INF2010 - Structures de données et algorithmes

Travail Pratique #5
File de priorité
Hiver 2016

File de priorité exploitant un monceau (5 pts)

Il existe différentes manière d'implémenter une file de priorité. Celle recourant à un monceau a été discutée en classe. Un monceau est un arbre binaire complet ordonné en tas, c'est-à-dire un arbre binaire complet obéissant à la définition récursive voulant que tout nœud dans l'arbre possède une clé dont la valeur est inférieure ou égale à l'ensemble des clés de ses enfants.

Tel que vu en classe, la construction d'un monceau peut s'effectuer au moins de deux manières :

- En insérant les éléments un à un dans le monceau.
- En insérant l'ensemble des éléments à la fois ; autrement dit, partant d'un tableau, ordonner en tas les éléments dans le tableau.

Néanmoins, l'implémentation vue en classe ne permet pas de modifier la clé d'un élément du monceau.

Objectifs :

- Compléter l'implémentation de la classe `HeapPriorityQueue` fournie avec l'énoncé.
- Implémenter les différentes techniques de construction d'un monceau.
- Pour toutes ces méthodes, adjoindre au monceau une mappe permettant de retrouver ses éléments et d'en modifier la priorité.

Partie I : File de priorité implémentée au moyen d'un monceau

Pour cette partie du laboratoire, vous mettrez `testPartII = false;` dans `TestPriorityQueue` pour tester vos fonctions.

Exercice 1 : Ajouter un élément à la file de priorité (0.5 point)

Complétez la méthode `add1(AnyType x, int priority)` qui permet d'ajouter l'élément `x` à la file de priorité avec une priorité `priority`. Pour mémoire, plus la valeur de `priority` est petite, plus l'élément `x` est prioritaire. On supposera la valeur de `priority` positive (0 inclus).

Dans votre implémentation, vous devez renvoyer un `IllegalArgumentException` si la valeur de `priority` est négative. Vous retournerez un `NullPointerException` si `x` est `null`. Ne vous préoccupez pas de `indexMap` pour cette partie du laboratoire.

Exercice 2 : Retirer l'élément le plus prioritaire de la file (0.5 point)

Complétez la méthode `poll1()` qui permet de retirer l'élément le plus prioritaire de la file de priorité. Pour ce faire, il vous faudra aussi compléter la méthode `percolateDown1(int hole)`. Ne vous préoccupez pas de `indexMap` pour cette partie du laboratoire.

Exercice 3 : Construction d'un monceau min depuis un tableau (0.5 point)

Le constructeur `HeapPriorityQueue( AnyType[] items, int[] priorities )` prend en entrée deux tableaux (les éléments et leur priorité), recopie les données de `items` puis manipule le tableau interne pour en obtenir un monceau, ce qui est fait au moyen de la fonction `buildHeap1(…)` pour cette partie.

Complétez `buildHeap1(…)` qui devra exploiter la fonction `percolateDown1(…)` que vous aviez précédemment implémentée. Vous retournerez un `NullPointerException` si un des deux tableaux est `null` ou si `items` contient un élément `null`. Renvoyez un `IllegalArgumentException` si `priorities` contient une valeur négative. Ne vous préoccupez pas de `indexMap` pour cette partie du laboratoire.

Exercice 4 : Retourner les éléments les moins prioritaires (0.5 point)

Complétez la méthode `getMax()` qui retourne une liste contenant les éléments les moins prioritaires de la file de priorité. Retournez une liste vide si la file l'est également.

Partie II : Modification des priorités

Pour cette partie du laboratoire, vous mettrez vous mettrez `testPartII = true;` dans `TestPriorityQueue` pour tester vos fonctions. Il vous est demandé également de commenter/décommenter les appels à `add1`/`add2` et `poll1`/`poll2` `buildHeap1`/`buildHeap2` dans les méthodes `add`, `poll` et le constructeur avec paramètres.

Exercice 5 : Ajouter un élément à la file de priorité (bis) (0.5 point)

Complétez la méthode `add2(AnyType x, int priority)` qui permet d'ajouter l'élément `x` à la file de priorité avec une priorité `priority`. Votre implémentation devrait ressembler à celle réalisée à l'exercice 1, à la différence que vous devez maintenir la mappe des indices des éléments (et par conséquent interdire les doublons) sans affecter la complexité de la méthode ($O(1)$ en cas moyen, $O(\lg(n))$ en pire cas). Pour ce faire, vous utiliserez les méthodes `put(…)` et `replace(…)` de la classe `Hashtable`, dont `indexMap` est une instance.

Exercice 6 : Retirer l'élément le plus prioritaire de la file (bis) (0.5 point)

Complétez la méthode `poll2()` qui permet de retirer l'élément le plus prioritaire de la file de priorité. Votre implémentation devrait ressembler à celle réalisée à l'exercice 2, à la différence que vous devez maintenir la mappe des indices des éléments. Pour ce faire, il vous faudra aussi compléter la méthode `percolateDown2(int hole)`. Assurez-vous de maintenir la mappe des indices à jour (gérer `indexMap`) sans affecter la complexité de la méthode ($O(\lg(n))$ en tout cas).

Exercice 7 : Construction d'un monceau min depuis un tableau (1.0 point)

Complétez `buildHeap2(…)` qui devra exploiter la fonction `percolateDown2(…)` que vous aviez précédemment implémentée. Vous devez maintenir la mappe des indices des éléments. La file de priorité n'admettant pas de doublons, en cas de répétition d'un élément, sa priorité sera la première rencontrée dans le tableau.

Exercice 8 : Mettre à jour la priorité d'un élément (1.0 point)

Complétez le code de `updatePriority(…)` qui permet de modifier la priorité d'un élément de la file de priorité. La complexité de votre implémentation devra être $O(\lg(n))$. Pour ce faire, vous procéderez ainsi : *i*) retrouver l'indice de l'élément recherché en vous aidant de `indexMap` ; *ii*) placer l'élément à la racine ; *iii*) retirer l'élément ; *iv*) le réinsérer pour qu'il se place à la bonne position. Assurez-vous de maintenir `indexMap` à jour durant vos manipulations.

Instructions pour la remise :

Le travail doit être fait par équipe de 2 personnes et doit être remis via Moodle au plus tard le 4 avril avant 23h50.

Veuillez envoyer vos fichiers .java **seulement**, dans un **seul répertoire**, le tout dans une archive de type *.**zip** (et seulement **zip**, pas de ~~**rar**~~, ~~**7z**~~, etc) qui portera le nom :

inf2010_lab5_MatriculeX_MatriculeY.zip, où MatriculeX < MatriculeY.

Les travaux en retard seront pénalisés de 20 % par jour de retard. Aucun travail ne sera accepté après 4 jours de retard. Si votre dépôt ne respecte pas la nomenclature définie ci-dessus, 0.5 point de pénalité sera appliqué.