UFR IMA

Informatique & Mathématiques Appliquées

Grenoble 1

UNIVERSITE JOSEPH FOURIER

Sciences, Technologie, Médecine

Arbres ordonnés, binaires, tassés, FAP, tri par FAP, tas, tri par tas

1. Arbres ordonnés

1.1. Arbres ordonnés (Arbres *O*)

On considère des arbres dont les nœuds sont étiquetés sur un ensemble muni d'un ordre total.
Un arbre est *ordonné* (Figure 1), si tout nœud de l'arbre a une étiquette supérieure ou égale à celles de chacun de ses (s'ils existent). On convient qu'un arbre vide est ordonné.

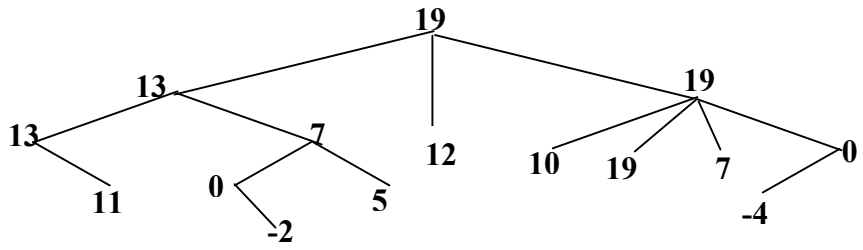

Figure 1 : Un arbre d'entiers ordonné

Propriétés

— Les sous-arbres d'un arbre *O* sont eux-mêmes des arbres *O*.
— Dans tout chemin de l'arbre, les étiquettes sont en ordre non croissant (de père en fils).
— L'étiquette de la racine d'un arbre *O* a la valeur maximum des étiquettes de l'arbre.

1.2. Arbres binaires ordonnés (Arbres *BO*)

Dans la suite, on s'intéressera aux arbres ordonnés *binaires*. Ils ont évidemment les mêmes propriétés que les arbres ordonnes.

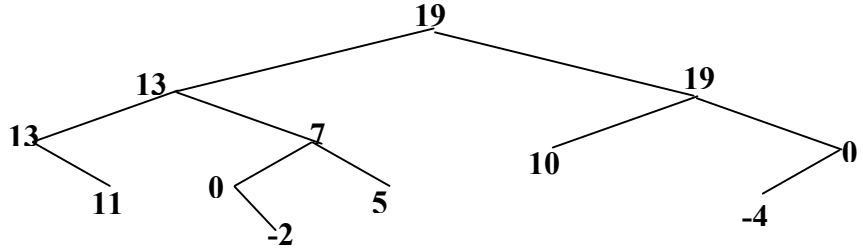

Figure 1 : Un arbre binaire d'entiers ordonné

2. Arbres binaires tassés (arbres *T*)

On rappelle que la hauteur (profondeur) d'un arbre est la longueur (nombre d'arcs) maximum de chemin dans l'arbre. Le niveau d'un nœud est la longueur de son chemin d'accès (0 pour la racine).

définition
Un arbre binaire de hauteur **h** est *tassé* (Figure 2) si les conditions suivantes sont respectées :
— Tous les nœuds internes ont deux fils, sauf éventuellement le dernier dans l'ordre par niveaux.
— Les feuilles de niveau **h** sont "tassées à gauche". Ceci équivaut à dire que les feuilles de niveau **h-1** sont "tassées à droite" ou que " les nœuds internes de niveau **h-1** sont "tassés à gauche".
Lorsque l'arbre a toutes ses feuilles au niveau **h**, l'arbre binaire est *complet*.

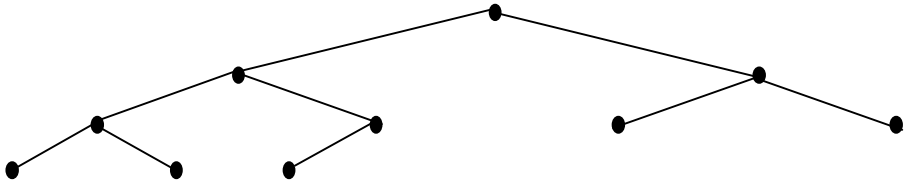

Figure 2 : Un arbre binaire tassé de hauteur 3

Propriétés
— Il y a au plus un nœud unaire (lorsque **n** est pair) et dans ce cas, ce nœud a un fils gauche.
— Le nombre de nœuds du niveau **i** est $\mathbf{2^i}$.
— Le nombre total de nœuds est $\mathbf{n = 2^h - 1}$ *{partie complète}* + **x** *{dernier niveau}* où $\mathbf{0 < x \leq 2^h}$.
Par conséquent : $\mathbf{h = \lfloor \log_2 n \rfloor}$.

3. Arbres binaires tassés ordonnés (arbres *TO*)
Un arbre binaire d'entiers est *tassé et ordonné* (Figure 3), s'il possède les deux propriétés précédentes.

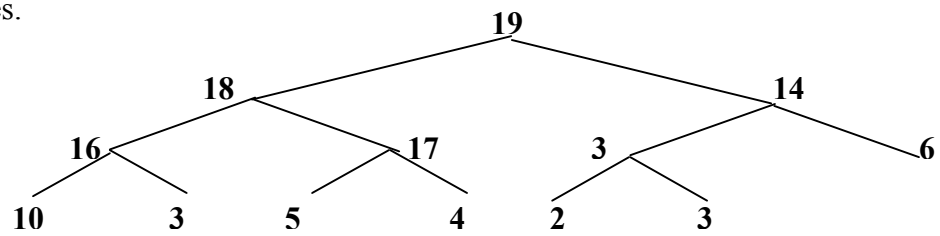

Figure 3 : Un arbre binaire d'entiers tassé et ordonné

4. Représentation d'une file d'attente à priorité par un arbre TO
On rappelle qu'une file d'attente à priorité (FAP) est une suite de couples $\mathbf{<p_i, M_i>}$ où $\mathbf{p_i}$ dénote une priorité (appartenant à un ensemble muni d'un ordre total) et $\mathbf{M_i}$ dénote l'ensemble des objets de priorité $\mathbf{p_i}$. Une FAP est manipulée avec les opérations **EstVide**, **Vider**, **Sortir**, et **Entrer**.
Une FAP peut être représentée par un arbre binaire TO dans lequel les nœuds sont étiquetés par des couples <priorité, objet> ou par des couples <priorité, ensemble d'objets ayant cette priorité>. Les nœuds sont ordonnés selon les valeurs de priorité.

4.1. Opération Sortir
L'étiquette de la racine de l'arbre a une priorité maximale. Pour réaliser l'opération **Sortir**, on supprime la racine et on reconstruit un arbre *TO* avec les nœuds restants :
— Pour maintenir la propriété *T*, on supprime la *dernière feuille* (la feuille la plus à droite du dernier niveau), et on place son étiquette à la racine. Mais l'arbre n'a plus la propriété *O*.
— Pour maintenir la propriété *O*, on échange l'étiquette de la racine avec celle du *plus grand fils* et on recommence l'opération avec le sous-arbre correspondant.
Cela revient à *descendre* l'étiquette de la dernière feuille le long du chemin des plus grands fils (Figure 4) par un algorithme d'insertion d'un élément dans une séquence triée.

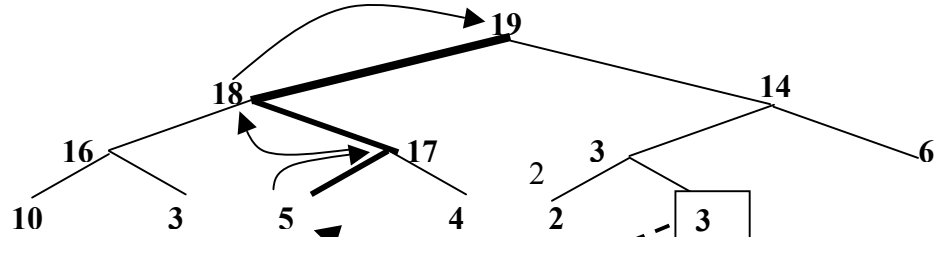

Figure 4 : Descente de l'étiquette de la dernière feuille dans le chemin des plus grands fils

4.2. Opération Entrer

Pour maintenir la propriété *T*, on ajoute une nouvelle feuille au dernier niveau, après la *dernière feuille*. Pour maintenir la propriété *O*, il faut placer la nouvelle étiquette au bon endroit dans le chemin d'accès de la racine à la nouvelle feuille : on *remonte* cette étiquette dans ce chemin par un algorithme d'insertion dans une séquence triée (Figure 5).

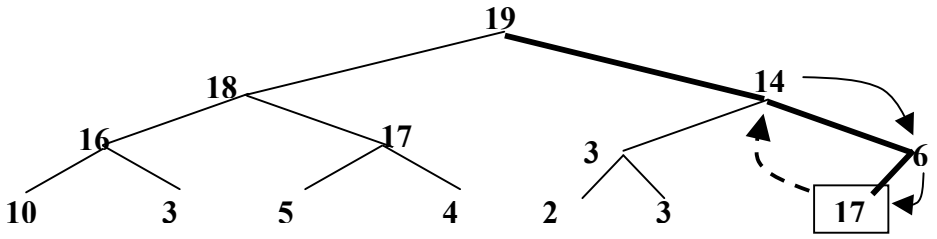

Figure 5 : Remontée de la valeur 17 dans le chemin de la racine à la nouvelle feuille

4.3. Complexité des opérations.

Le coût des opérations **Sortir** et **Entrer** est majoré par la hauteur de l'arbre. La complexité en nombre de comparaisons est donc d'ordre $\mathbf{log_2 n}$, **n** étant le nombre de nœuds de l'arbre.

4.4. Tri par FAP

Soit un tableau **T** de **n** entiers à trier. Le principe du tri par FAP est le suivant : à partir de **T** construire (opération **Entrer**) un arbre binaire *TO* étiquetés par les éléments de **T** ; chaque élément de **T** joue le double rôle d'objet et de priorité. Dans un deuxième temps, construire le résultat dans **T** (opération **Sortir**). L'esquisse de l'algorithme est la suivante :

Vider(F, n)
i parcourant [1…n] : Entrer(F, T_i, T_i)
i parcourant [1…n] en sens inverse : Sortir(F, T_i, T_i)

Le coût est majoré par $\mathbf{2*n*log_2 n}$. Le tri par FAP a ainsi une complexité en nombre de comparaisons d'ordre $\mathbf{n*log_2 n}$.

5. Structure de tas

5.1. Représentation contiguë d'un arbre *T* dans un tableau

Soit un arbre tassé de **n** nœuds et de hauteur **h**. Les étiquettes des nœuds sont placées dans le tableau, dans l'ordre par niveaux (du niveau 0 au niveau **h** et de gauche à droite dans chaque niveau). Tout nœud est identifié par son rang dans l'ordre par niveaux :

— racine = 1, gauche(i) = 2*i ; droit(i) = 2*i+1 ; père(i) = i div 2 ; ExisteGauche(i) = $2i \leq n$; ExisteDroit(i): $2i+1 \leq n$.

— La valeur maximum des rangs de nœuds internes est **n div 2**.

— Le niveau du nœud de rang **i** est $\lfloor \mathbf{log_2 i} \rfloor$.

— la hauteur du sous-arbre de racine **i** est bornée par $\mathbf{h-\lfloor log_2 i \rfloor = \lfloor log_2 n/i \rfloor}$.

5.2. Définition

Un *tas* est un arbre binaire *TO* représenté de manière contiguë dans un tableau. Par exemple, dans le cas de l'arbre de la Figure 5 (après entrée de 17), le tableau représente la séquence [19, 18, 17, 16, 17, 3, 14,10, 3, 5, 4, 2, 3, 6].

5.3. Représentation d'une FAP par un tas

Pour représenter une FAP de **n** éléments, on utilise un tableau de couples <priorité, objet> :

F : un tableau sur [1…max] de <prio : une priorité, obj : un objet>
n : un entier sur [0…max]

Remarque : on peut aussi utiliser deux tableaux, l'un pour les priorités, l'autre pour les objets.

a) Vider, EstVide

Vider(F, n) : n ← 0
EstVide(F, n) : n=0

b) Sortir

Pour déterminer le "plus grand fils" d'un nœud **i**, on utilise une fonction nommée **PgF** :

PgF : deux entiers sur [1…max] → un entier sur [1…max]
*{PgF(i, k) est le numéro, dans le tas $T_{1...k}$, du fils de **i** ayant la plus grande étiquette, si **i** est binaire ou celle du fils gauche. **Précondition** : **i** est un nœud interne.}*

PgF(i, k) : *{i ≤ k div 2}*
soit g = 2*i dans si g+1≤ k *{i est binaire}* et puis F_g•prio < F_{g+1}•prio alors g+1 sinon g

Sortir(F, n, x, p) :
ic : un entier sur [1…max] *{indice courant pour la descente}*
<x, p> ← F_1
n ← n – 1 ; F_1 ← F_{n+1} *{supprimer la dernière feuille }*
{descente de F_1}
ic ← 1
tant que ic ≤ n div 2 et puis F_{ic}•prio < $F_{PgF(ic, n)}$•prio
F_{ic} ↔ $F_{PgF(ic, n)}$; ic ← PgF(ic, n)
{La longueur du chemin parcouru est bornée par la profondeur de l'arbre. Le coût est donc majoré par $\lfloor log_2(n-1) \rfloor$.}

c) Entrer

Entrer(F, n, x, p) :
ic : un entier sur [1…max] *{indice courant pour la remontée}*
n ← n + 1 *{augmenter la taille du tas}*
F_n ← <x, p>
{remontée de F_n}
ic ← n
tant que ic > 1 et puis F_{ic}•prio > $F_{ic\ div\ 2}$•prio
F_{ic} ↔ $F_{ic\ div\ 2}$; ic ← ic div 2
{coût majoré par $\lfloor log_2(n) \rfloor$}

5.4. Tri par tas (Heapsort)

On veut trier un tableau **T** de **n** entiers défini sur **[1…n]** : dans un premier temps on modifie **T** de sorte qu'il devienne un tas de **n** éléments (**Entrer**) ; dans un deuxième temps, on modifie à nouveau **T** (**Sortir**) de sorte que ses éléments y soient rangés en ordre croissant.

Idée : on considère que les éléments du tableau à trier correspondent à un ensemble de couples <p,x> dans lesquels p = x. On adapte les algorithmes précédents en conséquence.

a) Version 1

k : un entier sur [1…n] *{taille du tas courant}*
PgF(i, k) : *{i ≤ k div 2}*

soit g = 2*i dans si g+1≤ k *{i est binaire}* et puis $T_g < T_{g+1}$ alors g+1 sinon g

{construction d'un tas dans T}
k ← 1
tant que k < n
{La valeur de T est une permutation de sa valeur initiale ; $T_{1...k}$ est un tas}
{Entrer T_{k+1} : coût majoré par $\lfloor log_2(k+1) \rfloor$}
k ← k+1 ; ic ← k
tant que ic > 1 et puis $T_{ic} > T_{ic\ div\ 2}$
$T_{ic} \leftrightarrow T_{ic\ div\ 2}$; ic ← ic div 2
{construction dans T d'une séquence en ordre croissant à partir du tas}
tant que k > 1
{La valeur de T est une permutation de sa valeur initiale ; T1…k est un tas ; Tk+1…n comporte les n-k plus grandes valeurs du T initial, en ordre croissant.}
{Sortir T_1 : coût majoré par $\lfloor log_2(k-1) \rfloor$}
$T_k \leftrightarrow T_1$; k ← k – 1 ; ic ← 1
tant que ic ≤ k quotient 2 et puis $T_{ic} < T_{PgF(ic, k)}$
$T_{ic} \leftrightarrow T_{PgF(ic, k)}$; ic ← PgF(ic, k)

b) analyse du coût

— création du tas initial : $\Sigma_{k = 2 \text{ à } n} \log_2(k) = \log_2 N!$, soit une complexité d'ordre $n*\log_2(n)$

— création du résultat : $\Sigma_{k = 1 \text{ à } n-1} \log_2(k-1)$, soit une complexité d'ordre $n*\log_2(n)$

L'algorithme a donc une complexité en nombre de comparaisons d'ordre $\mathbf{n*log_2(n)}$.

c) version 2: création du tas initial sans utiliser Entrer
Algorithme

Le tableau initial **T** représente un arbre binaire tassé. Les feuilles de cet arbre sont des arbres *TO*. L'idée est de donner progressivement la propriété *O* à **T**, en procédant par niveaux, des feuilles vers la racine : le traitement d'un niveau consiste à "descendre" les racines des sous-arbres de ce niveau.

{construction d'un tas dans T}
k ← n div 2
tant que k > 0
{∀ i ∈ [k+1…n] les arbres de racine i sont TO}
{donner la propriété O à l'arbre de racine k. Ses sous-arbres l'ont : descendre T_k.}
ic ← k
tant que ic ≤ n quotient 2 et puis $T_{ic} < T_{PgF(ic, n)}$
$T_{ic} \leftrightarrow T_{PgF(ic, n)}$; ic ← PgF(ic, n)

coût

On raisonne en regroupant les sommets par niveaux. Il y a $\mathbf{2^k}$ sous-arbres au niveau **k** et pour chacun, l'ordre de grandeur du coût de la descente de la racine est **h-k**, où $\mathbf{h = \lfloor log_2 n \rfloor}$:

$C = \Sigma_{k = 0 \text{ à } h-1} 2^k(h-k)$

On développe :

$C = h + 2(h-1) + 4(h-2) + ... + 2^i(h-i) + ... + 2^{h-1}$
$2C = 2*h + 4(h-1) + ... + 2^i(h-i+1) + ... + 2^{h-1}2 + 2^h$

Par différence, on obtient :

$C = -h + 2 + 4 + ... + 2^i + ... + 2^h$

Donc $C = -h + 2*(2^h - 1) = -\lfloor log_2 n \rfloor + 2(n-1)$

Ainsi la complexité de la construction du tas initial est maintenant d'ordre **n**.