrivez le corps de la fonction.



Validez votre fonction avec la solution.

Solution C++

@[UtilsTBOpers.cpp]

```
/**
 * Prédicat de permutation d'un ITableau
 * @param[in] t - un ITableau
 * @param[in] n - ordre du tableau dans [0..TMAX[
 * @return Vrai si t est une permutation d'ordre n, Faux sinon
 */
bool permutationTab(const ITableau& t, int n)
{
    bool vu[TMAX] = {false};
    bool rs = true;
    int j = 0;
    while (rs && j < n)
    {
        int valeur = t[j] - 1;
        if (valeur < 0 || valeur >= n || vu[valeur])
        {
            rs = false;
        }
        else
        {
            vu[valeur] = true;
            ++j;
        }
    }
    return rs;
}
```

2.4 Test de magie



Définissez le type `TCarre` comme étant un `ITableau`.



Écrivez une fonction `sommePartielle(t, ideb, jdeb, iincr, jincr, n)` qui renvoie la somme de `n` cases d'un `TCarre t` en partant de celle en `(ideb, jdeb)` et d'incrémentations de ligne `iincr` et de colonne `jincr`.

```
.      jdeb
ideb  .  .  .  .
      .  .  .  .
      .  .  .  .
      .  .  .  .
```

Solution simple

On calcule une somme cumulée de `n` cases. Il faut donc déclarer une variable et l'initialiser à zéro puis parcourir `n` cases (boucle `Pour`) en partant de celle en `(ideb, jdeb)`. A chaque tour de boucle, `ideb` est incrémenté avec `iincr` et `jdeb` avec `jincr`.



Validez votre fonction avec la solution.

Solution C++ @[pgtmagicite.cpp]

```
/**
 * Représente un Carré
 */

using TCarré = ITableau;

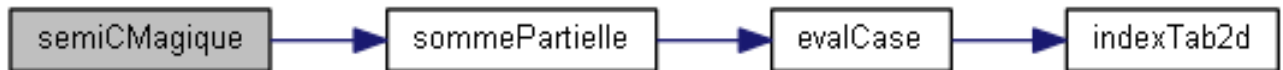
/**
 * Somme partielle depuis une case
 * @param[in] c -
 * @param[in] ideb - indice début ligne
 * @param[in] jdeb - indice début colonne
 * @param[in] iincr - incrément selon les lignes
 * @param[in] jincr - incrément selon les colonnes
 * @param[in] n - ordre de c
 * @return La somme de n éléments de c en partant de (ideb, jdeb)
 */

int sommePartielle(const TCarré& c, int ideb, int jdeb, int iincr, int jincr, int n)
{
    int rs = 0;
    int ix = ideb;
    int jx = jdeb;
    for (int k = 1; k <= n; ++k)
    {
        rs += UtilsTM::evalCase(c, ix, jx, n);
        ix += iincr;
        jx += jincr;
    }
}
```

```
return rs;
}
```



Écrivez ensuite une fonction `semiMagique(t,n)` qui teste et renvoie `Vrai` si un `TCarre t` d'ordre `n` est semi-magique, `Faux` sinon.



Solution simple

Il faut déterminer la somme théorique (sigma d'une ligne par exemple) et vérifier que chacune des lignes, colonnes et diagonales principales ont cette même valeur.



Validez votre fonction avec la solution.

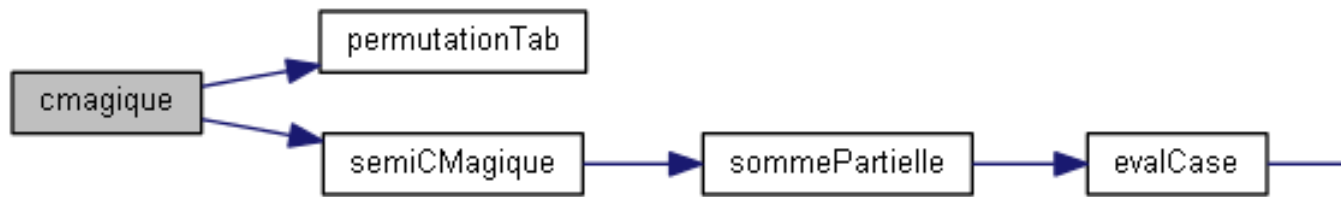
Solution C++ @[pgtmagicite.cpp]

```
/**
 * Prédicat de carre semi-magique
 * @param[in] c - un TCarre
 * @param[in] n - ordre de c
 * @return Vrai si c est un carré semi-magique d'ordre n
 */

bool semiMagique(const TCarre& c, int n)
{
    bool rs = true;
    int sth = n * (n * n + 1) / 2;
    int k;
    k = 1;
    while (k <= n && rs)
    {
        rs = rs && (sommePartielle(c,k,1,0,1,n) == sth);
        ++k;
    }
    k = 1;
    while (k <= n && rs)
    {
        rs = rs && (sommePartielle(c,1,k,1,0,n) == sth);
        ++k;
    }
    rs = rs && (sommePartielle(c,1,1,1,1,n) == sth);
    rs = rs && (sommePartielle(c,1,n,1,-1,n) == sth);
    return rs;
}
```



Déduisez une fonction `cmagique(t,n)` qui teste et renvoie `Vrai` si un `TCarre t` d'ordre `n` est magique, `Faux` sinon.



Validez votre fonction avec la solution.

Solution C++ @[pgtmagicite.cpp]

```

/**
 * Prédicat de carré magique
 * @param[in] c - un TCarre
 * @param[in] n - ordre de c
 * @return Vrai si c est un carré magique d'ordre n
 */
bool cmagique(const TCarre& c, int n)
{
    bool b1 = UtilsTB0pers::permutationTab(c, n * n);
    bool b2 = semiCMagique(c, n);
    return (b1 && b2);
}
  
```

2.5 Algorithme principal



Téléchargez le fichier suivant et mettez-le dans votre dossier.

C++ @[UtilsTB.cpp]



Copiez/collez ensuite les lignes suivantes :

C++ Au début de votre programme :

```

#include "UtilsTB.cpp"
using namespace UtilsTB;
  
```



Soit la fonction `saisirTab(t)` qui effectue la saisie contrôlée du nombre de valeurs (entier compris entre 1 et `TMAX`), saisit les valeurs entières dans un `ITableau t` puis renvoie l'entier du nombre de valeurs saisies.

C++ @[saisirTab] (dans UtilsTB)



Écrivez un programme qui :

- Déclare un `TCarre` et saisit les entiers dans le `TCarre` (fonction `saisirTab`).
- Demande et saisit l'ordre du `TCarre`.
- Enfin affiche le `TCarre` ainsi que ses caractéristiques (permutation, semi-magique, magique).



Testez avec les exemples ci-dessous :

- Le carré [4,5,6,1,2,3,7,8,9] d'ordre 3.
- Le carré [8,1,6,3,5,7,4,9,2] d'ordre 3.
- Le carré [16,3,2,13,5,10,11,8,9,6,7,12,4,15,14,1] d'ordre 4.



Validez votre procédure avec la solution.

Solution C++

@[pgtmagicite.cpp]

```
int main()
{
    TCarre c;
    int nelems = UtilsTB::saisirTab(c);
    int n;
    cout<<"Ordre n? ";
    cin>>n;
    UtilsTM::afficherCarre(c,n);

    bool b1 = UtilsTB0pers::permutationTab(c,n * n);
    bool b2 = semiCMagique(c,n);
    bool b3 = cmagique(c,n);
    cout<<"==> permutation: "<<b1<<endl;
    cout<<"==> semi-magique: "<<b2<<endl;
    cout<<"==> cmagique: "<<b3<<endl;
}
```

3 Références générales

Comprend [Maunoury-AL1 :c8 :ex12] ■