TD6 : Tas binaire

1 Rappels de cours

Un tas binaire est une structure de données permettant de modéliser une file à priorités. Elle permet de réaliser les opérations suivantes :

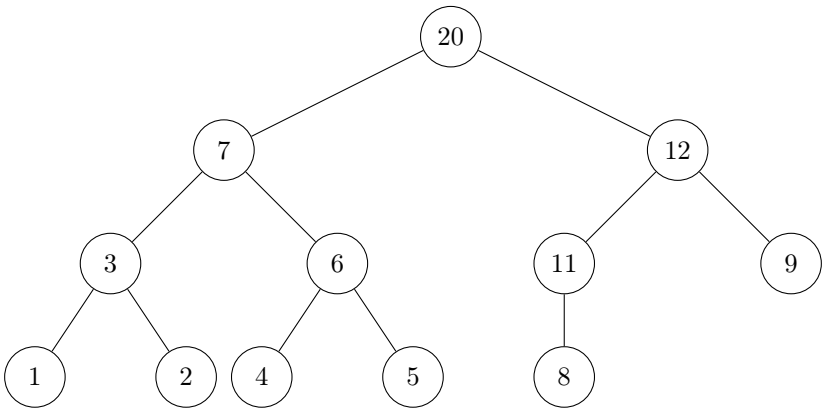
- insérer un nouvel élément associé à une priorité en $O(\log n)$;
- accéder à l'élément de priorité maximale en $O(1)$;
- supprimer l'élément de priorité maximale en $O(\log n)$;
- modifier la priorité d'un élément (si vous pouvez le retrouver dans le tableau) en $O(\log n)$.

Pour simplifier la suite, nous ne noterons que la priorité des éléments, et omettrons leur valeur.

Cette structure de données maintient un ordre partiel sur les données via un arbre binaire. Cet arbre respecte les propriétés suivantes :

- chaque nœud a une priorité supérieure à celles de ses enfants ;
- un nouveau niveau n'est créé que si le niveau précédent est plein ;
- les niveaux de l'arbre sont remplis de gauche à droite.

Par exemple :



Cette structure de données peut être implémentée efficacement dans un tableau : la racine est la première case du tableau, les deux cases suivantes contiennent le premier niveau, les quatre suivantes le second niveau, et ainsi de suite. L'arbre précédent est ainsi encodé dans le tableau :

20	7	12	3	6	11	9	1	2	4	5	8

Pour circuler dans l'arbre, étant donné un indice i, nous avons :

- $parent(i) = \lfloor \frac{i-1}{2} \rfloor$ où $\lfloor . \rfloor$ représente l'arrondi entier inférieur ;
- $gauche(i) = 2i + 1$ pour obtenir l'enfant gauche ;
- $droite(i) = 2i + 2$ pour obtenir l'enfant droit.

La racine de l'arbre a par construction la plus forte priorité, et est stockée dans la première case du tableau. Il est donc possible d'y accéder en temps constant. Pour les autres opérations, l'arbre étant équilibré, toute action qui nécessite de le parcourir de haut en bas ou de base en haut aura donc au pire une complexité en $O(\log n)$, la hauteur de l'arbre en fonction du nombre d'éléments.

Insertion : pour préserver l'agencement de l'arbre, l'élément est ajouté à la fin du tableau, et remonté ensuite vers la racine selon sa priorité. Pour déterminer si un nœud doit remonter, sa priorité est comparée avec celle de son parent.

Procédure InsertionTas(t, e)
- **données :** t : un tableau contenant un tas, e un élément
- **Algorithme**
  - $i \leftarrow$ taille de t
  - **tant que** $i > 0$ et $e > t\lfloor \frac{i-1}{2} \rfloor$ **faire**
    - $t[i] \leftarrow t\lfloor \frac{i-1}{2} \rfloor$
    - $i \leftarrow \lfloor \frac{i-1}{2} \rfloor$
  - $t[i] \leftarrow e$

Suppression : pour préserver l'agencement de l'arbre, l'élément en fin de tableau est déplacé à la racine. Il est ensuite descendu dans l'arbre jusqu'à retrouver une place qui convienne à sa priorité.

Procédure SuppressionTas(t)
- **données :** t : un tableau contenant un tas
- **Algorithme**
  - $s \leftarrow$ taille de $t - 1$
  - **si** s $= 0$ **alors**
    - réduire t d'un élément
    - **retourner**
  - $e \leftarrow t[s]$
  - réduire t d'un élément
  - $i \leftarrow 0$
  - **tant que** $(2i + 1 < s$ et $e < t[2i + 1])$ ou $(2i + 2 < s$ et $e < t[2i + 2])$ **faire**
    - **si** $2i + 2 < s$ et $t[2i + 1] < t[2i + 2]$ **alors**
      - $t[i] \leftarrow t[2i + 2]$
      - $i \leftarrow 2i + 2$
    - **sinon**
      - $t[i] \leftarrow t[2i + 1]$
      - $i \leftarrow 2i + 1$
  - $t[i] \leftarrow e$

2 Mise en œuvre

a) Insérez successivement les valeurs 7, 1, 9, 35, 4, 10, 13, 2, 20, 40 dans un tas, et indiquez à chaque insertion les états par lesquels passe le tas.

b) Supprimez un à un les éléments du tas, et indiquez à chaque itération l'état du tas.

3 Tri par tas

Le tri par tas est un algorithme de tri *en place*. Il ne demande pas la création d'autres tableaux intermédiaires pour construire le résultat, et n'utilise presque pas plus de mémoire que le tableau fourni (les seules données supplémentaires sont les variables de l'algorithme et un temporaire éventuel pour les échanges). Le principe est le suivant : à partir d'un tableau sans ordre, le tableau est tout d'abord transformé en tas, puis l'élément en tête du tas est itérativement prélevé jusqu'à ce que le tas soit vide.

c) Déterminez comment ce tri peut-être fait *en place* sans créer de tableau supplémentaire.

d) Quelle est la complexité de cet algorithme ?