Lille iut Université de Lille

Contrôle TP du 22 novembre 2017

M3103 – Algorithmique avancée
DUT info 2ème année
2017–2018

Dans cet exercice, nous allons programmer une file de priorité à l'aide d'un tableau à deux dimensions. Il n'est, bien sûr, pas nécessaire de faire les questions dans l'ordre. Pour rappel, l'interface correspondant à cette file est la suivante :

```
public interface FilePriorite<E> extends Iterable<E> {
    /** Removes all of the elements from this priority queue. */
    void clear();

    /** Returns the comparator used to order the elements in this queue,
       or null if this queue is sorted according to the natural ordering of its elements. */
    Comparator<? super E> comparator();

    /** Inserts the specified element into this priority queue. */
    boolean offer(E e);

    /** Retrieves, but does not remove, the head of this queue,
        or returns null if this queue is empty. */
    E peek();

    /** Retrieves and removes the head of this queue, or returns null if this queue is empty. */
    E poll();

    /** Returns the number of elements in this collection. */
    int size();

    /** Returns <code>true</code> if this collection contains no elements. */
    boolean isEmpty();
}
```

Dans ce type abstrait, la valeur à sortir (`poll`) est toujours la valeur la plus prioritaire ; pour nous, la valeur la plus prioritaire sera toujours la plus grande au sens de la comparaison déclarée à la construction de notre file de priorité.

L'implémentation choisie pour la file est une matrice carrée possédant la propriété suivante : pour toute case (i, j) de la matrice, la valeur contenue dans cette case est supérieure aux valeurs contenues dans ses voisines de droite $(i, j+1)$ et du dessous $(i+1, j)$. Dans cette matrice, chaque ligne est donc en ordre décroissant de gauche à droite, et chaque colonne de haut en bas.
De plus, le remplissage de cette matrice se fera toujours de gauche à droite, puis quand une ligne est pleine, en passant à la ligne suivante. De ce fait, la matrice ne peut avoir que des lignes pleines sauf éventuellement sa dernière ligne non vide.

Voici par exemple une telle matrice :

18	10	1
14	5	

Question 1 : Créez la classe `MatTas<E>` implémentant l'interface `FilePriorite<E>`, avec les attributs que vous jugerez nécessaires. Votre classe **doit** comporter un commentaire Javadoc qui précise le rôle de la classe que vous implémentez. Vous y **préciserez** en autre la façon que vous avez choisie pour connaître le nombre d'éléments présents dans votre `MatTas<E>` et si une case de la matrice peut être vide (`null`). N'oubliez pas le tag `@author` !

Question 2 : Vous y ajouterez deux constructeurs :

1. Un constructeur sans paramètre : la taille par défaut [1] de la matrice est alors 3×3.
2. Un constructeur avec une taille indiquée.

Question 3 : Programmez les méthodes de l'interface : `clear`, `isEmpty`, `size` et `peek`.

Question 4 : Pour la méthode `offer`, il faut ajouter la valeur sur la dernière ligne occupée, sur la première case libre. Il vous suffit juste de rétablir la propriété d'ordre entre les lignes et colonnes en permutant les valeurs avec son voisin de gauche ou du dessus. Réfléchissez à la permutation la plus intelligente en vous inspirant des tas (cf CM7), puis programmez la méthode `boolean offer(E e)`. Cette méthode retourne `false` s'il n'y a plus de place dans le conteneurs.

1. Pensez à déclarer une constante !!!

Question 5 : Pour la méthode `poll`, la valeur de retour est la valeur située en première ligne et première colonne. Puis vous devez supprimer celle-ci du `MatTas` : vous la remplacez par la dernière valeur du `MatTas` (dernière ligne, dernière colonne occupée). Ensuite grâce à des permutations, vous rétablissez la propriété d'ordre du `MatTas`, de la même façon que vous supprimez une valeur d'un tas.

Indication : *Pour les questions 4 et 5, il peut-être très judicieux de définir deux méthodes privées pour vous assister dans la descente ou la remontée des valeurs. Ces deux méthodes peuvent retourner un couple d'index si vous retournez les coordonnées de la case choisie, ou plus simplement un booléen qui discriminera les deux cases potentielles pour la permutation. Les signatures de telles méthodes pourraient être :*

```
coordOfMaxNeighbor(int i, int j)
coordOfMinNeighbor(int i, int j)
```

ou bien :

```
isLeftTheHighest(int i, int j)
isRightTheLowest(int i, int j)
```

Question 6 : Nous ajoutons maintenant la possibilité de décider de la notion de priorité à la construction de la `MatTas<E>`. Ajoutez un attribut de type `Comparator<? super E>` puis programmez une méthode `compare(E e1, E e2)` qui compare les deux éléments passés en paramètre. Cette méthode qui respecte la convention usuelle du Java pour les comparaisons, utilise le `Comparator` s'il n'est pas `null`, sinon elle utilise l'ordre naturel des éléments. Si aucune de ces deux possibilités n'est disponible, la méthode soulève une exception de type `ClassCastException`.

Question 7 : Créez une classe interne qui implémente `Iterator<E>` et ajoutez une méthode `iterator()` qui retourne un itérateur sur la `MatTas`. Quels sont les attributs nécessaires pour désigner une case de la `MatTas<E>` ?

Question 8 : Ajoutez deux constructeurs à votre `MatTas<E>` :

1. Un constructeur de `MatTas<E>` à partir d'une autre `FilePriorite<E>`
2. Un constructeur de `MatTas<E>` à partir d'un ensemble ordonné de valeurs (`SortedSet<E>`)

Pour ces deux constructeurs, essayez de minimiser le nombre d'opérations nécessaires à la construction de votre file.

Pour rendre votre TP :

1. Vérifiez que vous avez bien mis les commentaires javadoc.
2. Faites une archive ZIP ou TGZ **à votre nom** des fichiers sources **uniquement**.
3. Envoyer cette archive à votre enseignant de TD par email, avec comme entête [SDD] CTP. Les adresses mail sont les suivantes :

Groupe	Email
K	Frederic.Guyomarch@univ-lille1.fr
L	Patricia.Everaere@univ-lille1.fr
M	Mikael.Mousse@gmail.com
N	Frederic.Guyomarch@univ-lille1.fr

Pour illustrer les algorithmes d'ajout et de retrait, voici ce qu'il se passe alors de l'appel de `offer` pour les valeurs 5, 10, 1, 18, 14 à partir d'une file vide :

`offer(5)`

5		

`offer(10)`

5	10	

→

10	5	

`offer(1)`

10	5	1

`offer(18)`

10	5	1
18		

→

18	5	1
10		

`offer(14)`

18	5	1
10	14	

→

18	14	1
10	5	

Puis le retrait de toutes les valeurs par appel à `poll`. La première illustration de chaque ligne correspond au remplacement du maximum par la dernière valeur de la `MatTas`, puis permutations pour rétablir la propriété d'ordre.

`poll`

5	14	1
10		

→

14	5	1
10		

`poll`

10	5	1

`poll`

1	5	

→

5	1	

`poll`

1		

`poll`
