TAS

Dans ce chapitre, il est sous-entendu que les arbres sont binaires. On travaille avec un ensemble E d'éléments x auxquels on associe une clef $v(x)$ appartenant à un ensemble orodnné V. Par exemple,

1 Généralités

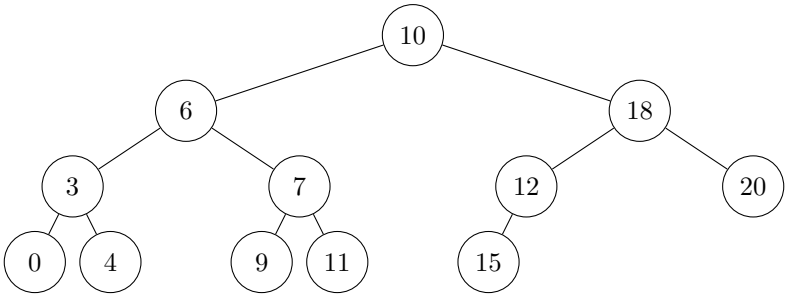
1.1 Quelques notations utilisées dans ce chapitre

Soit B un arbre sur E. Si s est un sommet, on note :
- $v(s)$ la clef du sommet s,
- $\mathrm{Fg}(s)$ et $\mathrm{Fd}(s)$ les fils gauche et droit de s (éventuellement vides),
- $S(B)$ l'ensemble des sommets de B,
- $N(B)$ l'ensemble des nœuds (internes) de B.

1.2 Définitions

Définition. Un arbre de hauteur h est complet à gauche quand
- le niveau de hauteur $h-1$ est rempli : il y a 2^{h-1} nœuds de hauteur $h-1$,
- toutes les feuilles de hauteur h sont à gauche de l'arbre

Exemple. Un arbre complet à gauche



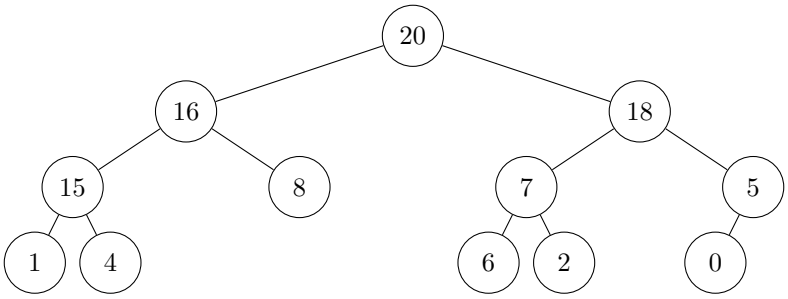
Dans un arbre complet à gauche de hauteur h, tous les niveaux sont remplis jusqu'au niveau $h-1$ et toutes les feuilles sont de hauteur h ou $h-1$, donc sa taille n vérifie l'inégalité $2^h \leqslant n < 2^{h+1}$, ou autrement dit $n = \lfloor \log n \rfloor$.

Proposition 1 *On réalise le parcours en largeur d'un arbre en différenciant les nœuds des feuilles et en marquant les fils vides des nœuds d'arité 1.*

Un arbre est complet à gauche si et seulement si la suite des clefs obtenues par ce type de parcours en largeur est de la forme (nœud,...,nœud, feuille,...,feuille) sans symbole de fils vide.

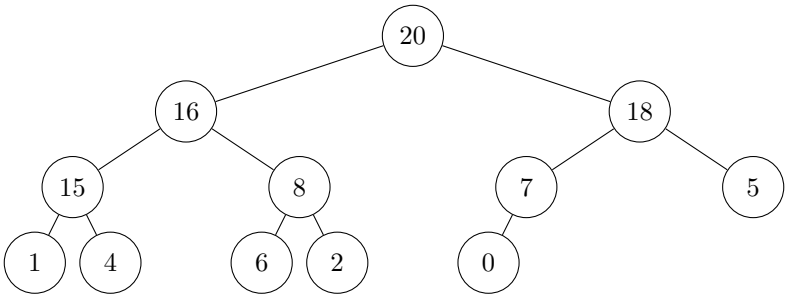
Définition. Un arbre est dit max-ordonné (resp. min-ordonné) quand chaque nœud a une clef supérieure (resp. inférieure) ou égale à celles de ses fils :
pour tout $s \in N(B)$, $v(\mathrm{Fd}(s)) \leqslant v(s)$ et $v(\mathrm{Fg}(s)) \leqslant v(s)$

Exemple. Un arbre max-ordonné d'entiers (ordonnés naturellement)



Définition. Un tas est un arbre binaire complet à gauche et ordonné. On parle de max-tas ou min-tas (aussi appelé tas-max ou tas-min).

Exemple.



Par conséquent, un tas de taille n et hauteur h vérifie l'inégalité $2^h \leqslant n < 2^{h+1}$, ou autrement dit $n = \lfloor \log n \rfloor$. Pour une taille fixée, il est donc de hauteur minimale : il est bien « tassé ».

2 Opérations faciles sur les tas

Les opérations principales sur les tas sont l'ajout et la suppression d'éléments. Dans la suite, on travaille sur des tas-max : en retournant le sens des inégalités, on a les mêmes opérations sur les tas-min.

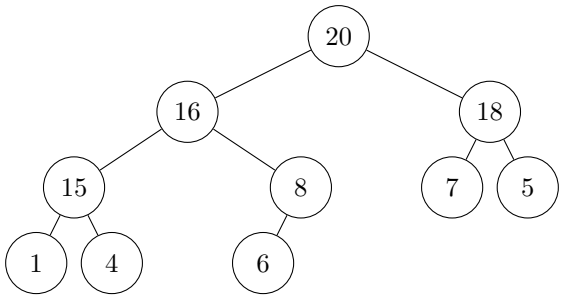
2.1 Ajout d'un élément

L'idée est simple à expliquer, plus difficile à mettre en œuvre :

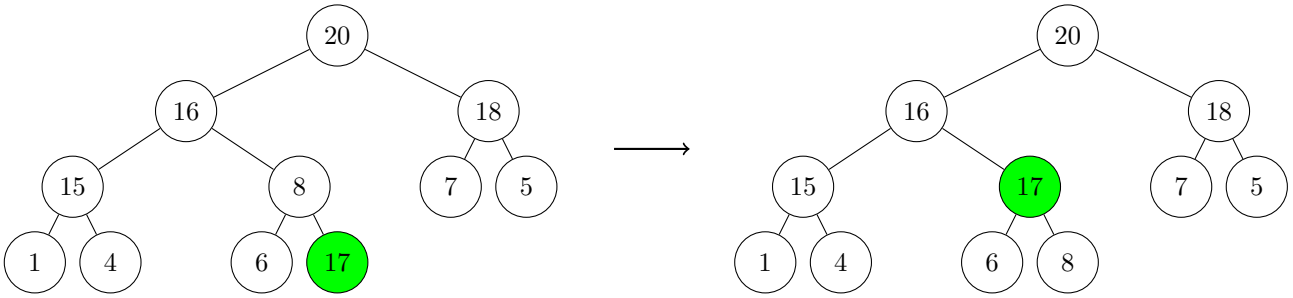
- on ajoute le nouvel élément sur le dernier niveau juste à droite du dernier élément (ou seul sur un nouveau niveau complètement à gauche si l'arbre est complet), de sorte à conserver la structure d'arbre complet à gauche, mais la structure d'arbre ordonné est en général perdue, il faut donc la retrouver ;
- on échange l'élément en question avec son parent tant que la clef de celui-ci est strictement inférieure à la clef du sommet en cours : on aboutit à un tas dans lequel on a remonté le nouvel élément à sa bonne position.

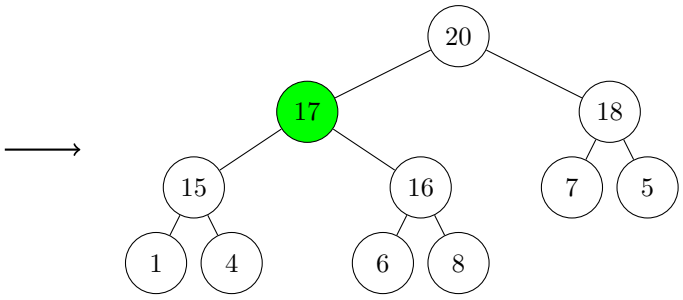
Exemple.

On part du tas suivant :



On ajoute l'élément 17 :





On a obtenu de nouveau un tas.

2.2 Suppression de la racine

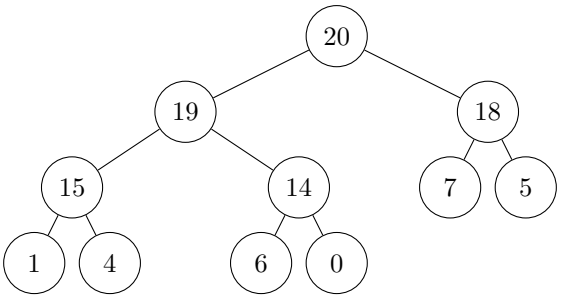
Là aussi, l'idée est simple à expliquer :

— on supprime la dernière feuille et on la met à la place de la racine ;

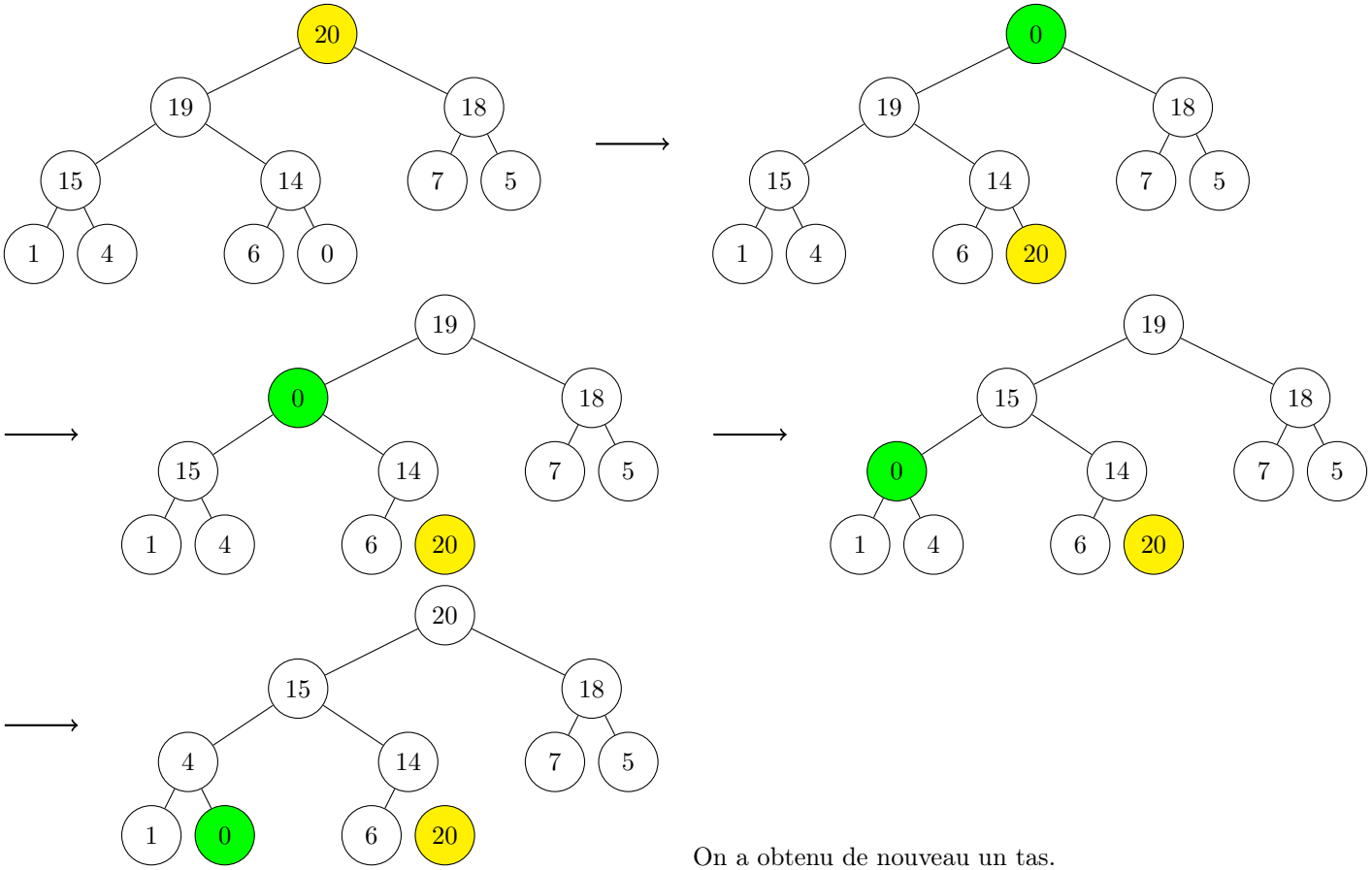
— tant que la clef de l'un des fils de ce nouveau sommet est strictement supérieure à la clef du sommet, on choisit le fils de plus garnde clef et on l'échange avec le nouveau sommet : on obtient un tas dans lequel on a descendu le nouveau sommet à une position correcte.

Exemple.

On part du tas suivant :



On supprime la racine (l'élément 20) :



On a obtenu de nouveau un tas.

2.3 Complexités

On constate aisément que la complexité de ces deux opérations est en $O(\quad) = O(\quad)$ (si on considère le calcul des clefs comme élémentaire).

Dans ces deux opérations, les difficulté sont de deux ordres :

— savoir trouver la dernière feuille de l'arbre ou en ajouter une derrière ;

— pour chaque sommet, savoir trouver rapidement son père, ce qui n'est pas facile sans reparcourir l'arbre systématiquement.

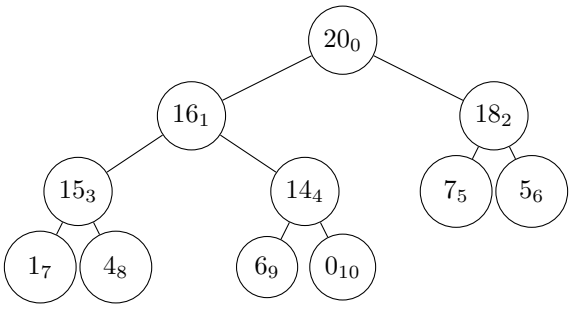
Pour résoudre ce problème, on va considérer une représentation des tas à l'aide de tableaux (tas mutables).

3 Représentation informatique des tas mutables

3.1 Tableau associé à un arbre binaire complet à gauche

Dans les algorithmes qui précèdent (qui sont de nature itératives plus que fonctionnels), il s'agit de trouver rapidement la fin du tas et le père de chaque sommet.

On considère un arbre binaire complet à gauche de taille n. Si on numérote les sommets dans l'ordre du parcours en largeur, la racine étant numérotée 0, la dernière feuille est numérotée $n-1$.



On associe à l'arbre le tableau à n cases contenant dans l'ordre du parcours en largeur les étiquettes des sommets : la case i contient l'étiquette du sommet numéroté i.

$$t = (20; 16; 15; 14; 6; 5; 1; 4; 6; 0)$$

Il est alors facile de lire dans le tableau les liens entre les sommets :

- la racine est $t[0]$;
- les deux fils du sommet associé à la case $t[i]$ ont leurs étiquettes dans les cases $t[2i+1]$ pour le fils gauche et $t[2i+2]$ pour le fils droit ;
- si $i \geqslant 1$, le père du sommet associé à la case $t[i]$ a son étiquette dans la case $t[(i-1)/2]$.

Par simple multiplication ou division par 2, on peut se déplacer dans l'arbre aussi bien vers le bas que vers le haut.

3.2 Application aux tas mutables

On représente un tas mutable par son tableau : ajouter ou supprimer un élément se fait directement dans le tableau.

Comme on ne peut pas facilement redimensionner les tableaux, on doit fixer une taille maximale des tas manipulables N. Une fois ce choix effectué, un tas de taille $n \leqslant N$ est représenté par un tableau de longueur N, comme précédemment. Il faut simplement une information supplémentaire : la taille du tas, qui doit être mise à jour lors des opérations (il faut savoir où se termine le tas dans le tableau).

On adopte par exemple la représentation suivante :

```
type 'a tas = {mutable taille : int; tab : 'a vect};;

let new x =
  {taille = 0; tab = make_vect N x};;
```

La fonction `new x` crée alors un tas vide de type identique à celui de `x`.

À l'aide de simples boucles, on peut alors remplir un tas, ajouter un élément ou supprimer la racine.

Remarque. En pratique, pour gagner encore un peu de temps, on ajoute une case au tableau et on commence la numérotation à 1. Dans ce cas, les fils du sommet i sont les sommets $2i$ et $2i+1$ et son père est le sommet $i/2$: ces opérations sur des entiers binaires dans une machine se font plus facilement par simple décalage de bits, ce qui permet d'éviter les additions ou soustractions, donc d'aller un peu plus vite.

4 Files de priorité

Soit E un ensemble d'objets et v une application de E dans un ensemble ordonné V.

Une file de priorité associée à E et v est une structure abstraite qui dispose des opérations suivantes :

- on peut créer une file de priorité initialement vide ;
- on peut ajouter un élément de E dans une file ;
- on peut retirer de la file l'élément de E de priorité maximale, c'est-à-dire l'élément x de la file telle que $v(x)$ soit maximal.

On souhaite bien sûr que ceci se fasse à un coût raisonnable en fonction de la longueur de la file de priorité (*i.e.* son nombre d'éléments).

On peut concrétiser la structure de file de priorité avec une liste quelconque ou une liste triée (complexités ?). Mais on peut faire mieux. D'après ce qu'on vient de voir sur les tas, ceux-ci sont de bons candidats, car la complexité des opérations est alors en $O(\log n)$.

Remarque. On ajoute souvent l'opération qui consiste à modifier la file quand on modifie la priorité d'un élément qu'elle contient.

Représentée par un tas, il suffit d'adapter les idées précédentes :
- si on modifie à la hausse la priorité d'un élément, alors on le remonte dans le tas comme lorsqu'on ajoute un élément ;
- si on modifie à la baisse la priorité d'un élément, alors on le descend dans le tas comme lorsqu'on supprime un élément ;

5 Tri par tas

5.1 Principe général

Soit t une structure (liste ou tableau) à n éléments à trier selon un certain critère v.

On applique l'algorithme suivant :
- on initialise un tas vide ;
- on parcourt la structure initiale et on ajoute successivement au tas les éléments selon la priorité définie par le critère de tri v ;
- on vide ensuite le tas : comme on récupère à chaque fois l'élément de priorité maximale, on peut alors créer la structure contenant les mêmes éléments rangés dans l'ordre sue l'on veut.

Cet algorithme est de complexité $O(n \log n)$.

5.2 Cas particulier du tri en place d'un tableau

On peut même appliquer le tri par tas en place dans un tableau.

D'abord on crée le tas par « ajouts successifs d'éléments » :
- le sous-tableau $t[0..0]$ est un tas ;
- pour i variant de 1 à n, si on a déjà rangé en tas le sous-tableau $t[0..(i-1)]$, alors on considère le sous-tableau $t[0..i]$ comme un tas auquel on vient d'ajouter un élément et on le retasse correctement en faisant remonter l'élément $t[i]$.

Puis on range le tableau en vidant le tas :
- pour i variant de n à 1, on échange le i-ème élément du tas avec le premier et on fait redescendre le premier jusqu'à sa bonne place ;
- en fin d'algorithme, le tableau est rangé par ordre **décroissant** du critère de tri.