

– POO1 – Travaux Pratiques – Série N°2

Partie I : Tableaux à une dimension

Exercice 1 : Recherche

Soit *tab* un tableau de *n* entiers (*int*).

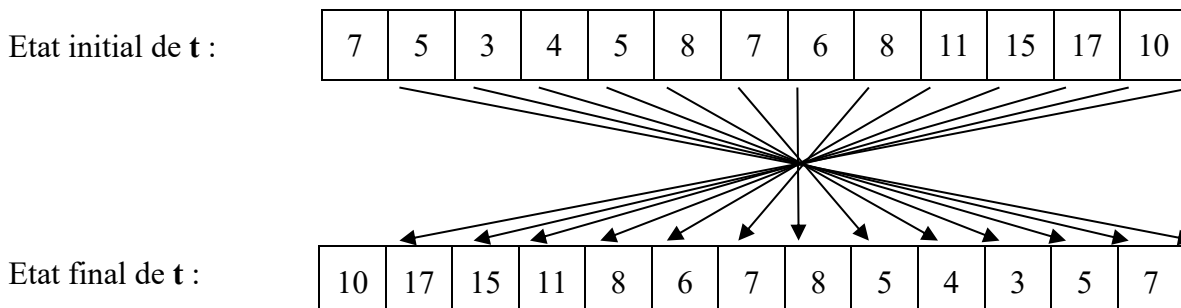
Ecrire un programme Java permettant de :

- ①- Lire la taille *n* du tableau
- ②- Créer le tableau
- ③- Saisir les éléments du tableaux (écrire une méthode *static void fill(int[] t, Scanner sc)*)
- ④- Afficher les éléments saisis (écrire une méthode *static void display(int[] t)*).
- ⑤- Ajouter la méthode *static int seek(int[] t, int val)* permettant de retourner la première position dans le tableau d'une valeur entière *val* si elle existe. Dans le cas contraire, la valeur **-1** est retournée. Tester cette nouvelle méthode.
- ⑥- Ecrire une autre méthode *static int seekAll(int[] t, int val, boolean d)* permettant de compter combien de fois la valeur entière *val* existe ($\equiv$ son nombre d'occurrences) et si *d = true*, les positions de la valeur *val* sont affichées. Tester cette méthode.
- ⑦- Ecrire une méthode *static int min(int[] t)* qui permet de chercher le minimum dans le tableau.
- ⑧- Ecrire une méthode *static int max(int[] t)* qui permet de chercher le maximum dans le tableau.
- ⑨- Ecrire une méthode *static float average(int[] t)* qui permet de calculer la moyenne des éléments du tableau.

Exercice 2 : Inversion / Tableau Symétrique

Q1) Ecrire une méthode *static void swap(int[] t)* qui permet d'inverser les éléments d'un tableau *t*. C'est-à-dire, le premier élément devient dernier, et le dernier devient premier. Le deuxième devient l'avant dernier et l'avant dernier devient deuxième, et ainsi de suite. Voir le schéma ci-dessous.

Exemple : Si N = 13



Remarque : Il est interdit d'utiliser un deuxième tableau. Tous les traitements doivent être effectués sur le tableau *t*.

Q2) Ecrire une méthode *static boolean isSymmetric(int[] t)* qui vérifie si un tableau *t* est symétrique. C'est-à-dire, le premier élément est égal au dernier, le deuxième est égal à l'avant dernier et ainsi de suite.

Exercice 3 : Tri par sélection (tri par extraction)

Le principe du tri par sélection est le suivant :

On commence par parcourir le tableau pour trouver le minimum. On le place à l'indice 1.

Ensuite, on recommence à parcourir le tableau à partir de l'indice 2 pour trouver le minimum que l'on place à l'indice 2. Et ainsi de suite pour l'indice 3, 4 jusqu'à $N - 1$. A la fin de l'algorithme, le tableau devient trié : le minimum se trouve au début du tableau (la position 1) et le maximum se trouve à la fin du tableau (la position N). **Question)** Ecrire un programme Java permettant d'effectuer le tri par sélection.

Partie II : Tableaux à deux dimensions

Exercice 4 : Recherche dans une matrice

Refaire les questions de l'exercice 1 pour le cas d'un tableau à deux dimensions.

Exercice 5 : Matrice transposée / Matrice Symétrique

Q1) Ecrire une méthode *static void swap(int[][] mat)* qui permet de calculer la matrice transposée de la matrice *mat*. La matrice transposée est obtenue en échangeant les lignes et les colonnes de la matrice.

Q2) Ecrire une méthode *static boolean isSymmetric(int[][] mat)* qui vérifie si une matrice carrée *mat* est symétrique ou non. Une matrice est symétrique si chaque ligne *i* est égale à la colonne *i*. En d'autres termes, chaque élément $mat_{ij} = mat_{ji}$.

La matrice ci-dessous est symétrique :

15	3	7	2
3	11	4	37
7	4	23	18
2	37	18	12

Exercice 6 : Triangle de PASCAL

Le triangle de Pascal est représenté par une matrice triangulaire *P* dont les éléments p_{ij} sont définis comme suit :

✓ $p_{ij} = 1$ si $j=1$ (la première colonne est toujours à 1).

✓ $p_{ij} = 1$ si $i=j$ (la diagonale est toujours à 1).

✓ $p_{ij} = p_{i-1, j-1} + p_{i-1, j}$ si $j > 1$ et $j < i$ (l'élément de la ligne *i* et colonne *j* s'obtient en ajoutant les valeurs des éléments de la ligne *i - 1* et colonne *j - 1* et de la ligne *i - 1* et colonne *j*).

Voici un triangle de Pascal de dimension $N = 6$.

En utilisant une matrice triangulaire, écrire un programme Java permettant de générer automatiquement le triangle de Pascal pour une taille *N* donnée.

P	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	1	5	10	10	5	1