# TD n°8

## Les Tas

## 1 Gestion d'une file de priorité

On suppose que des gens se présentent au guichet d'une banque avec un numéro écrit sur un bout de papier représentant leur degré de priorité. Plus ce nombre est élevé, plus ils sont importants et doivent passer rapidement. Bien sûr, il n'y a qu'un seul guichet ouvert, et l'employé(e) de la banque doit traiter rapidement tous ses clients pour que tout le monde garde le sourire. La file des personnes en attente s'appelle *une file de priorité*. L'employé de banque doit donc savoir faire rapidement les 3 opérations suivantes : trouver un maximum dans la file de priorité, retirer cet élément de la file, ajouter un nouvel élément à la file. Plusieurs solutions sont envisageables.

**Exercice 1** La première consiste à mettre la file dans un tableau et à trier la file de priorité dans l'ordre croissant des priorités. Rappeler la complexité au pire cas d'une telle opération. Combien de temps la recherche d'un maximum peut-elle prendre ? l'insertion d'un nouvel élément ?

**Exercice 2** Une autre méthode consiste à gérer la file comme une simple file, et à rechercher le maximum à chaque fois. Combien de temps la recherche d'un maximum peut-elle prendre ? l'insertion d'un nouvel élémént ?

Une méthode élégante consiste à gérer la file de priorité grâce aux tas. On rappelle qu'un tas est un arbre binaire où chaque noeud possède une clé, telle que la clé d'un noeud soit toujours supérieure à celle de ses fils. De plus, le tas doit être *complet calé à gauche* : toutes les lignes sont complètement remplies sauf la dernière qui est remplie en partant de la gauche jusqu'à un certain point.

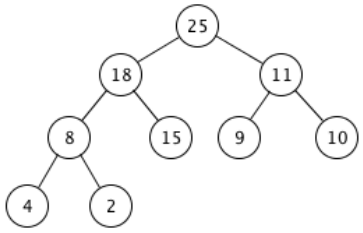


FIGURE 1 – Un tas

**Exercice 3** Quelle est la hauteur d'un arbre complet calé à gauche (ACCG) à $n$ noeuds ?

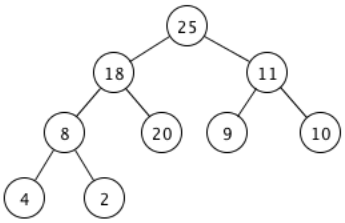


FIGURE 2 – N'est pas un tas

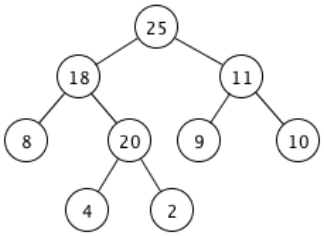


FIGURE 3 – N'est pas complet calé à gauche

**Exercice 4** Donner une procédure simple de suppression d'un élément d'un ACCG, tout en conservant sa structure.

On peut numéroter les noeuds d'un tas selon un parcours en largeur : on lit les étiquettes des noeuds de l'arbre de haut en bas et de gauche à droite. Ceci permet d'implémenter les tas sous la forme d'un tableau : le sommet dont le rang est $i$ est rangé à l'indice $i$ du tableau de représentation.

**Exercice 5** Donner les indices des éléments suivants :
– racine
– père du sommet $i$
– fils gauche de $i$
– fils droit de $i$

**Exercice 6** Ecrire une classe `Tas`, ayant un attribut `taille` qui compte le nombre de noeuds du tas, un attribut `static final MAX` (qui correspond à la taille maximale du tas) et un attribut privé `tab` (tableau représentatif du tas, de taille `MAX`). Écrire un constructeur et les fonctions suivantes :

```
int pere(int i)
int filsgauche(int i)
int filsdroit(int i)
```

qui retournent les indices des noeuds correspondants, ainsi que :

```
boolean estUneFeuille(int i)
```

Une nouvelle personne arrive au guichet avec son numéro de priorité.

**Exercice 7** Sachant que le guichetier a organisé ses clients en tas, proposer un algorithme lui permettant d'insérer la personne à sa bonne place, tout en conservant la structure de tas de ses clients. Dérouler cet algorithme sur l'exemple de la figure 1 (le numéro du nouveau client est le 12). Ecrire ensuite une méthode qui implémente cet algorithme. Quelle est la complexité de cet algorithme ?

Une fois les clients bien rangés selon leur priorité, le guichetier va commencer à traiter la personne la plus prioritaire, et une fois terminé, va l'enlever de ses clients de manière à garder la structure de tas.

**Exercice 8** Proposer un algorithme qui permet de supprimer le max du tas, tout en conservant sa structure. Dérouler cet algorithme sur le nouveau tas obtenu à la question précédente. Ecrire ensuite une méthode qui implémente cet algorithme. Quelle est la complexité de cet algorithme ?

**Exercice 9** Comparer la complexité des deux opérations précédentes à celles trouvées dans les deux premières questions.