UFR IM²AG
UNIVERSITÉ Grenoble Alpes

Tri par tas (heapsort)

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 $\lfloor \log_2 n \rfloor$.
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.

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 :

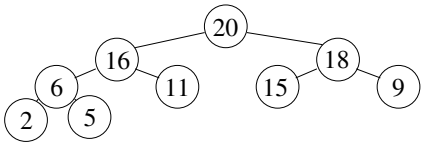
— F : un tableau sur $[1 \dots nmax]$ d'entiers qui sont leur propre clé.

— n : un entier de $[0 \dots nmax]$

Exemple 1
Le tas ci-contre est représenté par le tableau :

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

avec $n = 9$



Indices et hauteurs (∼ 10 minutes)

Les indices commencent à 1.

- Quel est l'indice de la racine ?
  1
- Si un nœud interne est situé à l'indice i, à quel indice est situé son fils gauche ? son fils droit ? son père ?
  Le fils gauche est à $2i$, le fils droit à $2i + 1$, et le père à $\lfloor i/2 \rfloor$.
- Quel est l'indice maximal d'un nœud interne ? À quelle condition un nœud d'incide i possède-t-il un fils gauche ? un fils droit ?
  L'indice du dernier nœud interne est $\lfloor n/2 \rfloor$.
  Il y a un fils gauche si $2i \leq n$, un fils droit si $2i + 1 \leq n$.
- Quel est le niveau d'un nœud d'indice i ?
  $\log_2 i$
- Quelle est la hauteur maximale d'un sous-arbre dont la racine est à l'indice i ?
  $\log_2 n - \log_2 i$

Opérations (∼ 35 minutes)

1. Reprenez (réécrivez) les opérations **Insérer** et **Extraire_max** telles que vous les avez vues en cours

   Insérer
   noeud <- nouvelle feuille “en bas à droite”

etiquette(noeud) <- nouvelle étiquette
tant que noeud n'est pas la racine
        etpuis étiquette(noeud) > étiquette(père(noeud)) faire
    échanger_étiquettes(noeud, père(noeud))
    noeud <- père(noeud)

Extraire_max
échanger_étiquettes(dernière feuille, racine)
noeud <- racine
tant que noeud a des fils
        etpuis étiquette(noeud) < étiquette(plus_grand_fils(noeud)) faire
    pgf <- plus_grand_fils(noeud)
    échanger_étiquettes(noeud, pgf)
    noeud <- pgf
supprimer(dernière feuille) et renvoyer sa racine

2. Traduisez chaque mot, chaque ligne ... en fonction de F et de n.
Insérer(e) :
    n <- n + 1
    F[n] <- e
    noeud <- n
    tant que noeud <> 1 etpuis F[noeud] > F[noeud/2] faire
        F[noeud] <-> F[noeud/2]
        noeud <- noeud/2

Extraire_max() :
    e <- F[1]
    F[1] <- F[n]
    n <- n-1
    noeud <- 1
    tant que noeud <= n/2
            etpuis F[noeud] < F[plus_grand_fils(noeud)] faire
        pgf <- plus_grand_fils(noeud)
        F[pgf] <-> F[noeud]
        noeud <- pgf
    renvoyer e

avec :

plus_grand_fils(noeud) :
    si n = 2*noeud alors renvoyer n   // Un seul fils
    sinon si F[2*noeud] > F[2*noeud + 1] alors renvoyer 2*noeud
        sinon renvoyer 2*noeud + 1

Tri par tas

On veut trier un tableau T de N entiers (rangés dans les cases d'indices $[1 \dots N]$). 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.*

Version « naïve » (∼ 30 minutes)

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

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

3. Dessinez cet invariant.

4. Écrivez l'algorithme de création du tas en utilisant une procédure similaire à **Insérer**.

```
k <- 0
pour i de 1 à n faire
    k <- k + 1
    noeud <- k
    tant que noeud <> 1 etpuis T[noeud] > T[noeud/2] faire
        T[noeud] <-> T[noeud/2]
        noeud <- noeud/2
```

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

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

5 Dessinez cet invariant.

6 Écrivez l'algorithme de cett étape en utilisant une procédure similaire à **Extraire_max**.

```
k <- N
pour i de n à 1 faire
    T[i] <-> T[1]
    k <- k -1
    noeud <- 1
    tant que noeud < k/2
            etpuis T[noeud] < T[plus_grand_fils(noeud)] faire
        pgf <- plus_grand_fils(noeud)
        T[pgf] <-> T[noeud]
        noeud <- pgf
```

7 Évaluez la complexité de chacune des deux étapes et en déduire celle du tri par tas.
Chacune de ces deux étapes est en $n \log_2 n$, donc le tri par tas également.

Optimisation de la création du tas initial (∼ 15 minutes)

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.

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

   Lorsque k est au niveau i, le coût de la descente 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 $C = h + 2(h-1) + 4(h-2) + ... + 2^{h-1}$. Donc $2C = 2h + 4(h-1) + ... + 2^{h-1}2 + 2^h$. Par différence on obtient : $C = -h + 2 + 4 + ... + 2^h = -h + 2 \times (2^h - 1)$. Comme $h = \lfloor log_2(n) \rfloor$, C est de l'ordre de n.

9. Écrivez cette nouvelle procédure.

```
pour k de n/2 à 1 faire
    noeud <- k
    tant que noeud <= n/2
            etpuis T[noeud] < T[plus_grand_fils(noeud)] faire
        pgf <- plus_grand_fils(noeud)
        T[pgf] <-> T[noeud]
        noeud <- pgf
```