TD 12 – Tas Binaires

Structures de données (IF 122)

1 Définitions

Une **structure de tas** est un arbre binaire *presque* complet : tous les niveaux sont entièrement remplis à l'exception, peut-être, du dernier niveau, et ce dernier niveau est rempli “à gauche”.

Pour implémenter cette structure, on utilise un tableau. Un nœud est alors représenté par un indice dans le tableau (la racine est l'indice 0) avec la convention suivante :

– le nœud d'indice i a pour fils gauche (éventuel) le nœud d'indice $(2 \times i) + 1$.

– la nœud d'indice i a pour fils droit (éventuel) le nœud d'indice $(2 \times i) + 2$.

Le nombre d'éléments du tableau est fixé indépendamment du tas. On utilise donc un entier supplémentaire pour mémoriser le nombre réel d'éléments stockés. Cela donne la structure suivante :

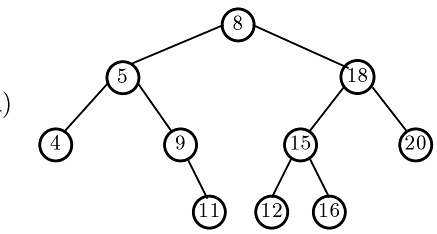
```
import fr.jussieu.script.Deug;
class Tas {

 int [] tab;
 int nbReel;

 }
```

La notion de structure de tas est indépendante de la **propriété de tas** qui est que chaque nœud de l'arbre a une étiquette plus grande que celle de ses fils.

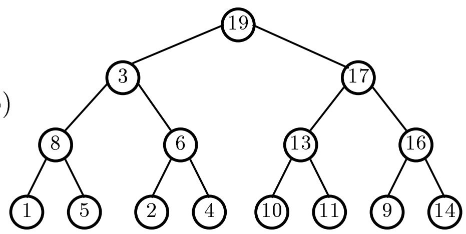
Exercice 1 (TD) — ***Questions préliminaires*** Pour chacune des figures suivantes, dites si :

1. elle représente un arbre binaire de recherche.
2. elle représente une structure de tas. Si oui, représentez la sous la forme d'un tableau.
3. elle vérifie la propriété de tas.

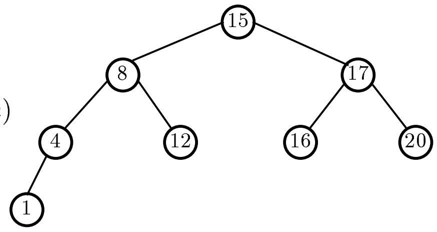
a)



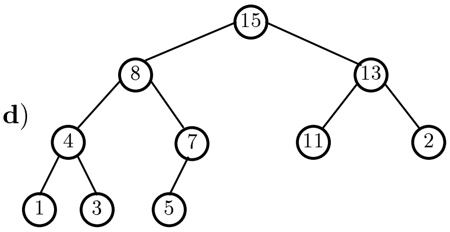
b)



c)



d)



Est-il possible qu'un arbre binaire de recherche vérifie la propriété de tas (on suppose que tous les nœuds sont deux à deux distincts) ?

2 Manipulations des tas

Exercice 2 (TD) — *feuille* Quels sont les indices des feuilles d'un tas ayant n éléments stockés ? Écrivez une méthode **estFeuille(int pos)** testant si le nœud d'indice *pos* est une feuille.

Exercice 3 (TD - TP) — *père et fils* Écrivez une méthode **filsGauche(int pos)** qui renvoie l'indice représentant le fils gauche du nœud *pos* ou -1 si celui-ci n'existe pas.
Sur le même modèle, écrivez les méthodes **filsDroit(int pos)** et **pere(int pos)**.

Exercice 4 (TD) — *hauteur* Rappelez la définition de la hauteur d'un arbre binaire quelconque. Quelle est la plus longue branche dans une structure de tas ?
Écrivez une méthode **hauteur()** calculant la hauteur d'un tas.

Exercice 5 (TP) — *propriété de tas* Écrivez une méthode **verifTas()** testant si un tas vérifie la propriété de tas.

3 Conservation de la propriété de tas

Dans les exercices suivants, on suppose que les méthodes sont appelées sur des tas vérifiant la propriété de tas et on cherche à ce que cette propriété soit préservée.

Exercice 6 (TD) — *maximum et minimum* Écrivez une méthode **max()** calculant l'élément maximum dans un tas. Comparez cette méthode à celle utilisée pour calculer l'élément maximum d'un arbre binaire de recherche.
Où peut se trouver l'élément minimun ?

Exercice 7 (TD-TP) — *insertion* Pour insérer un nouvel élément dans un tas, il suffit d'incrémenter `nbReel`, de placer l'élément en fin de tas, puis de le faire "remonter" dans le tas jusqu'à une position satisfaisante.

Écrivez une méthode **inserer(int e)** permettant l'insertion de l'élément d'étiquette e dans un tas. Simulez l'insertion de l'élément d'étiquette 12 dans le tas **d**)

Exercice 8 (TD-TP) — *entassement* Si un tas ne vérifie pas la propriété de tas à cause d'un seul élément mal placé, il est possible de corriger cette erreur en faisant "descendre" l'élément le long de l'arbre jusqu'à trouver sa place, c'est à dire lorsque sa valeur est supérieure à celle de ses fils.

Pour cela, il suffit de le comparer avec ses deux fils : s'il est plus grand, on ne change rien, sinon on l'échange avec le plus grand de ses deux fils et on relance la procédure à partir de sa nouvelle position.

Simulez cette opération sur le nœud d'indice 3 du tas **b**).
Écrivez une méthode **entasser(int pos)** qui réalise l'opération décrite ci-dessus.

Exercice 9 (TD) — *conversion* En déduire un algorithme permettant de convertir un tas en un tas vérifiant la propriété de tas. Écrivez la méthode **convertirTas()** implémentant cet algorithme.

Exercice 10 (TD) — *suppression* Pour supprimer un élément dans un tas, on propose la méthode suivante : on échange cet élément avec le dernier élément a du tas, on décrémente le nombre d'éléments stockés, et on lance la procédure d'entassement sur l'élément a qui vient d'être déplacé (c'est en effet le seul qui brise la propriété de tas). Notez qu'une fois l'élément supprimé, il est toujours présent dans le tableau (mais pas dans le tas), à la position *nbReel*.

Simulez cet algorithme sur le nœud d'indice 2 du tas **d**). Écrivez la méthode **supprimer(int pos)** correspondante.

4 Tri par tas

Exercice 11 (TD) — *Tri d'un tableau* Le tri par tas est une procédure permettant de trier un tableau d'entiers en travaillant "en place".

L'idée est la suivante : on commence par convertir le tableau en un tas vérifiant la propriété de tas (cf exercice 9). Puis on supprime la racine a en utilisant la méthode de l'exercice 10 : l'élément a se retrouve alors au bout du tableau, et c'est sa "bonne place", puisque la racine est toujours le plus grand élément du tas. Considérant le tas obtenu après suppression, on supprime à nouveau sa racine, etc, jusqu'à avoir vidé entièrement le tas : le tableau est alors trié !

Utilisez cet algorithme pour trier "manuellement" le tableau suivant : `[5 ;14 ;4 ;12 ;3 ;9]`

Implémentez la méthode statique **triTas(int [] t)** correspondant à cette méthode.