Chapitre 5

Tableaux

5.1 Tableaux à une dimension

Il est parfois nécessaire de conserver en mémoire plusieurs données de même type, par exemple une valeur calculée à chaque itération d'une boucle. Cela nécessite de pouvoir écrire un programme dans lequel le nombre de variables disponibles est lui-même un paramètre – une suite finie (mais de longueur variable) de valeurs du même type. Une telle construction s'appelle un *tableau*.
En `Java`, le type "tableau d'éléments de type `T`" se note `T[]`. Les éléments d'un tableau `t` de taille `n` sont numérotés de `0` à `n-1`, et sont désignés par `t[0]`, `t[1]`, ..., `t[n-1]`.

Remarque. On a déjà rencontré un exemple de tableau dans l'en-tête de la fonction `main` d'un programme dont le paramètre est un tableau sur le type `String` (il contient la liste des paramètres donnés au lancement du programme).

```
public static void main(String[] args) { ... }
```

Le fragment de programme suivant déclare une variable `tab` de type "tableau d'entiers", puis il alloue un tableau de 5 entiers (via l'opérateur `new`) et l'affecte à la variable `tab`. Les cinq éléments du tableau `tab` se comporteront ensuite comme cinq variables de type `int`, désignées par `tab[0]`, `tab[1]`, ..., `tab[4]`.

```
int[] tab;
tab = new int[5];
```

On peut obtenir la *longueur* du tableau correspondant à une variable `t` (c'est-à-dire son nombre de cases) à l'aide de l'expression `t.length`. Les éléments d'un tableau `t` sont ainsi désignés par `t[0]`, `t[1]`, ..., `t[t.length - 1]`.

Remarque. Contrairement à une chaîne de caractères `w` de type `String` (autre type référence), pour laquelle on récupère la longueur via l'expression `w.length()` (méthode sans paramètre), un tableau `t` voit sa longueur fixée au moment de l'allocation, longueur qui devient alors un *attribut* du tableau, auquel on accède via l'expression `t.length` (pas de parenthèses dans ce cas).

Exemple 26. Supposons que l'on veuille lire un entier `n`, puis lire `n` entiers et les afficher dans l'ordre inverse de celui dans lequel ils ont été lus. La séquence de code `Java` correspondante est :

```
        int n;
2       Scanner sc = new Scanner(System.in);
        System.out.print("Taille␣du␣tableau␣a␣renverser␣:␣");
4       n = sc.nextInt();
        int[] t = new int[n];
```

```
        for (int i = 0; i < t.length; i++) {
            System.out.print("Donnez-moi␣l'entier␣" + i + "␣:␣");
            t[i] = sc.nextInt();
        }
        System.out.println("Voici␣la␣liste␣renversee:␣");
        for (int i = t.length-1; i >= 0; i--) {
            System.out.print(t[i]);
            System.out.print("␣");
        }
```

Renversement.java

► **Exercice 47** :

Écrire un programme qui affiche les paramètres avec lesquels il a été lancé à raison d'un paramètre par ligne.

Dans l'instruction `tab = new int[5];` l'opérateur `new` renvoie l'adresse mémoire de l'emplacement des éléments du tableau nouvellement créé ; cette adresse est alors affectée à la variable `tab`, ce qui permet ensuite d'accéder à chacun des éléments du tableau.
Ainsi, la *valeur* d'une variable de type tableau est une indication de l'emplacement mémoire de ce tableau et est sans rapport avec le *contenu* du tableau (c'est-à-dire les valeurs de ses cases).

Exemple 27. Après la séquence de code suivante :

```
int[] tab = new int[5];
tab[0] = 4; tab[1] = 12; tab[2] = -3; tab[3] = 0; tab[4] = 5;
System.out.println(tab);

int[] tac = tab;
tac[0] = 3;
System.out.println(tab[0]);

int[] tad = {3,12,-3,0,5};
System.out.println(tab==tac);
System.out.println(tab==tad);
```

Tout d'abord, l'instruction 3 provoque l'affichage de la valeur de `tab` (quelque chose comme `[I@735cda3f`) et en aucun cas l'affichage de son contenu.
Ensuite, l'instruction 5 fait que la variable `tac` désigne le *même* tableau que `tab`. En particulier, l'instruction 7 produit l'affichage 3. L'affectation `tac = tab` ne recopie donc pas le contenu du tableau. Pour recopier le contenu d'un tableau, il faut recopier les valeurs des cases une à une (en utilisant une boucle) — voir l'exemple 28.
Enfin, après l'instruction 9, la variable `tad` désigne un nouveau tableau ayant le même contenu que celui désigné par `tab` : l'évaluation du test `tab==tad` cependant donne `false`.

► **Exercice 48** :

Écrire une fonction `sontEgaux` qui teste si deux tableaux d'entiers sont égaux.

► **Exercice 49** (Examen 2003/2004, exercice 2) :

On s'intéresse ici à des tableaux de nombres entiers à une seule dimension, possédant un nombre impair d'éléments et dont tous les éléments sont différents. On appelle médiane d'un tel tableau `t` l'élément `m` de `t` tel que `t` contienne autant d'éléments strictement inférieurs à `m` que d'éléments strictement supérieurs à `m`.

1. Écrire une fonction `nbInf` qui, étant donné un tableau d'entiers `t` et un entier `v`, renvoie le nombre d'éléments du tableau `t` strictement inférieurs à `v`.
2. Écrire une fonction `mediane` qui, étant donné un tableau `t` satisfaisant les conditions énoncées, renvoie la position de la médiane dans le tableau `t`.
3. Écrire une fonction `verifTableau` qui, étant donné un tableau `t` de nombres entiers, renvoie la valeur booléenne `true` si le tableau `t` satisfait effectivement les conditions énoncées et la valeur `false` si ce n'est pas le cas.
4. Écrire une fonction `tabInf` qui, étant donné un tableau `t`, renvoie `null` si `t` ne satisfait pas les conditions énoncées et un tableau contenant tous les éléments de `t` inférieurs à sa médiane sinon.

► **Exercice 50** (Partiel 2003/2004, exercice 5) :

On considère n points $A_0, A_1, \ldots, A_{n-1}$ de l'espace dont les coordonnées (abscisse x_i, ordonnée y_i et cote z_i) sont regroupées dans trois tableaux

```
double[] abscisses, ordonnees, cotes;
```

Ces points sont par ailleurs pondérés, c'est-à-dire que chacun possède une masse m_i. Les masses des différents points sont regroupées dans un tableau

```
double[] masses;
```

On appelle barycentre de cet ensemble de n points, le point G défini uniquement si la somme des n masses est non nulle (c'est-à-dire si $m = \sum_{i=0}^{n-1} m_i \neq 0$ et dont les coordonnées sont alors

$$x_G = \frac{1}{m}\sum_{i=0}^{n-1} m_i x_i \qquad y_G = \frac{1}{m}\sum_{i=0}^{n-1} m_i y_i \qquad z_G = \frac{1}{m}\sum_{i=0}^{n-1} m_i z_i.$$

Écrire une fonction `barycentre` qui recevant en paramètres quatre tableaux `abscisses`, `ordonnees`, `cotes` et `masses`
– affiche les coordonnées du barycentre de l'ensemble donné de points pondérés, s'il existe,
– affiche un message d'erreur sinon.

5.2 Création d'un tableau à l'intérieur d'une fonction

On peut créer un tableau dans une fonction. Cela peut être utile pour :
– effectuer des calculs intermédiaires,
– créer un nouveau tableau qui sera renvoyé par la fonction.

5.2.1 Calculs intermédiaires

On considère une suite d'entiers définie par récurrence par

$$x_{n+k+1} = a_k x_{n+k} + a_{k-1} x_{n+(k-1)} + \cdots + a_1 x_{n+1} + a_0 x_n,$$

où k, les $(a_i)_{0\leq i\leq k}$ et les $(x_i)_{0\leq i\leq k}$ sont fixés.
La fonction suivante renvoie la valeur de x_n sans perdre les données initiales :

```
class Suite {
2    /**
      * calcul du n-eme terme d'une suite recurrente lineaire
```

```
     * @param a les coefficients de la recurrence
     * @param x les premiers termes
     * @param n le rang a calculer
     * @return x_n
     */
    static int suite(int[] a, int[] x, int n){
        int[] tmp = new int[x.length];
        int elementSuivant = 0;

        if (n<0) System.exit(0);

        if (n<x.length) return x[n];

        for(int i=0; i<x.length; i++)
            tmp[i] = x[i];
        for(int i=x.length; i<=n; i++){
            elementSuivant = 0; // calul de x_i
            for(int j=0; j<x.length; j++)
                elementSuivant += a[j]*tmp[j];
            for(int j=0; j<x.length-1; j++) // mise a jour
                tmp[j] = tmp[j+1];          // de tmp
            tmp[x.length-1] = elementSuivant;
        }
        return elementSuivant;
    }

    public static void main(String[] args){
        int[] fibonacci = {1, 1};
        int[] valeursInitiales = {0, 1};
        int n = 10;
        System.out.println("Fibonacci au rang " + n + " = " +
                           suite(fibonacci, valeursInitiales, n));
    }
}
```

Suite.java

5.2.2 Valeur renvoyée

On peut aussi se servir d'un tableau comme valeur de retour d'une fonction.

Exemple 28. Revenons sur une question levée dans l'exemple 27. La fonction suivante prend en paramètre un tableau d'entiers et renvoie une copie de ce tableau : un autre tableau d'entiers de même longueur et de même contenu que ce paramètre :

```
    static int[] copieTableau(int[] t){
        int[] s = new int[t.length];
        for(int i=0; i<t.length; i++)
            s[i] = t[i];
        return s;
    }
```

CopieTableau.java

► **Exercice 51** (Partiel 2004/2005, exercice 6) :

Écrire une fonction `begaiement` qui prend en paramètre un tableau de caractères et qui renvoie un nouveau tableau de caractères déduit du premier en dédoublant chacun de ses caractères (on n'utilisera pas les chaînes de caractères). Par exemple :

t	a	b	l	e	a	u	→	t	t	a	a	b	b	l	l	e	e	a	a	u	u

► **Exercice 52** (Partiel 2003/2004, exercice 4) :

On suppose qu'un tableau `tab` de `n` entiers contient les `n` premiers nombres entiers (c'est-à-dire `0`, `1`, ..., `n-1`) : chaque entier apparaît exactement une fois dans le tableau, mais les nombres n'apparaissent pas forcément dans l'ordre.
Écrire un fragment de code construisant à partir de `tab`, un tableau `position` de même longueur que `tab` et initialisant son contenu comme suit :

`position[i]` est égal à la position de l'entier `i` dans `tab`.

Par exemple, si `tab` est | 2 | 4 | 1 | 0 | 5 | 3 |, le tableau `position` sera | 3 | 2 | 0 | 5 | 1 | 4 | après l'exécution du code.

► **Exercice 53** (Partiel 2009/2010, exercice 6, questions 1 et 2) :

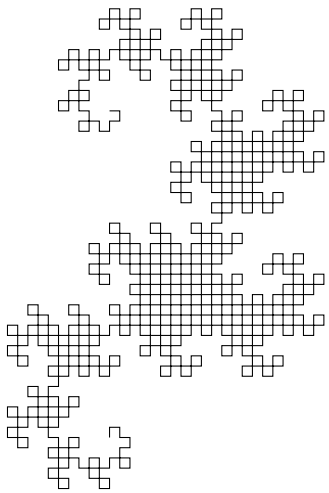
À partir d'un tableau d'entiers et d'un entier $p > 0$, on construit le tableau de longueur `p` obtenu en répétant *cyclique*ment les éléments du tableau initial soit vers la *droite* soit vers la *gauche*.

1	2	3	2	—cycliqueDroite avec p=6→	1	2	3	2	1	2
1	2	3	2	—cycliqueGauche avec p=9→	2	1	2	3	2	1	2	3	2
1	2	3	2	—cycliqueDroite avec p=2→	1	2

1. Écrire la fonction `cycliqueDroite`.
2. Écrire la fonction `cycliqueGauche`.

► **Exercice 54** (Examen 2011/2012, exercice 5) :

La *courbe du dragon* est une courbe récursive dont le nom provient de sa ressemblance avec une créature mythique. Elle peut être construite en représentant un virage à gauche par `true` et un virage à droite par `false`. Chaque courbe d'ordre donné peut ainsi être codée sous forme d'un tableau de booléens. La courbe du premier ordre est codée par | false |. Pour la courbe d'ordre `n`>`1`, on concatène le codage de la courbe `c` d'ordre `n-1`, celui de la courbe d'ordre `1` et celui de cette même courbe `c` mais parcourue en sens inverse (en particulier, les virages gauche et droite y sont échangés). Par exemple, la courbe du second ordre est codée par | false | false | true |, et celle du troisième ordre par | false | false | true | false | false | true | true |.



1. Donner le tableau de booléens codant la courbe du quatrième ordre. Dessiner celle-ci.
2. Écrire une fonction `suivant` qui prend une courbe en paramètre et renvoie la courbe d'ordre immédiatement supérieur.
3. Écrire une fonction récursive `dragon` qui renvoie la courbe de l'ordre en paramètre.
4. Le dessin ci-contre est réalisé via une instruction de la forme `(0,0)-(1,0)-(1,-1)-(0,-1)-(0,-2)-` ··· (les deux premières coordonnées sont fixées). Écrire une fonction `afficherInstructionDragon` qui affiche l'instruction associée à la courbe dont l'ordre est donné en paramètre.

5.3 Modification d'un tableau par une fonction

Une particularité intéressante est le fait qu'une fonction peut modifier l'objet référencé par un paramètre dès que celui-ci est d'un type référence, en particulier d'un type tableau. La fonction modifie alors le *contenu* du tableau, c'est-à-dire les valeurs de ses cases, mais laisse bien entendu inchangée la *valeur* du tableau, c'est-à-dire son emplacement mémoire.

Exemple 29. Revenons sur l'exemple 26, nous pouvons imaginer une fonction `tableauRenverse` qui prend un tableau en paramètre et renvoie un nouveau tableau de même contenu mais dans un ordre renversé.

```
    static String[] tableauRenverse(String[] t){
2       String[] r=new String[t.length];
        for (int i=0; i<t.length; i++)
4           r[i]=t[t.length-i-1]; //copie de l'element symetrique
        return r;
6   }
```

RenversementTableau.java

On peut envisager le renversement d'un tableau autrement en construisant une fonction `renverserTableau` qui prend un tableau en paramètre et renverse l'ordre de ses éléments (sans rien renvoyer). On dit que le renversement se fait *sur place*.

```
    static void renverserTableau(String[] t){
2       String tamp;
        for (int i=0; i<t.length/2; i++){ //demi parcours
4           tamp=t[i];              //mise en tampon
            t[i]=t[t.length-i-1]; //copie de l'element symetrique
6           t[t.length-i-1]=tamp; //copie de l'element mis en tampon
        }
8   }
```

RenversementTableau.java

Il est important de noter que pour la fonction `renverserTableau` :
- le type de retour est `void` (donc pas de commande `return`)
- le contenu du tableau en paramètre est modifié (pas son adresse)
- aucun nouveau tableau n'est créé (donc pas de commande `new`)

► **Exercice 55** (Partiel 2007/2008, exercice 5) :

1. Écrire une fonction `decaler1` qui reçoit en paramètre un tableau `t` d'entiers et décale les éléments d'une position vers la gauche de façon circulaire (le premier élément prend la place du dernier).
   Si le tableau est initialement t = | 5 | 2 | 3 | 0 | 8 | 7 | 3 | 1 |,
   la fonction le transforme en | 2 | 3 | 0 | 8 | 7 | 3 | 1 | 5 |.
   Cette fonction doit modifier le tableau qu'elle reçoit en paramètre, et ne renvoie rien.
2. Écrire une fonction `decaler` qui recoit en paramètre un tableau `t` d'entiers et un entier `k` et décale les éléments de `k` positions vers la gauche de façon circulaire.

► **Exercice 56** (Partiel 2009/2010, exercice 4) :

On se propose de construire une méthode de tri, appelé *tri par sélection*.

1. Écrire une fonction `maximum` qui prend en paramètres deux entiers `a` et `b` de type `int` et qui renvoie le maximum des deux.

2. Écrire une fonction `maximum` qui prend en paramètres un tableau `t` d'entiers de type `int` et deux indices `i` et `j` et qui, en supposant `i`$\leq$`j` valides et `t` non vide, renvoie la plus grande valeur contenue dans `t` à un indice compris entre `i` et `j`.
3. Écrire une fonction `indice` qui prend en paramètres un tableau `t` d'entiers de type `int` et une valeur `c` de type `int` et qui renvoie le plus petit indice d'une occurrence de `c` dans `t` si une telle occurrence existe et renvoie `-1` sinon.
4. Écrire une fonction `echanger` qui prend en paramètres un tableau `t` d'entiers de type `int` et deux indices `i` et `j` et qui échange dans `t` les éléments d'indice `i` et `j`.
5. Des trois fonctions précédentes, déduire une fonction `trier` qui prend en paramètre un tableau `t` d'entiers de type `int` et le trie : sélection de l'élément maximum et déplacement en dernière position, sélection de l'élément maximum sur la partie du tableau restant à trier et déplacement en avant dernière position, etc.

► **Exercice 57** (Examen 2009/2010, exercice 5) :

Le numéro d'immatriculation d'un véhicule est constitué de 7 caractères (il peut être représenté par un tableau de 7 éléments de type `char`) organisés en un premier bloc (deux lettres), un deuxième bloc (trois chiffres) et un troisième bloc (deux lettres). Le deuxième bloc évolue le plus vite, le premier bloc le moins vite :

	de	`AA-001-AA`	à	`AA-999-AA`	(deuxième bloc) ;
puis	de	`AA-001-AB`	à	`AA-999-AZ`	(troisième bloc, lettre de droite) ;
puis	de	`AA-001-BA`	à	`AA-999-ZZ`	(troisième bloc, lettre de gauche) ;
puis	de	`AB-001-AA`	à	`AZ-999-ZZ`	(premier bloc, lettre de droite) ;
puis	de	`BA-001-AA`	à	`ZZ-999-ZZ`	(premier bloc, lettre de gauche).

Pour le deuxième bloc, les dix chiffres sont autorisés. Pour les premier et troisième blocs, les lettres `I`, `O`, `U` sont interdites (pour éviter qu'elles ne soient confondues avec les chiffres `1`, `0` et la lettre `V`).
Les cinq méthodes ont pour argument un tableau de 7 caractères représentant un numéro d'immatriculation.

1. Écrire une méthode `toString` qui renvoie la chaîne de caractères représentant le numéro d'immatriculation associé avec des `"-"` entre les blocs.
2. Écrire une méthode `incrementerBloc2` qui incrémente le deuxième bloc.
3. Écrire une méthode `incrementerBloc3` qui incrémente le troisième bloc.
4. Écrire une méthode `incrementerBloc1` qui incrémente le premier bloc en évitant la série `WW` (réservée à l'immatriculation temporaire des véhicules neufs).
5. En déduire une méthode `incrementer` qui l'incrémente.

5.4 Tableaux à plusieurs dimensions

Les tableaux peuvent avoir plusieurs dimensions. On parle alors de tableaux multidimensionnels (ne pas confondre nombre de dimensions et taille du tableau). Un tableau multidimensionnel est un tableau de tableaux (de tableaux, de tableaux, ...).

5.4.1 Syntaxe

Chaque paire de crochets `[ ]` représente une dimension et ceci aussi bien pour la déclaration, la création que l'accès aux éléments.

```
// table de multiplication 0x0 ... 9x9
int[][] tableMult = new int[10][10];    // elements initialises a 0
for (int i=0; i<tableMult.length; i++)
    for (int j=0; j<tableMult[i].length; j++)
        tableMult[i][j] = i*j;
```

L'attribut `length` peut être consulté pour chacune des dimensions.

```
char[][] motsCroises = new char[5][10];
int nbLignes   = motsCroises.length;    // nbLignes=5
int nbColonnes = motsCroises[0].length; // nbColonnes=10
```

Les valeurs littérales sont constituées par l'emboîtement d'accolades contenant les éléments de chacune des dimensions, reflétant la structure de tableau de tableaux. Cette facilité n'est possible qu'au moment de l'initialisation.

```
// tableau de trois elements
// dont chacun represente un tableau de cinq elements
int[][] tableMult = {{0, 0, 0, 0, 0},
                     {0, 1, 2, 3, 4},
                     {0, 2, 4, 6, 8}};
```

► **Exercice 58** :

Écrire une fonction `toString` qui prend en argument un tableau à deux dimensions de nombres entiers et renvoie une chaîne de caractères le représentant.

► **Exercice 59** (Examen 2004/2005, exercice 3) :

Écrire une fonction qui, étant donnés un tableau `t` bidimensionnel de nombres entiers et un nombre entier `n`, renvoie :

- la valeur `null` si le tableau donné contient (au moins) un nombre `x` tel que $x < 0$ ou $x > n$,
- un tableau `tt` de $n+1$ entiers tel que `tt[i]` soit égal au nombre d'éléments de `t` égaux à `i` si le tableau ne contient que des nombres compris, au sens large, entre 0 et `n`.

► **Exercice 60** :

Étant donnée une matrice `a` (tableau bidimensionnel) avec `n` lignes et `m` colonnes, un couple d'indices `(i,j)` représente un *min-max* de cette matrice si la valeur `a[i][j]` est un minimum de la ligne `i` et un maximum de la colonne `j`, c'est-à-dire

$$\texttt{a[i][j]} = \min\{\texttt{a[i][0]},\ldots,\texttt{a[i][m}-1\texttt{]}\} \quad \text{et} \quad \texttt{a[i][j]} = \max\{\texttt{a[0][j]},\ldots,\texttt{a[n}-1\texttt{][j]}\}.$$

Écrire une fonction `afficherMinMax` qui prend un tableau bidimensionnel d'entiers en argument et affiche l'ensemble de tels couples `(i,j)`. On pourra opérer de la façon suivante :

1. pour chaque ligne `i`, trouver les minima de la ligne `i` et en mémoriser les numéros de colonne
2. pour chacun de ces rangs `j`, déterminer si `a[i][j]` est un maximum pour sa colonne.

5.4.2 Allocation partielle

Lors de la création d'un tableau multidimensionnel, il n'est pas obligatoire de définir la taille de toutes les dimensions (cependant, la première dimension doit obligatoirement être définie). Par contre, si l'on définit les tailles, il faut le faire dans l'ordre, c'est-à-dire de gauche à droite ou de l'extérieur vers l'intérieur.

```
byte[][][] c1 = new byte[10][][];   /* ok, declaration
                                       de la premiere dimension */
byte[][][] c2 = new byte[10][20][]; /* ok, declaration
                                       des deux premieres dimensions */
byte[][][] c3 = new byte[][20][];   /* erreur a la compilation */
```

Les tableaux multidimensionnels étant des tableaux de tableaux, ils ne sont pas obligatoirement rectangulaires (ou cubiques etc).

```
// tableau de quatre elements
// dont chacun represente un tableau de taille variable
// (tableau triangulaire)
int[][] tableMult = {{0},
                     {0, 1},
                     {0, 2, 4},
                     {0, 3, 6, 9}};
```

Les éléments tableaux des tableaux multidimensionnels peuvent être définis dynamiquement.

```
// tableau de cent elements representant chacun un
// tableau de taille variable (cree dynamiquement).
// chaque element du tableau est une chaine de caracteres.
String[][] s = new String[100][];
for (int i=0; i<s.length; i++) {
    s[i] = new String[(i%16)+1]; // creation dynamique du ss-tableau
    for (int j=0; j<s[i].length; j++)
        s[i][j] = i + "/" + j;
}
```

► **Exercice 61** :

Écrire une fonction `trianglePascal` qui, étant donné un entier `n`, renvoie un tableau bidimensionnel triangulaire représentant le triangle de Pascal de hauteur `n+1`.

► **Exercice 62** (Examen 2008/2009, exercice 4) :

Toute suite finie d'entiers peut être décomposée de manière unique en une suite de séquences strictement croissantes maximales. En représentant une suite dans un tableau, le tableau

[1 2 5 7 2 6 0 5 2 4 6 7 8 9 3 4 6 1 2 7 8 9 4 2 3 1 5 9 7 1 6 6 3]

se décompose ainsi en le tableau de 13 tableaux d'entiers suivant :

[[1 2 5 7] [2 6] [0 5] [2 4 6 7 8 9] [3 4 6] [1 2 7 8 9] [4] [2 3] [1 5 9] [7] [1 6] [6] [3]].

Les premiers éléments de ces séquences sont, en plus du premier élément du tableau, les éléments (soulignés sur l'exemple) qui sont inférieurs ou égaux à leur prédécesseur dans le tableau.

1. Écrire une fonction `rupture` qui, étant donné un tableau `t` d'entiers, renvoie un tableau contenant `0` en premier élément et les indices des éléments de `t` inférieurs ou égaux à leur prédécesseur en ordre croissant. Pour le tableau donné en exemple, la fonction renvoie le tableau :

   [0 4 6 8 14 17 22 23 25 28 29 31 32].

2. Écrire une fonction `factorisation` qui, étant donné un tableau `t` d'entiers, renvoie un tableau bidimensionnel d'entiers, dont la `i`-ème ligne contient la `i`-ème plus longue séquence croissante de nombres adjacents dans le tableau `t` (résultat tel que celui donné dans l'exemple). On pourra utiliser la fonction `rupture` pour déterminer le nombre de lignes et la taille de chaque ligne de ce tableau bidimensionnel.

► **Exercice 63** (Examen 2009/2010, exercice 7) :

On considère une élection avec `n` candidats numérotés de `1` jusqu'à `n` et `m` électeurs numérotés de `0` jusqu'à `m-1`. Chaque électeur doit voter en donnant un classement complet des candidats dans l'ordre de sa préférence. Par exemple, pour `4` candidats, un vote correct d'un électeur est la suite `3,1,2,4` (on dit que `3` est premier, `1` est deuxième, etc), alors que `1,2,2,4` ou `1,3,2` ne sont pas corrects. Un vote d'un électeur est un tableau de `n` entiers. Une élection est un tableau de `m` votes.

1. Écrire une fonction `estUnVoteCorrect` qui teste si un vote est correct.
2. Écrire une fonction `estUneElectionCorrecte` qui teste si tous les votes d'une élection sont corrects.

Étant donné un vote, le score d'un candidat est `n` s'il est premier, `n-1` s'il est deuxième, etc. Le score d'un candidat pour une élection est la somme de tous les scores de chaque vote de l'élection. Le candidat avec le plus grand score gagne l'élection. En cas d'égalité entre deux ou plusieurs candidats, on regarde celui arrivant en premier le plus souvent. S'il y a toujours égalité, l'élection n'a pas de vainqueur.

3. Écrire une fonction `score` qui renvoie le score d'un candidat pour une élection.
4. Écrire une fonction `scores` qui renvoie le score de *tous* les candidats pour une élection (il s'agira ici de créer un tableau et d'y mettre les scores en parcourant le tableau d'élection qu'une seule fois).
5. Écrire une fonction `vainqueur` qui renvoie le numéro du candidat vainqueur s'il existe et `-1` sinon.
6. Écrire une fonction `main` qui permet de tester les fonctions précédentes.

► **Exercice 64** (Examen 2010/2011, exercice 6) :

On considère un tableau à 3 dimensions stockant les scores d'un championnat de handball. Pour `n` équipes, le tableau aura `n` lignes et `n` colonnes et dans chacune de ses cases on trouvera un tableau à une dimension de longueur 2 contenant le score d'un match (on ne tient pas compte de ce qui est stocké dans la diagonale). Ainsi, pour le tableau championnat `ch`, on trouvera dans `ch[i][j]` le score du match de l'équipe `i+1` contre l'équipe `j+1` et dans `ch[j][i]` le score du match de l'équipe `j+1` contre l'équipe `i+1`. De même, pour un score stocké, le premier entier de `ch[i][j]` sera le nombre de but(s) marqué(s) par l'équipe `i+1` dans le match l'opposant à l'équipe `j+1`. Finalement, on suppose que lorsqu'une équipe gagne un match, elle obtient 3 points, 1 seul point pour un match nul et 0 point dans le cas où elle perd le match.

1. Écrire une fonction `nombrePoints` qui prend en arguments un tableau championnat `ch` de côté `n` et le numéro d'une équipe (entre `1` à `n`) et qui renvoie le nombre de point(s) obtenu(s) par cette équipe pendant le championnat.
2. Écrire une fonction `stockerScore` qui prend en arguments un tableau championnat `ch` de côté `n`, le numéro `i` d'une équipe, le numéro `j` d'une autre équipe et le score du match de `i` contre `j` et qui met à jour le tableau `ch`.
3. Écrire une fonction `champion` qui prend en argument un tableau championnat `ch` et qui renvoie le numéro de l'équipe championne. Une équipe est championne si elle a strictement plus de points que toutes les autres. Si plusieurs équipes ont le même nombre de points, alors une équipe est meilleure si elle a marqué strictement plus de buts. Dans le cas d'égalité parfaite (même nombre maximum de points et même nombre maximum de buts marqués), la fonction renverra `0` pour signaler l'impossibilité de désigner un champion.

► **Exercice 65** (Examen 2010/2011, exercice 5) :

Sur le principe du pousse-pousse, deux joueurs (`o` et `x`) enfilent (par le haut) à tour de rôle des jetons (`o` et `x`) dans une grille carrée jusqu'à obtenir strictement plus d'alignement(s) (suivant les lignes, les colonnes et/ou les deux diagonales). Les jetons poussés hors de la grille (par le bas) sont remis en jeu. Dans l'exemple de partie ci-contre, le joueur `o` débute et gagne la partie.
La grille est codée par un tableau carré contenant les entiers `-1` (désignant un jeton `o`), `0` (une case vide) et `+1` (un jeton `x`).

```
Entrer la taille : 3
        . . .
        . . .
        . . .
Joueur o, quelle colonne ? 1
        . o .
        . . .
        . . .
Joueur x, quelle colonne ? 1
        . x .
        . o .
        . . .
Joueur o, quelle colonne ? 0
        o x .
        . o .
        . . .
Joueur x, quelle colonne ? 2
        o x x
        . o .
        . . .
Joueur o, quelle colonne ? 1
        o o x
        . x .
        . o .
Joueur x, quelle colonne ? 1
        o x x
        . o .
        . x .
Joueur o, quelle colonne ? 2
        o x o
        . o x
        . x .
Joueur x, quelle colonne ? 2
        o x x
        . o o
        . x x
Joueur o, quelle colonne ? 2
        o x o
        . o x
        . x o
Le joueur o gagne!
```

1a. Écrire une fonction `jeton` qui prend en argument un entier `k` et renvoie le caractère `'o'` si `k` est strictement négatif, renvoie le caractère `'x'` s'il est strictement positif et renvoie `'.'` sinon.

1b. Écrire une fonction `toString` qui prend en argument un tableau carré de `-1`, `0`, `+1` et renvoie une chaîne de caractères représentant la grille associée.

2. Écrire une fonction `jouer` qui prend en arguments un joueur (`-1` ou `+1`), un indice de colonne et un tableau carré et met à jour ce dernier en conséquence.

3a. Écrire une fonction `evaluation` qui prend en argument un tableau à une dimension de `-1`, `0`, `+1` et renvoie `-1` s'il s'agit d'un alignement de `-1`, renvoie `+1` s'il s'agit d'un alignement de `+1` et renvoie `0` sinon.

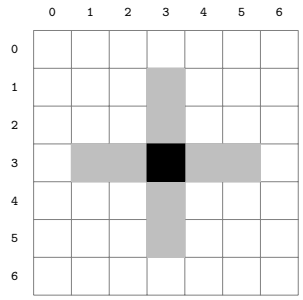
3b. Écrire des fonctions `diagonale`, `antidiagonale` et `colonne` qui prennent en argument un tableau carré (plus un indice `j` pour `colonne`) et renvoient le tableau correspondant respectivement à la diagonale (NO-SE), à l'antidiagonale (NE-SO) et à la colonne d'indice `j`.

3c. Écrire une fonction `evaluation` qui prend en argument un tableau carré de `-1`, `0`, `+1` et renvoie le nombre d'alignement(s) de `+1` moins le nombre d'alignement(s) de `-1`.

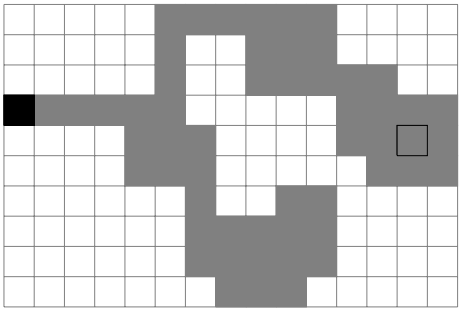
4. Écrire une fonction `main` qui demande la taille de la grille, puis demande alternativement à deux joueurs quelle colonne jouer en affichant à chaque fois la grille obtenue, et annonce le vainqueur dès que possible.

► **Exercice 66** (Examen 2011/2012, exercice 6) :

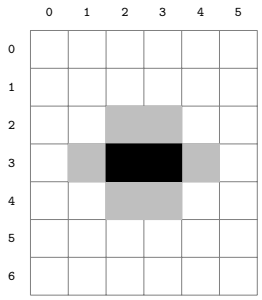
On considère un jeu de labyrinthe dans lequel se déplace une pièce de dimension $1\times1\times2$ en *roulant* sur ses propres arêtes : c'est un *pavé droit* ou *parallélépipède rectangle* (solide ayant 6 faces rectangulaires (dont 2 carrés ici), 8 sommets et 12 arêtes). Dessiné sur une grille, le labyrinthe est codé par un tableau `lab` bidimensionnel de booléens, la valeur `true` indiquant que la case est autorisée. À tout moment, la pièce est repérée par sa position `pos` (position sur la grille et position debout ou couchée) avec la convention suivante :
– debout sur la case autorisée (x,y), sa position est `2x | 2y` ;
– couchée sur deux cases autorisées (x,y) et $(x+1,y)$, sa position est `2x+1 | 2y` ;
– couchée sur deux cases autorisées (x,y) et $(x,y+1)$, sa position est `2x | 2y+1`.



la pièce (debout) en `6 | 6` peut rouler en `6 | 9`, `3 | 6`, `6 | 3` ou `9 | 6` (si les cases associées sont autorisées)



Dans cet exemple de labyrinthe, la pièce initialement en `6 | 0` doit parvenir en `8 | 26` : 35 coups minimum!



la pièce (couchée) en `6 | 5` peut rouler en `6 | 8`, `4 | 5`, `6 | 2` ou `8 | 5` (si les cases associées sont autorisées)

1. Écrire une méthode `estDedans` qui prend en arguments les dimensions d'un tableau `lab` et une position `pos` (selon la convention) et teste si la case ou les cases associées sont sur la grille rectangulaire.
2. Écrire une méthode `estValable` qui prend en arguments un tableau `lab` et une position `pos` (selon la convention) et teste si la case ou les cases associées sont autorisées.
3. Écrire une méthode `nouvelle` qui prend en arguments une position `pos` (selon la convention) et une direction `0`,`1`,`2` ou `3` (codant droite, haut, gauche ou bas) et renvoie la nouvelle position (sans se préoccuper ici de sa validité).
4. Écrire une méthode `afficher` qui prend en arguments un tableau `lab` et des positions `pos` et `arr` (selon la convention) supposées autorisées et affiche le labyrinthe en distinguant les cases `'o'` autorisées, la ou les cases `'*'` associées à la position `pos` et la case `'@'` associée à la position `arr` (supposée debout).
5. Écrire une méthode `jouer` qui prend en arguments un tableau `lab` et des positions `dep` et `arr` supposées debout (selon la convention) et, en partant de la position `dep`, à chaque coup, affiche le labyrinthe, demande la direction (jusqu'à obtenir une nouvelle position valable) et met à jour la position jusqu'à parvenir à `arr`.
6. Écrire une méthode `aleatoire` qui prend en arguments deux entiers `haut` et `larg` et un réel `dens` et qui renvoie un labyrinthe aléatoire jouable. Il sera construit sur une grille de hauteur `haut` et de largeur `larg` et aura un rapport de cases autorisées inférieur à `dens`. Toutes les deux debout, les positions de départ et d'arrivée seront situées au tiers et aux deux tiers de la diagonale nord-ouest sud-est de la grille.