Test de magicité [tb04] - Exercice résolu

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Version 19 mai 2018

Table des matières

alg - Test de magicité (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 « magicité » 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.

4	5	6	=15
1	2	3	=6
7	8	9	=24

Exemple

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

			=15(diag)
8	1	6	=15
3	5	7	=15
4	9	2	=15
=15	=15	=15	=15(diag)

Exemple

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

				=34(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(diag)

Objectif

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

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

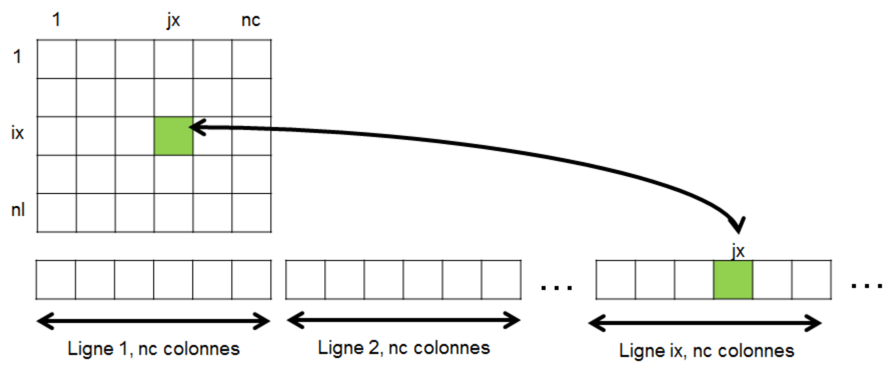
Définissez la constante `TMAX=100` (nombre maximal de cases d'un tableau), le type `ITableau` comme étant un tableau de `TMAX` entiers, puis le typ `TCarre` comme étant un `ITableau`.

Validez vos définitions avec la solution.

Solution alg @[pgtmagicite.alg]

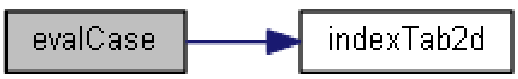
2.1 Stockage linéaire (tableau bidimensionnel)

Stockage linéaire d'un tableau bidimensionnel
Il permet d'optimiser son espace mémoire et correspond à effectuer la transformation suivante :

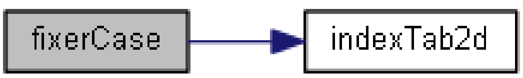


Écrivez une fonction `indexTab2d(j,k,ncols)` qui renvoie l'index linéaire de la case en `(j,k)` pour un `ITableau` à `ncols` colonnes. **Attention,** les indices sont numérotés à partir de 1.

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 alg @[UtilsTM.alg]

```
Fonction indexTab2d ( ix : Entier ; jx : Entier ; ncols : Entier ) : Entier
Début
  |  Retourner ( ( ix - 1 ) * ncols + jx )
Fin
```

```
Fonction evalCase ( DR t : Entier [ NMAX ] ; ix : Entier ; jx : Entier ; n : Entier ) :
    Entier
Début
  |  Retourner ( t [ indexTab2d ( ix , jx , n ) ] )
Fin
```

```
Action fixerCase ( DR t : Entier [ NMAX ] ; ix : Entier ; jx : Entier ; n : Entier ; val
    : Entier )
Début
  |  t [ indexTab2d ( ix , jx , n ) ] <- val
Fin
```

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 alg @[UtilsTM.alg]

```
Action afficherCarre ( DR t : Entier [ NMAX ] ; n : Entier )
Variable ix , jx : Entier
Début
  |  Pour ix <- 1 à n Faire
  |    |  Pour jx <- 1 à n Faire
  |    |    |  Afficher ( evalCase ( t , ix , jx , n ) : 3 )
  |    |  FinPour
  |    |  Afficher ( "" )
  |  FinPour
Fin
```

2.3 Fonction permutationTab (test de permutation)

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

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

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 alg @[UtilsTBOpers.alg]

```
Fonction permutationTab ( DR t : Entier [ TMAX ] ; n : Entier ) : Booléen
Variable ix : Entier
Variable vu : Booléen [ TMAX ]
Variable rs : Booléen
Variable valeur : Entier
Début
  |  Pour ix <- 1 à n Faire
  |    |  vu [ ix ] <- Faux
  |  FinPour
  |  rs <- Vrai
  |  ix <- 1
  |  TantQue ( rs Et ix <= n ) Faire
  |    |  valeur <- t [ ix ]
  |    |  Si ( valeur < 1 Ou n < valeur Ou vu [ valeur ] ) Alors
  |    |    |  rs <- Faux
  |    |  Sinon
  |    |    |  vu [ valeur ] <- Vrai
  |    |  FinSi
  |    |  ix <- ix + 1
  |  FinTantQue
```

```
  |  Retourner ( rs )
Fin
```

2.4 Test de magicité

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

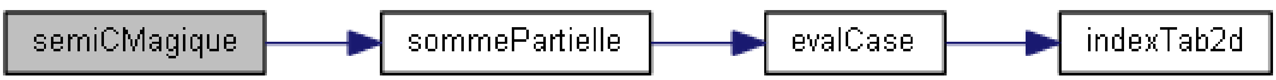
On calcule une somme cumulée de `n` cases. Il faut donc déclarer une variable et l'initialiser à zéro puis parcourir `n` cases (boucle `Pour`) en partant de celle en `(ideb,jdeb)`. A chaque tour de boucle, `ideb` est incrémenté avec `iincr` et `jdeb` avec `jincr`.

Validez votre fonction avec la solution.

Solution alg @[UtilsCMagique.alg]

```
Fonction sommePartielle ( DR t : Entier [ NMAX ] ; ideb : Entier ; jdeb : Entier ; iincr
    : Entier ; jincr : Entier ; n : Entier ) : Entier
Variable rs : Entier
Variable ix , jx , k : Entier
Début
  |  rs <- 0
  |  ix <- ideb
  |  jx <- jdeb
  |  Pour k <- 1 à n Faire
  |    |  rs <- rs + evalCase ( t , ix , jx , n )
  |    |  ix <- ix + iincr
  |    |  jx <- jx + jincr
  |  FinPour
  |  Retourner ( rs )
Fin
```

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



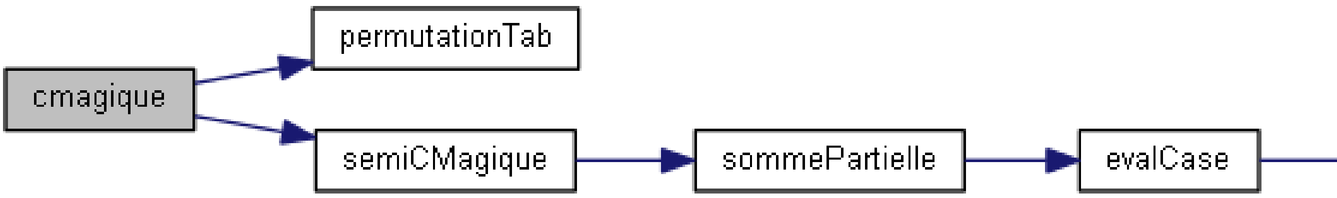
Solution simple
Il faut déterminer la somme théorique (sigma d'une ligne par exemple) et vérifier que chacune des lignes, colonnes et diagonales principales ont cette même valeur.

Validez votre fonction avec la solution.

Solution alg @[UtilsCMagique.alg]

```
Fonction semiCMagique ( DR t : Entier [ NMAX ] ; n : Entier ) : Booléen
Variable rs : Booléen
Variable somme : Entier
Variable ix , jx : Entier
Début
  |  rs <- Vrai
  |  somme <- DivEnt ( n * ( n * n + 1 ) , 2 )
  |  ix <- 1
  |  TantQue ( rs Et ix <= n ) Faire
  |    |  rs <- rs Et ( sommePartielle ( t , ix , 1 , 0 , 1 , n ) = somme )
  |    |  ix <- ix + 1
  |  FinTantQue
  |  jx <- 1
  |  TantQue ( rs Et jx <= n ) Faire
  |    |  rs <- rs Et ( sommePartielle ( t , 1 , jx , 1 , 0 , n ) = somme )
  |    |  jx <- jx + 1
  |  FinTantQue
  |  rs <- rs Et ( sommePartielle ( t , 1 , 1 , 1 , 1 , n ) = somme )
  |  rs <- rs Et ( sommePartielle ( t , 1 , n , 1 , - 1 , n ) = somme )
  |  Retourner ( rs )
Fin
```

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 alg @[UtilsCMagique.alg]

```
Fonction cmagique ( DR t : Entier [ NMAX ] ; n : Entier ) : Booléen
Début
  |  Retourner ( permutationTab ( t , n * n ) Et semiCMagique ( t , n ) )
Fin
```

2.5 Algorithme principal

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.

Écrivez un algorithme 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 alg @[pgtmagicite.alg]

```
Algorithme pgtmagicite
Variable tab : Entier [ NMAX ]
Variable nelems : Entier
Variable n : Entier
Début
  | n <- saisirTab ( tab )
  | Afficher ( "Ordre n? " )
  | Saisir ( n )
  | afficherCarre ( tab , n )
  | Afficher ( "Est permutation: " , permutationTab ( tab , n * n ) )
  | Afficher ( "Est semi-magique: " , semiCMagique ( tab , n ) )
  | Afficher ( "Est cmagique: " , cmagique ( tab , n ) )
Fin
```

3 Références générales

Comprend [Maunoury-AL1 :c8 :ex12] ■