Chapitre 3

Tas et file de priorité

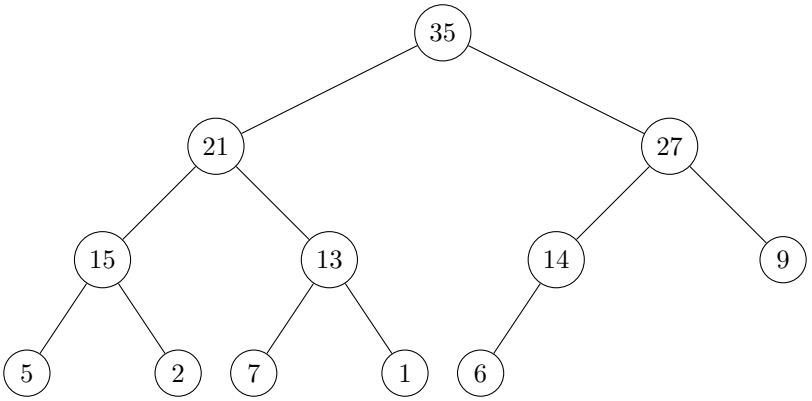
1 Tas binaire

1.1 Définition

Définition.

On appelle **tas binaire** (ou **tas-max**) tout arbre binaire parfait étiqueté par un ensemble totalement ordonné tel que, pour tout sommet autre que la racine, son étiquette est inférieure ou égale à l'étiquette de son père.

Exemple.



Remarques.

1. L'étiquette de la racine d'un tas-max est la valeur maximale des étiquettes des sommets de ce tas.
2. Nous pouvons définir de la même manière les tas-min : l'étiquette de chaque sommet autre que la racine est supérieure ou égale à l'étiquette de son père.

   Dans ce cas, l'étiquette de la racine d'un tas-min est la valeur minimale des étiquettes des sommets de ce tas.

1.2 Implémentation d'un tas

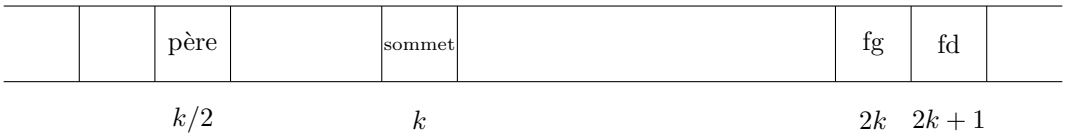
Comme tout arbre binaire, nous pouvons représenter un tas par une structure d'arbre.

Nous pouvons aussi proposer une autre représentation basée sur une numérotation des sommets en partant de la racine avec un parcours en largeur (aussi appelée numérotation Sosa-Stradonitz) :

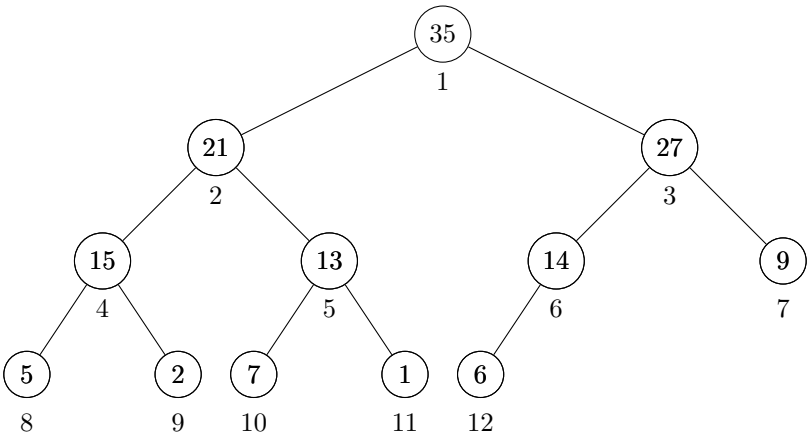
- la racine sera numérotée 1,
- si un sommet autre qu'une feuille est numéroté k, son fils gauche est numéroté $2k$ et son fils droit est numéroté $2k+1$.

Par conséquent, pour tout sommet autre que la racine numéroté k, son père est numéroté $\left\lfloor \frac{k}{2} \right\rfloor$.

Nous pouvons alors représenter un tas par une structure de tableau :



Exemple. Le tas suivant



peut être représenté par le tableau

Indice	1	2	3	4	5	6	7	8	9	10	11	12
Valeur	35	21	27	15	13	14	9	5	2	7	1	6

Remarques.

1. Dans le cas d'un arbre binaire quelconque, cette représentation ne présente en général pas d'intérêt car le nombre de cases inutilisées peut être très important.

   Par exemple, dans le cas extrême d'un arbre peigne gauche, seules les cases correspondant aux puissances de 2 sont utilisées, ce qui nécessiterait un coût spatial exponentiel pour le représenter.

2. En OCaml, les tableaux sont indexés à partir de 0... Deux possibilités :

   (1) Soit nous conservons la numérotation définie ci-dessus, la case d'indice 0 du tableau est alors inutile.

   (2) Soit nous numérotons à partir de 0 et dans ce cas :

   - la racine sera numérotée 0,
   - si un sommet autre qu'une feuille est numéroté k, son fils gauche est numéroté $2k+1$ et son fils droit est numéroté $2k+2$.

   Par conséquent, pour tout sommet autre que la racine numéroté k, son père est numéroté $\left\lfloor \frac{k-1}{2} \right\rfloor$.

   Nous poursuivrons l'exposé avec cette numération à partir de 0.

3. Considérons un tas ayant n sommets représenté par un tableau de longueur n (dont les indices commencent à 0) et k appartenant à $[\![0, n-1]\!]$.
   - Dans le cas où $2k+2 < n$, le sommet numéroté k a 2 fils, son fils gauche est numéroté $2k+1$ et son fils droit est numéroté $2k+2$.
   - Dans le cas où $2k+2 = n$, le sommet numéroté k a 1 fils et son fils (gauche) est numéroté $2k+1$.
   - Dans le cas où $2k+2 > n$, le sommet numéroté k a 0 fils. C'est donc une feuille.

   En particulier, les éléments d'indices $\left\lfloor \frac{n}{2} \right\rfloor$ à $n-1$ sont les feuilles de l'arbre.

2 Opérations usuelles sur les tas

2.1 Structure de tas

Nous considérons un tableau t et nous nous demandons si ce tableau représente un tas-max.

Par contrôle des fils.

L'idée est de s'assurer que, pour tout sommet du tas, les valeurs des étiquettes de ses éventuels fils sont bien inférieures ou égales à la valeur de l'étiquette du sommet.

Version itérative.

Version récursive.

Par contrôle des pères.

L'idée est de s'assurer que pour tout sommet du tas, la valeur de l'étiquette de son éventuel père est bien supérieure ou égale à la valeur de l'étiquette du sommet.

Version itérative.

Version récursive.

2.2 Modification de la valeur de l'étiquette d'un sommet

Pour conserver la structure de tas lors de la modification de la valeur d'une étiquette, nous pouvons être amenés :

- soit à remonter la valeur dans le tas,
- soit à descendre la valeur dans le tas.

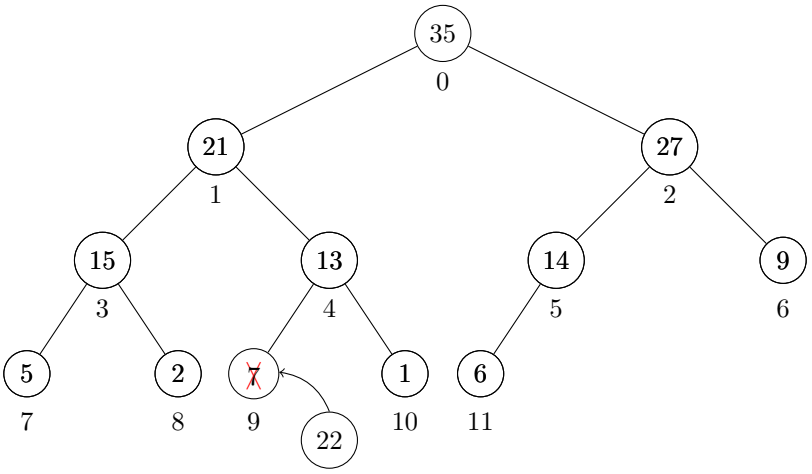
Nous parlerons de **percolation ascendante** ou de **percolation descendante**.

On donne la fonction suivante d'échange dans un tableau utile pour la suite :

```
let echange t i j = let aux = t.(i) in t.(i) <- t.(j); t.(j) <- aux;;
```

Percolation ascendante.

Dans le cas de la percolation ascendante, nous souhaitons remplacer la valeur de l'étiquette d'un sommet par une valeur supérieure :



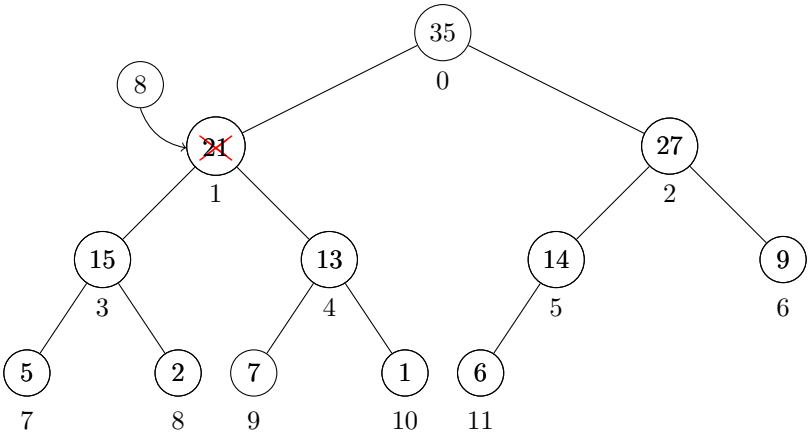
Tant que l'étiquette de son père est strictement inférieure à son étiquette, on l'échange avec son père. Plus précisément, on affecte la valeur x à la case d'indice k du tas t puis :

- si $k = 0$, on ne fait rien ;
- si $k \geq 1$ et si l'étiquette du sommet est strictement supérieure à celle de son père, on échange leurs étiquettes et on fait un appel récursif sur le sommet père.

Implémentation.

Percolation descendante.

Dans le cas de la percolation descendante, nous souhaitons remplacer la valeur de l'étiquette d'un sommet par une valeur inférieure :



Tant que l'étiquette du sommet est strictement inférieure à l'étiquette d'un de ses fils, on l'échange avec son fils de plus grande étiquette. Plus précisément, on affecte la valeur x à la case d'indice k du tas t puis :

- si $2k+2 > n$, on ne fait rien ;
- si $2k+2 = n$ et si l'étiquette du sommet est strictement inférieure à celle de son unique fils, on les échange ;
- si $2k+2 < n$ et si l'étiquette du sommet est strictement inférieure à celle de l'un de ses fils (c'est-à-dire à celle du fils d'étiquette maximale), on échange le sommet avec son fils d'étiquette maximale et on fait un appel récursif sur ce sommet fils

Implémentation.

Complexité. Un mot sur la complexité des fonctions de percolation. La taille de donnée est le nombre n de sommets du tas (la longueur du tableau), l'opération fondamentale est l'échange de deux éléments dans un tableau et dans le pire des cas, la percolation se fait sur toute la hauteur du tas. Donc :

$$C(n) = O(\log_2(n)).$$

2.3 Construction d'un tas

Nous disposons d'un tableau de longueur n et nous souhaitons le modifier en tas. Deux méthodes possibles : la remontée des sommets ou la descente des pères.

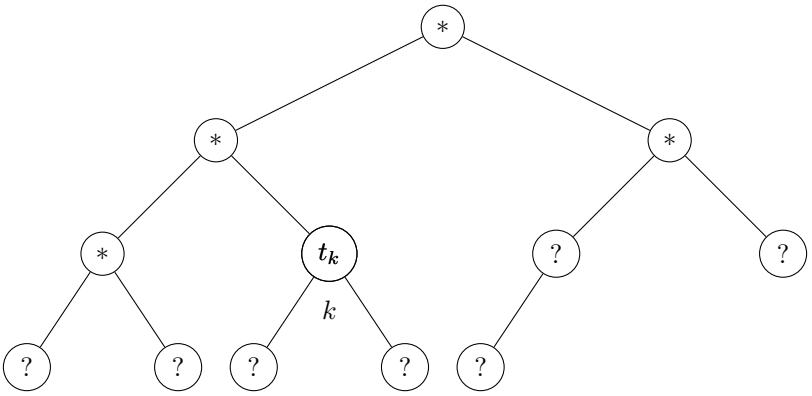
La remontée des sommets.

L'algorithme est basé sur le principe de percolation ascendante. Il consiste à maintenir l'invariant de boucle

$$\text{"}t[0\dots k] \text{ est un tas-max"}$$

en faisant remonter l'élément t_k dans le tas situé à sa gauche à l'aide de la fonction de percolation ascendante.

On est donc dans la situation suivante :



Implémentation.

Complexité. La taille de donnée est la longueur n du tableau et l'opération fondamentale est l'échange de deux éléments dans un tableau.

Dans le pire des cas, chaque remontée aboutit à la racine (c'est le cas par exemple lorsque le tableau est initialement trié par ordre croissant). Dans ce cas, le nombre d'échanges effectués est égal à la somme des profondeurs de chacun des sommets à l'exception de la racine. Si on note $p = \lfloor \log_2(n) \rfloor$ la hauteur du tas, on a donc :

$$C(n) = \sum_{k=1}^{p-1} k2^k + p(n - 2^p + 1).$$

Rappelons que, si $x \neq 1$,

$$\sum_{k=0}^{p-1} x^k = \frac{1-x^p}{1-x} \quad \text{et en dérivant} \quad \sum_{k=1}^{p-1} kx^{k-1} = \frac{-px^{p-1}(1-x) + (1-x^p)}{(1-x)^2}.$$

Finalement :

$$C(n) = (n+1)p - 2^{p+1} + 2 = (n+1)\lfloor \log_2(n) \rfloor - 2 \times 2^{\lfloor \log_2(n) \rfloor} + 2 \leq (n+1)\log_2(n).$$

Ainsi, $C(n) = O(n \log_2(n))$ et la complexité est donc quasi-linéaire.

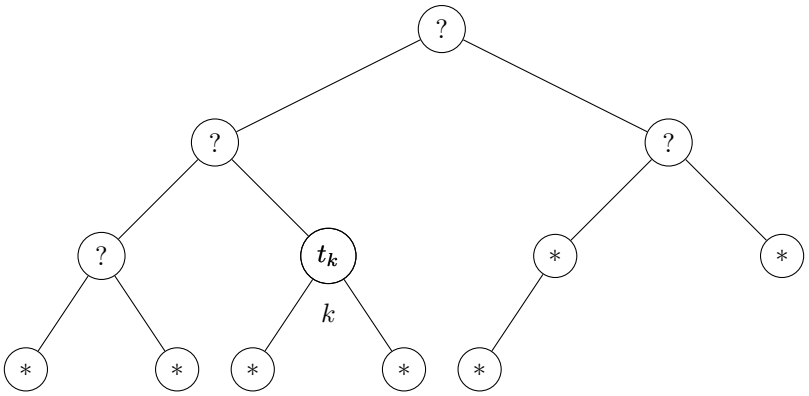
La descente des pères.

L'algorithme est basé sur le même principe mais par percolation descendante. Il consiste à maintenir l'invariant de boucle

"tous les sommets de $t[k \dots (n-1)]$ sont les racines d'un tas-max"

pour k diminuant de $n-1$ à 0 en faisant descendre l'élément t_k dans le tas situé à sa droite à l'aide de la fonction de percolation descendante.

On est donc dans la situation suivante :



Remarque. Considérons un tas constitué de n sommets représenté par un tableau t.

Comme l'indice du fils droit d'un sommet d'indice k est égal à $2k+2$, k est le numéro d'une feuille si et seulement si $2k+2 > n$, c'est-à-dire si et seulement si $k > n/2 - 1$. Ainsi, les feuilles du tas t correspondent aux indices de $\lfloor n/2 \rfloor$ à $(n-1)$.

Comme $t[n/2 \dots (n-1)]$ contient les feuilles de t et que chacune de ces feuilles est un tas à un élément, tous les sommets de $t[n/2 \dots (n-1)]$ sont bien les racines d'un tas-max. Il nous suffit donc de faire diminuer k de $n/2$ à 0.

Implémentation.

Complexité. La taille de donnée est la longueur n du tableau et l'opération fondamentale est l'échange de deux éléments dans un tableau. Notons $p = \lfloor \log_2(n) \rfloor$ la hauteur du tas.

Pour tout k appartenant à $[\![0, p-1]\!]$, le nombre de sommets de profondeur k est égal à 2^k et le nombre de sommets de profondeur p est compris entre 1 et 2^p. Et pour tout k appartenant à $[\![0, p]\!]$, pour tout sommet de profondeur k, la percolation descendante amènera au plus $p-k$ échanges.

Donc :

$$C(n) = \sum_{k=0}^{p}(p-k)2^k = \sum_{k=0}^{p-1}(p-k)2^k = p\sum_{k=0}^{p-1}2^k - 2\sum_{k=0}^{p-1}k2^{k-1} = p(2^p-1) - 2\sum_{k=0}^{p-1}k2^{k-1}.$$

Avec la même formule que précédemment, $\sum_{k=0}^{p-1} k2^{k-1} = p2^{p-1} + 1 - 2^p$. Finalement :

$$C(n) = p(2^p - 1) - 2(p2^{p-1} + 1 - 2^p) = -p - 2 + 2^{p+1} \leq 2^{p+1} = 2^{\lfloor \log_2(n) \rfloor + 1} \leq 2^{\log_2(n)+1} = 2n.$$

Ainsi, $C(n) = O(n)$ et la complexité est donc linéaire.

Remarque. La descente des pères est donc une méthode plus efficace que la remontée des sommets qui avait une complexité quasi-linéaire.

2.4 Fonctions de conversion

Un tas peut être représenté par une structure d'arbre ou par une structure de tableau.

Conversion d'un arbre en un tableau.

Conversion d'un tableau en un arbre.

2.5 Le tri par tas.

Considérons un tableau t de n valeurs d'un ensemble totalement ordonné à trier par valeur croissante. L'idée du **tri par tas** (ou **heap sort**) est la suivante :

- Nous commençons par transformer le tableau t en un tas.
- La plus grande valeur contenue dans le tableau t se trouve alors à l'indice 0. Nous échangeons alors les éléments d'indice 0 et d'indice $n-1$.
- Nous transformons de nouveau le sous-tableau $t[0 \dots n-2]$ en un tas par simple percolation descendante de l'élément d'indice 0.
- La plus grande valeur contenue dans le sous-tableau $t[0 \dots n-2]$ se trouve alors à l'indice 0. Nous échangeons alors les éléments d'indice 0 et d'indice $n-2$.
- Nous transformons de nouveau le sous-tableau $t[0 \dots n-3]$ en un tas par simple percolation descendante de l'élément d'indice 0.
- ...

De manière générale, k diminuant de $n-1$ à 1, nous trouvons dans la case d'indice 0 du tableau t la plus grande valeur de $t[0 \dots k]$, nous échangeons alors les éléments d'indice 0 et d'indice k puis nous transformons de nouveau le sous-tableau $t[0 \dots k-1]$ en un tas par simple percolation descendante de l'élément d'indice 0 (complexité logarithmique).
Ainsi, nous conservons l'invariant de boucle suivant :

> "À la fin de la boucle d'indice k, le sous-tableau $t[0 \dots k-1]$ est un tas-max contenant les k plus petits éléments de t, et le sous-tableau $t[k \dots n-1]$ contient les $n-k$ plus grands éléments de t, triés."

Ce qui prouve la correction de l'algorithme.

Implémentation.

Complexité. La taille de donnée est la longueur n du tableau et l'opération fondamentale est l'échange de deux éléments dans un tableau. La transformation d'un tableau en tas a un coût temporel en $O(n)$ et la percolation descendante un coût en $O(\log_2(n))$. Donc :

$$C(n) = O(n \log_2(n)).$$

3 Files de priorité

3.1 Structure de données abstraite

Les **files de priorité** permettent de gérer un ensemble d'objets, chacun de ces objets étant associé à un élément d'un ensemble totalement ordonné appelé **priorité**. Dans une file de priorité, l'élément le plus important (de plus grande priorité) sort le premier.

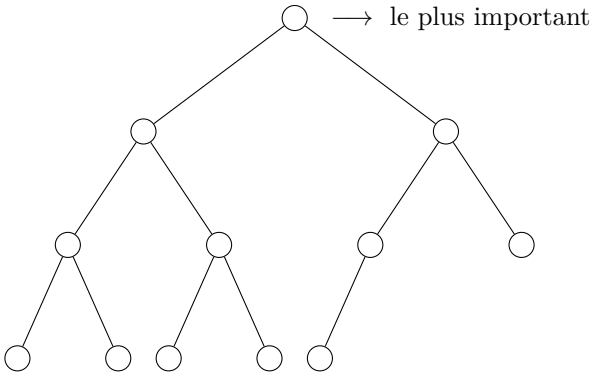
Exemple. Une file de priorité permet de planifier les tâches d'un ordinateur : la file gère les travaux en attente, avec leurs priorités relatives. Lorsqu'une tâche est terminée, l'ordinateur exécute la tâche de plus forte priorité parmi les travaux en attente.

Comme pour toute structure de données abstraite, nous allons développer quelques primitives agissant sur les files de priorité comme :

- la création d'une file de priorité,
- l'ajout d'un élément dans une file de priorité,
- la suppression de l'élément de plus grande priorité,
- l'augmentation de la priorité d'un élément,

et tout ceci dans le respect d'une complexité satisfaisante.

Une file de priorité sera représentée par un tas-max dans lequel chaque sommet est étiqueté par un couple (valeur, priorité), l'ordre étant donné par la priorité :



3.2 Structure de données concrète

Ici, un tas est représenté par un tableau et tout sommet du tas est un couple (valeur, priorité) que nous décidons d'implémenter sous la forme d'un type enregistrement :

```
type ('a, 'b) sommet = {valeur : 'a; mutable priorite : 'b} ;;
```

Par exemple :

```
let sommet1_ex = {valeur = 1; priorite = 23} ;;
```

La taille de la file de priorité pouvant augmenter, nous rencontrons un problème avec la taille statique d'un tableau. Aussi, au moment de sa création, nous choisirons un tableau de taille arbitrairement très grande et nous associerons à ce tableau, un entier pour préciser le nombre d'éléments dans la file de priorité.
Une file de priorité sera donc un enregistrement contenant la taille et le tas :

```
type ('a, 'b) file_priorite = {mutable taille : int; tab : ('a, 'b) sommet array} ;;
```

Par exemple :

```
let file_ex = {taille = 5; tab = [|{valeur = 1; priorite = 35}; {valeur = 2; priorite = 23};
 {valeur = 3; priorite = 19}; {valeur = 4; priorite = 11}; {valeur = 5; priorite = 3};
 {valeur = 6; priorite = 85}; {valeur = 7; priorite = 45}|]} ;;
```

Création d'une file de priorité vide.

Ajout d'un élément.

L'idée est tout simplement d'ajouter l'élément en fin de file et de le remonter afin de conserver la structure de tas.

Suppression de l'élément prioritaire.

L'idée est d'échanger le premier élément de la file avec le dernier puis de descendre le premier élément de la file afin de conserver la structure de tas.

Augmentation de la priorité d'un élément.

L'idée est, après avoir augmenté la priorité de l'élément numéro k, de le remonter dans la file.