INF2010 - Structures de données et algorithmes
Automne 2016
Travail Pratique 5
Monceaux

Objectifs:

Implémenter une structure d'arbre binomial
Implémenter un monceau binomial

Instructions générales

Dans ce TP, contrairement au TP précédent, aucun fichier main ne vous est fourni. C'est à vous de l'écrire, si vous souhaitez tester votre programme (ce qui est **très** vivement conseillé).

Ce fichier ne sera pas évalué, mais je vous recommande d'y porter une attention toute particulière. Par ailleurs, je vous conseille de l'écrire avant d'écrire les méthodes demandées, afin de conformer votre implémentation au cahier des charges plutôt que de conformer le cahier des charges à votre implémentation.

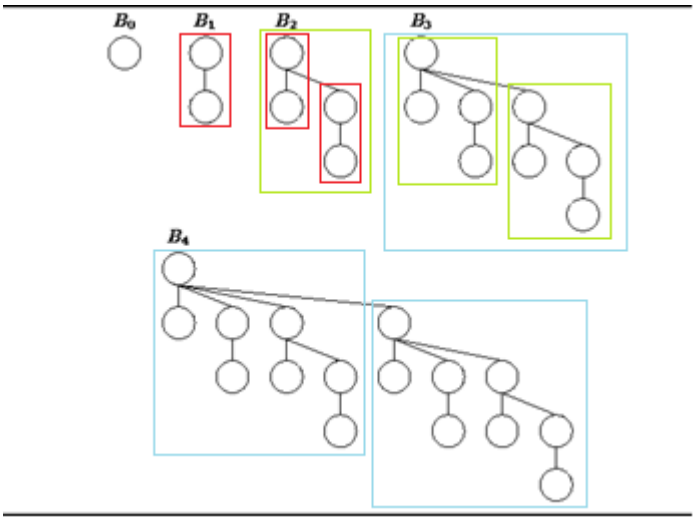
Monceau binomial

Vous avez vu en cours comment créer un monceau à partir d'un arbre binaire. Un inconvénient majeur d'un tel monceau est que la fusion de deux monceaux est une opération longue (O(n), où n est le nombre d'éléments). Nous allons ici nous intéresser à une autre implémentation possible, le monceau binomial, permettant une fusion beaucoup plus rapide.

Pour rappel, un monceau min (respectivement max) respecte la propriété que les enfants d'un nœud ont une valeur plus grande (resp. petite) que celle du parent. Ainsi, le nœud racine contient la valeur la plus petite (resp. grande) de l'arbre. Nous allons ici implémenter un monceau min.

Un monceau binomial est en fait une collection d'**arbres binomiaux**. Un arbre binomial peut se définir par récurrence :

- Un arbre binomial d'ordre 0 est constitué uniquement de la racine
- Un arbre binomial d'ordre k est constitué de deux arbres d'ordre k-1, la racine de l'un étant l'enfant de la racine de l'autre



Source : http://www.brpreiss.com/books/opus4/html/page371.html

Pour pouvoir construire un monceau binomial, ces arbres doivent en plus respecter la propriété d'ordre des monceaux énoncée plus haut.

La première étape de ce TP est d'implémenter une telle classe. Pour simplifier le travail, on ne va travailler ici qu'avec des arbres d'entiers.

Exercice 1 : Création d'une classe d'arbre binomial

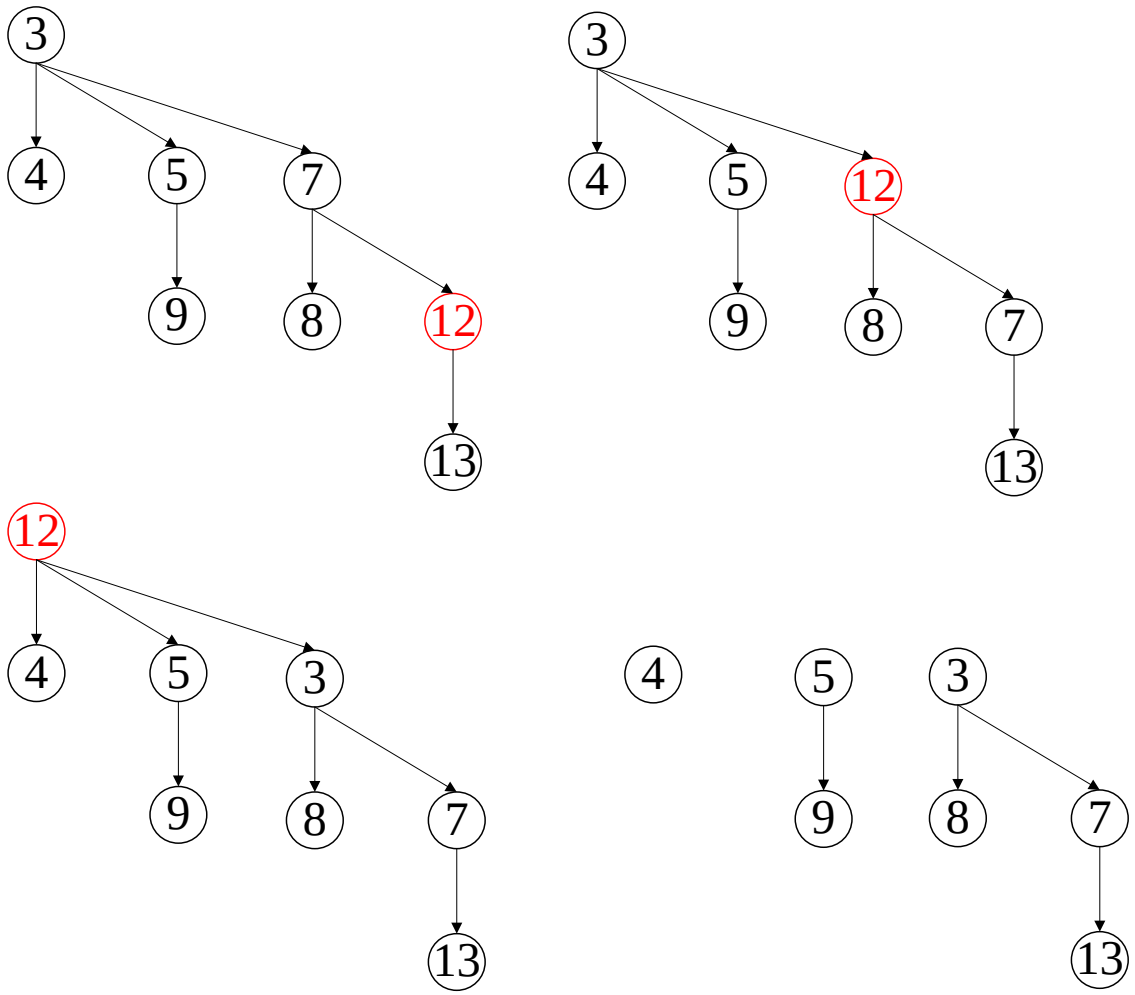
Les propriétés de l'arbre binomial font qu'il ne peut avoir que n = 2^k éléments, k étant son ordre. On ne peut donc pas insérer ou supprimer les éléments un à un. Par contre, il est très facile de fusionner deux arbres d'ordre k pour former un arbre d'ordre k+1.

On vous fournit une classe Node, représentant un nœud d'arbre binomial. L'arbre lui-même est tout simplement représenté par le nœud racine. Plusieurs méthodes sont déjà implémentées. Vous devez compléter les méthodes suivantes :

- `public Node fusion(Node autre)` : fusionne les deux arbres donnés. Vous devez vérifier que les arbres sont de même ordre et que les nœuds sont bien des racines (parent == null). Faites en sorte de respecter la condition d'ordre du monceau (valeur du parent < valeur des enfants) !
- `public Node findValue(int valeur)` : renvoie le premier nœud contenant la valeur demandée, s'il existe, parmi les descendants (enfants, petits-enfants, etc.) de ce nœud, null sinon. Pensez à vérifier la valeur des enfants pour ne pas parcourir tout l'arbre inutilement, i.e. si on recherche 8 et qu'un enfant a pour valeur 12, on sait que ce n'est pas la peine d'aller explorer vers cet enfant.
- `private void moveUp()` : dans l'arbre, échange la position du nœud et de son parent. Attention, ceci brise la condition d'ordre du monceau : il ne faut utiliser cette fonction que dans `delete`. Pensez aussi à modifier la valeur d'ordre de tous les nœuds dont vous avez changé la position.

- `public ArrayList<Node> delete()` : supprime le nœud de l'arbre. Comme un arbre binaire ne peut contenir que 2^k éléments, l'opération de suppression consiste en fait à faire remonter le nœud jusqu'à ce qu'il soit racine de l'arbre (à l'aide de `moveUp`), et à renvoyer ses k-1 enfants, chacun étant un arbre binomial.

Les graphes suivants montrent le processus complet pour supprimer l'élément 12 d'un arbre binomial d'ordre 3.



Dans ce cas, node12.delete() renverrait donc un tableau contenant node4, node5 et node3.

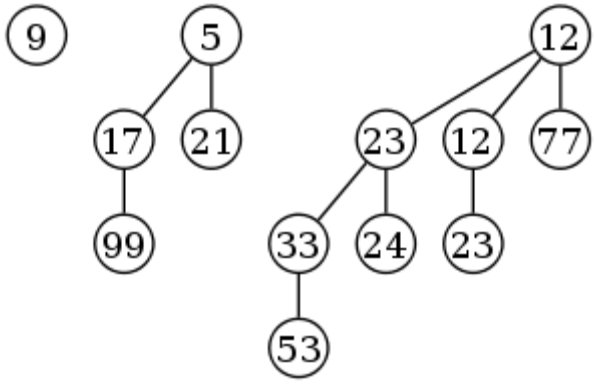
Vous devez également implémenter une méthode `print` permettant d'afficher les valeurs d'un arbre de manière lisible. Il faut que l'on puisse voir clairement la hiérarchie des nœuds, par exemple avec des tabulations. Vous pouvez vous inspirer de ce que vous avez fait au TP4 ou expérimenter autre chose. Un affichage correct serait par exemple, pour l'arbre vu précédemment (avant suppression) :

```
3     4

      5     9

      7     8

            12    13
```

Pour **cette méthode uniquement**, vous pouvez si nécessaire modifier le prototype, i.e. ajouter des paramètres ou un type de retour.

Exercice 2 : Création du monceau binomial

Un monceau binomial est un ensemble d'arbres binomiaux. Cet ensemble peut contenir exactement 0 ou 1 arbre binomial de chaque ordre : il ne peut pas y avoir 2 arbres d'ordre 3 par exemple. Comme le nombre d'éléments contenu dans un arbre ne peut être qu'une puissance de deux, les arbres présents dans le monceau dépendent de l'écriture en binaire du nombre d'éléments. Par exemple, si un monceau a 13 éléments, 13 en binaire s'écrit 1101. Il y aura donc un arbre d'ordre 3 (contenant 8 éléments), d'ordre 2 (contenant 4 éléments) et d'ordre 0 (contenant 1 élément).



Exemple de monceau binomial contenant 13 éléments (source : Wikipedia)

L'opération de base est, comme pour l'arbre binomial, la fusion de deux monceaux binomiaux. Cette fusion se fait de la façon suivante :

On a la possibilité de garder un arbre temporaire, la « retenue ».

Pour tout j, en commençant par j = 0, on regarde le nombre d'arbres d'ordre j dans les deux monceaux plus la retenue. Il peut donc y en avoir entre 0 et 3.

- S'il n'y en a pas, le monceau final n'aura pas d'arbre d'ordre j.
- S'il y en a un, on place cet arbre dans le monceau final.

- S'il y en a deux, on les fusionne et place l'arbre résultat, d'ordre j+1, dans la retenue, pour l'itération suivante. Le monceau final n'aura pas d'arbre d'ordre j.
- S'il y en a trois, on en fusionne deux, on place le troisième dans le monceau final, et place le résultat de la fusion, d'ordre j+1, dans la retenue.

Cela semble compliqué, mais il s'agit en fait exactement du même processus que lorsque l'on fait une addition binaire avec retenue.

Implémentez cette opération en complétant la méthode `public void fusion(Monceau autre)` dans la classe Monceau.

Implémentez ensuite la méthode `public void insert(int Val)` permettant d'insérer une valeur dans le monceau. Cette méthode doit utiliser `fusion`.

À l'aide des fonctions `Node.findValue` et `Node.delete`, implémentez la méthode `public boolean delete(int val)` permettant de supprimer tous les nœuds ayant une certaine valeur. Attention, `Node.findValue` ne renvoie qu'un seul nœud, il faut donc potentiellement l'appeler plusieurs fois.

Enfin, à l'aide de `Node.print`, implémentez une méthode permettant d'afficher le monceau de manière lisible. L'ordre de chaque arbre doit apparaitre.

Instructions pour la remise

Le travail doit être fait par équipe de 2 personnes idéalement et doit être remis via Moodle :

Groupe Mardi (B2) : 21 Novembre 2016 à 23h55
Groupe Mardi (B1) : 28 Novembre 2016 à 23h55
Groupe Mercredi (B1) : 29 Novembre 2016 à 23h55

Veuillez envoyer vos fichiers .java seulement dans une archive de type *.zip qui portera le nom inf2010_lab5_MatriculeX_MatriculeY (de sorte que MatriculeX < MatriculeY). Les travaux en retard seront pénalisés de 20 % par jour de retard. Aucun travail ne sera accepté après 4 jours de retard.