

INF2010 - Structures de données et algorithmes

Travail Pratique #4

Monceaux

Automne 2018.

Objectifs :

- Implémenter les différentes techniques de construction d'un monceau.
- Implémenter une méthode pour retirer la racine d'un monceau
- Implémenter une fonction pour afficher le monceau sous forme d'arbre.
- Tester votre implémentation

Initiation :

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 où :

- La valeur de la clé est \leq à l'ensemble des clés de ses enfants dans le cas d'un *min-heap*.
- La valeur de la clé est \geq à l'ensemble des clés de ses enfants dans le cas d'un *max-heap*.

La construction d'un monceau peut s'effectuer d'au moins deux manières :

1. En insérant les éléments un à un dans le monceau.
2. En créant un arbre binaire complet contenant tous les éléments et en ordonnant progressivement les éléments de l'arbre jusqu'à l'obtention d'un monceau.

Exercice 1 : Construction d'un monceau (0.5 point)

Complétez la méthode `offer(AnyType x)` qui permet d'insérer l'élément `x` dans le monceau.

Remarque : notez que les éléments doivent être stockés à partir de la case 1 et non pas 0 dans le tableau interne.

Exercice 2 : Construction d'un monceau depuis un tableau (1 point)

Le constructeur `BinaryHeap(AnyType[] items, boolean min)` prend en entrée un tableau et un booléen indiquant s'il s'agit d'un *min-heap* ou d'un *max-heap*. Le constructeur copie les données dans un tableau interne, puis manipule le tableau interne pour obtenir le type de monceau désiré en faisant appel à `buildMinHeap()` ou `buildMaxHeap()` selon le cas.

Complétez `buildMinHeap()` et `buildMaxHeap()` au moyen des fonctions `percolateDownMinHeap()`, `percolateDownMaxHeap()` et `leftChild()`.

Exercice 3 : Retrait de la racine d'un monceau (0.75 point)

Complétez la méthode `poll()` qui permet de retirer l'élément en tête du monceau.

Exercice 4 : Implantation d'un itérateur *fail-fast* (0.5 point)

Les monceaux sont souvent utilisés pour implémenter des files de priorités. D'ailleurs, le monceau binaire de Java se nomme `PriorityQueue` et hérite de la classe abstraite `AbstractQueue`. Afin de fournir une implantation compatible avec les standards de Java, la classe `BinaryHeap` hérite aussi d'`AbstractQueue`. En plus des méthodes `peek()`, `poll()` et `offer()`, `AbstractQueue` impose l'implantation de la méthode `iterator()`.

Pour ce TP, on vous impose une contrainte supplémentaire : votre itérateur doit être *fail-fast*. C'est-à-dire qu'il doit soulever une exception de type `ConcurrentModificationException` s'il détecte une modification au monceau (insertion ou délétion) en cours d'itération. La vérification doit se faire au moment de l'appel à la méthode `next()`.

Compléter le `HeapIterator`. Utilisez la variable de classe `modifications` afin de détecter s'il y a des modifications en cours d'itération. L'itération peut se faire dans un ordre arbitraire.

Exercice 5 : Tri par monceau (0.75 points)

La class `BinaryHeap` offre une fonction statique de tri `heapSort()` qui reçoit un tableau et le trie suivant la technique du tri par monceau. On vous demande de la compléter en faisant appel à `percolateDownMaxHeap()`. Inversement, complétez `heapSortReverse()` en faisant appel à `percolateDownMinHeap()`.

Exercice 6 : Affichage du monceau en arbre (0.75 points)

Complétez le code de `nonRecursivePrintFancyTree()` permettant d'afficher le monceau sous la forme d'un arbre en n'effectuant aucun appel récursif. Référez-vous au document `Affichage.txt` pour voir le résultat attendu.

Exercice 7 : Vérification de votre implantation (0.75 points)

Complétez le fichier `Main.java` afin de vérifier le bon fonctionnement de votre structure de données. Par exemple, assurez-vous que `poll()` retourne toujours l'élément minimal ou maximal, vérifiez que votre itérateur détecte bien tous les types de modifications concurrentes, etc. Notez que puisque votre structure implémente les mêmes méthodes que la `PriorityQueue` de Java, il est aisé d'effectuer les mêmes suites d'opérations sur votre `BinaryHeap` et sur une `PriorityQueue` et de vérifier la concordance entre les deux structures.

Instructions pour la remise :

Le travail doit être fait par équipe de 2 personnes et doit être remis via Moodle au plus tard le :

- 15 Novembre avant 23h55 pour le groupe 01.
- 08 Novembre avant 23h55 pour le groupe 02.
- 14 Novembre avant 23h55 pour le groupe 03.
- 07 Novembre avant 23h55 pour le groupe 04.
- 18 Novembre avant 23h55 pour le groupe 05.

Veillez envoyer vos fichiers dans une archive de type ***.zip** (et seulement **zip**, pas de ~~rar~~, ~~7z~~, etc) qui portera le nom :

inf2010_lab4_MatriculeX_MatriculeY.zip, où $\text{MatriculeX} < \text{MatriculeY}$.

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.