INF2010 – Structures de données et algorithmes

Automne 2021 Travail Pratique 4

Monceaux

POLYTECHNIQUE MONTRÉAL

Objectifs du laboratoire :

1. Comprendre le fonctionnement d'un monceau (heap)
2. Comprendre la complexité asymptotique des opérations d'un monceau
3. Utiliser un monceau pour résoudre un problème d'entrevue commun

Ce laboratoire aura pour objectif de vous familiariser avec **les monceaux** et la matière du **cours 7**. Le travail à effectuer sera identifier dans le code avec les « TODO ». Les « TODO » peuvent indiquer deux situations :

1. Compléter du code existant
2. Faire l'implémentation de la méthode entière

Le laboratoire sera évalué à l'aide de tests ainsi qu'un survol du code par le chargé. Veuillez-vous référer à la section « barème de correction » pour de l'information additionnel sur la correction.

Format de la remise :

Remettre dans un dossier nommé : *tp4_MatriculeX_MatriculeY.zip* avec tous les fichiers du laboratoire à remettre.

Important : Les travaux mal nommés ou avec des fichiers supplémentaires auront une pénalité de 20% sur le travail. Les travaux en retard seront pénalisés de 20% par jour de retard

Révision : Monceaux

Un monceau se résume à un arbre binaire de recherche complet. Le site web suivant peut aider à la visualisation du monceau :

https://visualgo.net/en/heap

Pour des explications en détail, veuillez-vous référer à la matière contenue dans les acétates du **cours 7**.

Travail à effectuer :

Partie 1 : Implémenter un monceau

Constructeur du monceau :
La liste des éléments devrait être initialisé et on devrait faire appel à *buildHeap*.

leftChildIndex :
Retourner l'index de l'enfant gauche du nœud.

rightChildIndex :
Retourner l'index de l'enfant droite du nœud.

parentIndex :
Retourner l'index du nœud parent.

isLeaf :
Retourner *vrai* ou *faux* si l'élément est une feuille.

buildHeap :
Construire le monceau en utilisant la méthode percolateDown.

percolateDown :
Compléter l'implémentation de la méthode (référez-vous aux notes de cours 7).

swap :
Cette méthode prend deux indexes. Ensuite elle va échanger la position des nœuds qui se retrouvent aux indexes de paramètres. La modification devrait se faire dans *elements*.

insert :
Selon le type de monceau (max / min), lors de l'insertion on devrait ajouter un nœud à la fin de la liste *éléments* et ensuite comparer les nœuds parents et modifier leur emplacement à l'aide de *swap*.

getLeftElements :
Retourner une liste des nœuds gauche du monceau en utilisant la méthode *leftChildIndex.*

getRightElements :
Retourner une liste des nœuds droite du monceau en utilisant la méthode *rightChildIndex.*

getParentElements :
Retourner une liste des nœuds parent du monceau en utilisant la méthode *parentIndex.*

getLeaves :
Retourner une liste des nœuds feuilles du monceau en utilisant la méthode *isLeaf*.

Pour les méthodes *buildHeap, percolateDown* et *insert* indiquez la complexité de l'opération en commentaire au-dessus dans la méthode.

Information importante pour partie 2 :

Dans le langage de programmation Java, l'équivalent du monceau s'appelle *PriorityQueue*. Vous pouvez utiliser le lien suivant pour mieux comprendre l'interface :
https://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html

Partie 2 : Utiliser un monceau afin de résoudre un problème complexe

Maintenant que vous comprenez mieux comment un monceau fonctionne, essayez de résoudre le problème suivant :

Vous devez utiliser un PriorityQueue pour résoudre ce problème

Problème :

Vous recevez en paramètre une collection d'entier (Integer). Chaque entier dans la collection représente le poids d'une boîte.

À chaque itération, on prend **les deux boîtes les plus lourdes** et les deux vont entrer en collision. Vous pouvez considérer $boîte_1 <= boîte_2$.

Après une collision il a deux résultats possibles :

- $Boîte_1 == Boîte_2$: les deux boîtes sont détruites
- $Boîte_1 != Boîte_2$: la $boîte_1$ sera détruite et la $boîte_2$ a maintenant le poids de $Boîte_2 = Boîte_2 - Boîte_1$

À la fin du jeu, il aura **au plus** une boîte restante.

Retourner le plus petit poids de la boîte. S'il n'a plus de boîte, retourner 0.

Indiquez la complexité de la solution au-dessus de la méthode.

Exemple :

Boîtes : [3,2,1,4]

Itération 1:
3 et 4 sont combinés pour devenir 1.
Le tableau devient [2,1,1]

Itération 2:
2 et 1 sont combines pour devenir 1.
Le tableau devient [1,1]

Itération 3:
1 et 1 sont combines pour devenir 0.
Le tableau devient []

Retourner 0.

Dates de remise :

Groupe B2: 25 novembre à 23h59

Groupe B1: 2 décembre à 23h59

Barème de correction :

Partie 1 : 13 points

Tests partie 1 + correction manuelle : / 8

Complexité : /3

Qualité du code : /2

Partie 2 : 7 points

Tests partie 2 + correction manuelle : / 4

Complexité : /2

Qualité du code : /1

Note : - /20