File de priorité

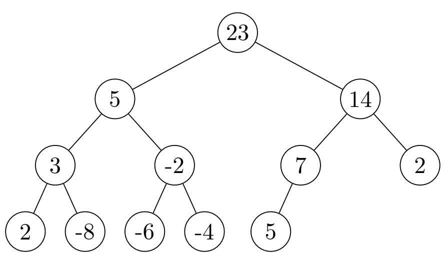
- Une **file de priorité max** (FP max) est une structure de données possédant les opérations suivantes :
  - Extraire maximum : supprime et renvoie le maximum
  - Ajouter élément
  - Tester si la FP est vide

  On définit une FP min en remplaçant maximum par minimum.

- Il est possible d'utiliser un arbre binaire de recherche pour implémenter une FP max, en utilisant le fait que le maximum est tout à droite de l'arbre. Les opérations d'ajout et d'extraction sont alors linéaires en la hauteur de l'arbre. On peut mettre à jour un élément (changer sa valeur) en le supprimant puis en le réajoutant (avec la nouvelle valeur).

Tas

- Un **tas max** est un arbre binaire presque complet (tous les niveaux, sauf le dernier, sont complets) où chaque noeud est plus grand que ses fils.
  Remarque : la racine est le maximum du tas.



Exemple de tas max

  Un tas min est comme un tas max, sauf que chaque noeud doit être plus petit que ses fils.

- Soit T un arbre binaire presque complet de hauteur h et avec n noeuds. Alors $h = \lfloor \log_2(n) \rfloor$ (donc $\boxed{h = \mathrm{O}(\log(n))}$).

  > Preuve : T contient plus de sommets qu'un arbre binaire complet de hauteur $h-1$ et moins de sommets qu'un arbre binaire complet de hauteur h, donc :
  >
  > $$\begin{aligned} & 2^h - 1 < n \leq 2^{h+1} - 1 \\ \Longrightarrow\ & 2^h \leq n < 2^{h+1} \\ \Longrightarrow\ & h \leq \log_2(n) < h+1 \\ \Longrightarrow\ & h = \lfloor \log_2(n) \rfloor \end{aligned}$$

- On stocke les noeuds du tas dans un tableau `t` tel que :
  - `t.(0)` est la racine de `t`.
  - `t.(i)` a pour fils `t.(2*i + 1)` et `t.(2*i + 2)`, si ceux-ci sont définis.

  Le père de `t.(j)` est donc `t.((j - 1)/2)` (si $j \neq 0$).
  Exemple : le tas en exemple ci-dessus est représenté par `t = [|23; 5; 14; 3; -2; 7; 2; 2; -8; -6; -4; 5|]`.

- En pratique, comme on ne peut pas ajouter d'élément à un tableau, on utilise un tableau `t` plus grand que le nombre d'éléments du tas pour pouvoir y ajouter des éléments. On stocke le nombre d'éléments dans une variable `n`.

```
type 'a tas = {t : 'a array; mutable n : int}

let swap tas i j = (* échange tas.t.(i) et tas.t.(j) *)
    let tmp = tas.t.(i) in
    tas.t.(i) <- tas.t.(j);
    tas.t.(j) <- tmp
```

- On utilise deux fonctions auxiliaires pour rétablir la propriété de tas après modification :
  - `up tas i` : suppose que `tas` est un tas max sauf `tas.t.(i)` qui peut être supérieur à son père. Fait monter (on dit aussi percoler) `tas.t.(i)` de façon à obtenir un tas max.
  - `down tas i` : suppose que `tas` est un tas max sauf `tas.t.(i)` qui peut être inférieur à un fils. Fait descendre `tas.t.(i)` de façon à obtenir un tas max.

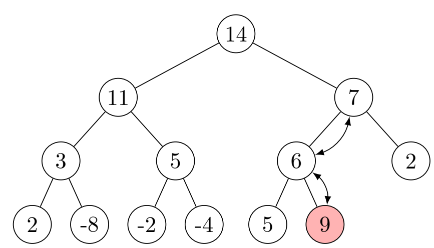
```
let rec up h i =
    let p = (i - 1)/2 in
    if i <> 0 && tas.t.(p) < tas.t.(i) then (
        swap h i p;
        up h p
    )

let rec down tas i =
  let m = ref i in (* maximum parmi i et ses fils *)
  if 2*i + 1 < tas.n && tas.t.(2*i + 1) > tas.t.(!m)
  then m := 2*i + 1;
  if 2*i + 2 < tas.n && tas.t.(2*i + 2) > tas.t.(!m)
  then m := 2*i + 2;
  if !m <> i then (
    swap h i !m;
    down h !m
  )
```

- Pour ajouter un élément à un tas : l'ajouter comme dernière feuille (dernier indice du tableau) puis faire remonter.
  Complexité : $\mathrm{O}(h) = \boxed{\mathrm{O}(\log(n))}$.

```
let ajoute tas e =
    tas.t.(tas.n) <- e;
    up h tas.n;
    tas.n <- tas.n + 1
```

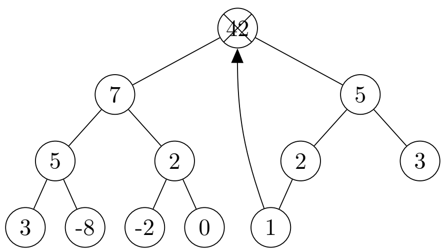
  Exemple : ajout de 9 dans un tas.



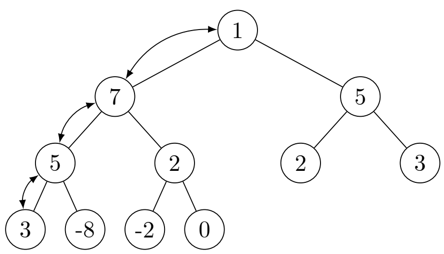
- Pour extraire le maximum d'un tas : échanger la racine avec la dernière feuille puis descendre la nouvelle racine.
  Complexité : $\mathrm{O}(h) = \boxed{\mathrm{O}(\log(n))}$.

```
let rec extract_max h =
    swap h 0 (h.n - 1);
    h.n <- h.n - 1;
    down h 0;
    h.a.(h.n)
```

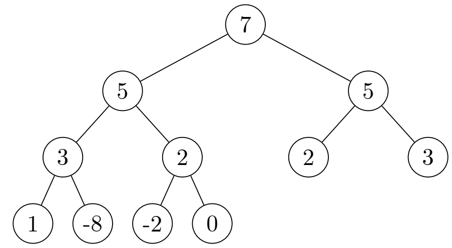
Exemple :



Suppression de la racine qu'on remplace par la dernière feuille



Percolation



Tas obtenu après extraction du maximum

Applications

- **Tri avec une file de priorité** : on ajoute tous les éléments dans un tas puis on les extrait un par un.
  On obtient ainsi le **tri par tas** en $O(n \log(n))$ avec un tas. On peut même utiliser le tableau en entrée comme tableau `tas.t` du tas, ce qui permet d'obtenir un tri en place, c'est-à-dire sans utiliser de tableau supplémentaire ($O(1)$ en mémoire).
- **Algorithme de Dijkstra**, pour extraire le sommet de distance estimée minimum à chaque itération. Dans le pseudocode suivant, on peut utiliser une file de priorité pour `q` :

```
Initialement : q contient tous les sommets
               dist.(s) <- 0
               dist.(v) <- infini, si v <> s

Tant que q est non vide:
    Extraire u de q tel que dist.(u) soit minimum
    Pour tout voisin v de u:
        Si dist.(u) + w(u, v) < dist.(v):
            dist.(v) <- dist.(u) + w(u, v)
```

- **Algorithme de Kruskal**, pour obtenir l'arête de poids minimum à chaque itération.