UNIVERSITÉ DE MONTRÉAL
Département IRO

Été 2016
Neil Stewart

IFT 2015 E16

Devoir 3.

10/10, soit 10% de la note finale.

Les 10 points "Partie pratique" pour le cours E16 ont été distribués de la façon suivante : Devoir #1 (1 point), Devoir #2 (3 points), Devoir #3 (6 points).

1 Partie Pratique (6 points)

Dans ce devoir, nous allons implanter les deux versions de l'algorithme de Prim (sans et avec monceau), afin de trouver l'arbre sous-tendant minimal dans un graphe simple non-orienté. On vous met à disposition les classes suivantes :

- *TableElement* : un objet de ce type contient de l'information sur un noeud dans la table utilisée par Prim. Plus précisement ; si un noeud a été visité, la distance minimale, et son parent.
- *HeapElement* : un objet de ce type est un élément du monceau et il contient la priorité d'un noeud (plus la distance est petite, plus le noeud est prioritaire).
- *Heap* : un objet de ce type représente le monceau, contenant des éléments de type *HeapElement*.
- *Devoir3* : La classe principale qui contient la méthode *main()*.

Tâches :

- (1.5 points) Implanter le constructeur ainsi que les méthodes *percolateDown*, *deleteMin* et *insert* de la classe *Heap*. Vous pouvez commencer le monceau à partir de 0 ou 1, selon votre choix.
- (3 points) Implanter les méthodes *Prim()* et *HeapPrim()* de la classe *Devoir3*.
- (0.5 points) Tester votre code pour le fichier data.in (graphe complet) et data2.in (graphe creux). Exécuter le code 10 fois, et reporter la moyenne de temps d'exécution pour chaque fichier. Quelle méthode prend le moins de temps ? Comment expliquez-vous cela ? Ecrivez un petit rapport de **maximum 1/2 page** .

Vous pouvez tester votre code en utilisant les fichiers data.in et data2.in disponibles sur le site *StudiUM*.

Modalités de remise : envoyer une archive <nom1-prenom1_nom2-prenom2.zip> à l'adresse teodora.dan@umontreal.ca.

2 Partie Théorique (3 points)

1. (1 point)

   Soit le *Type Abstrait de Données* (TAD) suivant :
   Liste avec éléments de la forme $L = \{x_1, \ldots, x_N\}$, $N \geq 1$, et $L = \Lambda$ (liste vide). La longueur d'une liste est égale à N, $N \geq 1$, la longueur de la liste vide est 0.

   Les opérations dans le TAD sont :

   — *Initialiser* (initialiser la liste L : $L \leftarrow \Lambda$)
   — *Vide ?* (décider si L est vide).
   — *Longueur* (trouver la longueur de L)
   — *Accéder(i)* (accéder à x_i, $1 \leq i \leq N$)
   — *Remplacer(i)* (remplacer la valeur emmagasinée à x_i par une nouvelle valeur, $1 \leq i \leq N$)
   — *Retirer(i)* (retirer x_i de L)
   — *Insérer* (insérer un nouvel élément dans la liste dans une position arbitraire : avant x_1, après x_N, entre x_j et x_{j+1}, $1 \leq j \leq N-1$)

   Prenons comme méthode la structure Arbre-AVL, avec un champs supplémentaire dans chaque noeud qui contient toujours (1 + le nombre de noeuds dans le sous-arbre gauche).

   Identifiez les opérations (s'il y en a) qui ne peuvent pas se réaliser dans un temps $O(\log N)$. (La valeur "emmagasinée" à x_i est emmagasinée par référence, donc le remplacement même d'une valeur se fait en $O(1)$.)

   (Si vous donnez des justifications *très* brèves dans chaque cas, une réponse fausse pourrait justifier une note partielle de 0.5 point.)

2. (1 point)

   Dans la méthode de chaînage externe, nous pouvons imaginer garder les listes triées. Quels sont les coûts (coût moyen) en fonction de la longueur de la liste $\lambda = N/M$ dans les deux cas (triées et non-triées) pour les opérations :

   — Insertion dans le cas où nous savons que la clef est absente.
   — Recherche quand nous ne savons pas si la clef est présente. La réponse ici varie peut-être, dépendant du cas : *a posteriori* la clef est présente, ou la clef n'est pas présente.

3. (1 point)

   Dans les arbres cousus ("Threaded tree") , nous avons remplacé les références nulles par des ficelles vers des prédecesseurs et des successeurs.

   (a) S'il y a N noeuds avec clef dans l'arbre cousu, il y a combien de ficelles ?

   (b) En vous servant de la réponse à la partie 3a, montrez que la complexité d'un parcours in-order (GRD) d'un arbre avec ficelles se fait en $O(N)$.

À réaliser en équipes de 1 ou 2. À remettre le 13 juillet, 2016, avant `9:00`. Les solutions seront affichées le 13 juillet. Les devoirs en retard ne seront pas acceptés.