

Objectifs

- À la fin de cette séance, vous devriez être capable de :
- implémenter une structure de tas dans un tableau ;
 - manipuler cette structure ;
 - maîtriser le tri par tas.

Nous avons vu en cours la structure de *tas* (arbre tassé ordonné), qui permet d’implanter efficacement une file à priorité de taille n sous forme d’un arbre de hauteur $\lceil \log_2 n \rceil$.

Nous avons pour l’instant décrit les opérations sur cette structure sous la forme de fonctions sur des arbres. L’objet de ce TD est de montrer comment un arbre binaire tassé peut être représenté sous forme de tableau, et de traduire les opérations du tas dans cette représentation.

Attention : dans le cours nous avons considéré un « tas min », dans lequel la clé de chaque nœud doit être inférieure à celles de ses fils.
 Dans ce TD, pour que le tri par tas fonctionne, il faudra considérer un « tas max », dans lequel la clé de chaque nœud doit être **supérieure** à celles de ses fils.
 Les opérations étudiées en cours s’adaptent facilement pour tenir compte de cet ordre sur les nœuds.

Implantation du tas dans un tableau

Soit un arbre binaire tassé de n nœuds et de hauteur h .

On place les étiquettes des nœuds dans un tableau (de taille au moins n), dans l’ordre du parcours par niveaux (du niveau 0 au niveau h et de gauche à droite dans chaque niveau). Comme la forme de l’arbre est complètement déterminée par n , on a ainsi une correspondance univoque entre les tas et les tableaux.

Par ailleurs, pour pouvoir insérer de nouveaux éléments, on prévoit un tableau d’une taille suffisante $nmax$, et on maintient donc :

- F : un tableau sur $[0 \dots nmax - 1]$ d’entiers qui sont leur propre clé.
- n : un entier de $[0 \dots nmax]$, qui donne le nombre d’éléments dans le tas

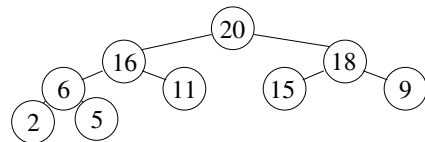
Exemple :

Le tas ci-contre est représenté par le tableau :

20	16	18	6	11	15	9	2	5	...
----	----	----	---	----	----	---	---	---	-----

avec $n = 9$.

(On remarque que la racine du tas contient l’élément maximal.)



Exercice 1 : Indices et hauteurs

Les indices commencent à 0.

1. Quel est l’indice de la racine ?
2. Si un nœud interne est situé à l’indice i , à quel indice est situé son fils gauche ? son fils droit ? son père ?
3. Quel est l’indice maximal d’un nœud interne ? À quelle condition un nœud d’indice i possède-t-il un fils gauche ? un fils droit ?
4. Quel est le niveau d’un nœud d’indice i ?
5. Quelle est la hauteur maximale d’un sous-arbre dont la racine est à l’indice i ?

Correction de l’exercice 1

1. 0
2. Le fils gauche est à $2i + 1$, le fils droit à $2i + 2$, et le père à $\lfloor (i - 1)/2 \rfloor$.

3. L'indice du dernier nœud interne est $\lfloor n/2 \rfloor - 1$.
Il y a un fils gauche car $2i + 1 \leq n - 1$, un fils droit si $2i + 2 \leq n - 1$.
4. $\lfloor \log_2(i + 1) \rfloor$
5. $\lfloor \log_2 n \rfloor - \lfloor \log_2(i + 1) \rfloor$

Exercice 2 : Opérations

1. Reprenez (réécrivez) les opérations **Insérer** et **Extraire_max** telles que vous les avez vues en cours
2. Traduisez chaque mot, chaque ligne ... en fonction de F et de n .

Correction de l'exercice 2

1. Insérer

```

noeud := nouvelle feuille "en bas à droite"
etiquette(noeud) := nouvelle étiquette
tant que noeud n'est pas la racine
    et puis étiquette(noeud) > étiquette(père(noeud)) faire
        échanger_étiquettes(noeud, père(noeud))
    noeud := père(noeud)
  
```

Extraire_max

```

max := étiquette de la racine
échanger_étiquettes(dernière feuille, racine)
supprimer(dernière feuille)
noeud := racine
tant que noeud a des fils
    et puis étiquette(noeud) < étiquette(plus_grand_fils(noeud)) faire
        pgf := plus_grand_fils(noeud)
        échanger_étiquettes(noeud, pgf)
    noeud := pgf
renvoyer max
  
```

2. Insérer(e) :

```

n := n + 1
F[n] := e
noeud := n
pere := (noeud-1) / 2
tant que noeud <> 0 et puis F[noeud] > F[pere] faire
    Échanger F[noeud] et F[pere]
    noeud := pere
    pere := (noeud-1) / 2
  
```

Extraire_max() :

```

e := F[0]
F[0] := F[n-1]
n := n - 1
noeud := 0
tant que noeud <= n/2 - 1
    et puis F[noeud] < F[plus_grand_fils(noeud)] faire
        pgf := plus_grand_fils(noeud)
        Échanger F[pgf] et F[noeud]
    noeud := pgf
  
```

renvoyer e

avec :

plus_grand_fils(noeud) :

si $n-1 = 2 * \text{noeud} + 1$ alors renvoyer $2 * \text{noeud} + 1$ // Un seul fils
 sinon si $F[2 * \text{noeud} + 1] > F[2 * \text{noeud} + 2]$ alors renvoyer $2 * \text{noeud} + 1$
 sinon renvoyer $2 * \text{noeud} + 2$

Tri par tas

On veut trier un tableau T de N entiers (rangés dans les cases d'indices $[0 \dots N - 1]$). On applique le principe suivant :

1. Dans un premier temps on modifie T de sorte qu'il devienne un tas de N éléments.
2. Dans un deuxième temps, on modifie à nouveau T de sorte que ses éléments y soient rangés en ordre croissant.

Exercice 3 : Version “naïve”

Au cours de la première étape, un indice k parcourt les valeurs de 0 à $N - 1$ et on maintient l'invariant suivant :

- $T[0 \dots k]$ est une permutation de sa valeur initiale ;
- $T[k + 1 \dots N - 1]$ est inchangé ;
- $T[0 \dots k]$ est un tas.

1. Dessinez cet invariant.
2. Écrivez l'algorithme de création du tas en utilisant une procédure similaire à **Insérer**.

Au cours de la deuxième étape, l'indice k parcourt les valeurs de $N - 1$ à 0 et on maintient l'invariant suivant :

- T est une permutation de sa valeur initiale ;
- $T[0 \dots k]$ est un tas.
- $T[k + 1 \dots N - 1]$ comporte les $N - k$ plus grandes valeurs du T initial, en ordre croissant.

3. Dessinez cet invariant.
4. Écrivez l'algorithme de cette étape en utilisant une procédure similaire à **Extraire_max**.
5. Évaluez la complexité de chacune des deux étapes et en déduire celle du tri par tas.

Correction de l'exercice 3

- 1.
2. pour k de 0 à $N-1$ faire
 - noeud := $k+1$
 - pere := $(\text{noeud}-1) / 2$
 - tant que noeud $<> 0$ et puis $T[\text{noeud}] > T[\text{pere}]$ faire
 - $T[\text{noeud}] \leftrightarrow T[\text{pere}]$
 - noeud := pere
 - pere := $(\text{noeud}-1) / 2$
- 3.

4. pour k de N-1 à 0 faire
 - T[k] ↔ T[0]
 - k := k - 1
 - noeud := 0
 - tant que noeud ≤ (k-1)/2 - 1
 - et puis T[noeud] < T[plus_grand_fils(noeud)] faire
 - pgf := plus_grand_fils(noeud)
 - T[pgf] ↔ T[noeud]
 - noeud := pgf
5. Chacune de ces deux étapes est en $n \log_2 n$, donc le tri par tas également.

Exercice 4 : Optimisation de la création du tas initial

Le tableau initial T représente un arbre binaire tassé. Les feuilles de cet arbre sont des arbres tassés ordonnés. L'idée est de donner progressivement la propriété « ordonné » à T, en procédant par niveaux, des feuilles vers la racine : le traitement d'un niveau consiste à faire percoler vers le bas (si nécessaire) les racines des sous-arbres de ce niveau.

1. Évaluez le coût de cette nouvelle procédure et montrer le gain par rapport à la version précédente.
2. Écrivez cette nouvelle procédure.

Correction de l'exercice 4

1. Depuis le niveau i , le coût de la percolation vers le bas est au maximum $h - i$ (h est la hauteur de l'arbre), rappelons que $h = \lfloor \log_2(n) \rfloor$. Au niveau i , il y a 2^i noeuds traités, sauf pour le niveau le plus bas (celui dont on part) où il y en a moins. Le coût de la construction du tas est donc majoré par C avec

$$C = \sum_{i=0}^{h-1} 2^i \times (h - i)$$

soit $C = h + 2(h - 1) + 4(h - 2) + \dots + 2^{h-1}$. Donc $2C = 2h + 4(h - 1) + \dots + 2^{h-1} \times 2 + 2^h$. Par différence on obtient : $C = -h + 2 + 4 + \dots + 2^h = -h + 2 \times (2^h - 1)$. Comme $h = \lfloor \log_2(n) \rfloor$, C est de l'ordre de n .

2. pour k de n/2 - 1 à 0 faire
 - noeud := k
 - tant que noeud ≤ n/2 - 1
 - et puis T[noeud] < T[plus_grand_fils(noeud)] faire
 - pgf := plus_grand_fils(noeud)
 - T[pgf] ↔ T[noeud]
 - noeud := pgf