File de priorité et tas

Quentin Fortier

November 8, 2023

File de priorité (FP)

Une **file de priorité max** (FP max) est une structure de données possédant les opérations :

1. extraire maximum : supprime et renvoie le maximum
2. ajouter élément
3. tester si la FP est vide
4. (mettre à jour un élément)

File de priorité (FP)

Une **file de priorité max** (FP max) est une structure de données possédant les opérations :

1. extraire maximum : supprime et renvoie le maximum
2. ajouter élément
3. tester si la FP est vide
4. (mettre à jour un élément)

Une FP max est utilisée lorsque l'on a souvent besoin de trouver le maximum.
On définit une FP min en remplaçant maximum par minimum.

File de priorité (FP)

Une **file de priorité max** (FP max) est une structure de données possédant les opérations :

1. extraire maximum : supprime et renvoie le maximum
2. ajouter élément
3. tester si la FP est vide
4. (mettre à jour un élément)

Une FP max est utilisée lorsque l'on a souvent besoin de trouver le maximum.
On définit une FP min en remplaçant maximum par minimum.

Exercice

Donner des implémentations possibles de FP.

File de priorité max

Implémentation avec liste triée en décroissant :

1. extraire maximum :

File de priorité max

Implémentation avec liste triée en décroissant :

1. extraire maximum : en $O(1)$
2. ajouter élément :

File de priorité max

Implémentation avec liste triée en décroissant :

1. extraire maximum : en $\mathrm{O}(1)$
2. ajouter élément : en $\mathrm{O}(n)$
3. mettre à jour : en $\mathrm{O}(n)$

File de priorité max

Implémentation avec arbre binaire de recherche (ABR) équilibré (par exemple AVL ou ARN) :

1. extraire maximum :

File de priorité max

Implémentation avec arbre binaire de recherche (ABR) équilibré (par exemple AVL ou ARN) :

1. extraire maximum : en $O(\log(n))$ (sommet tout à droite)
2. ajouter élément :

File de priorité max

Implémentation avec arbre binaire de recherche (ABR) équilibré (par exemple AVL ou ARN) :

1. extraire maximum : en $\mathrm{O}(\log(n))$ (sommet tout à droite)
2. ajouter élément : en $\mathrm{O}(\log(n))$
3. mettre à jour :

File de priorité max

Implémentation avec arbre binaire de recherche (ABR) équilibré (par exemple AVL ou ARN) :

1. extraire maximum : en $O(\log(n))$ (sommet tout à droite)
2. ajouter élément : en $O(\log(n))$
3. mettre à jour : en $O(\log(n))$

C'est une bonne implémentation mais il y a plus efficace en pratique...

Tas max

L'implémentation de FP la plus utilisée est un **tas binaire max** :

1. un **arbre binaire**...

Tas max

L'implémentation de FP la plus utilisée est un **tas binaire max** :

1. un **arbre binaire**...
2. ... **presque complet** : tous les niveaux sont complets, sauf éventuellement le dernier niveau ...

Tas max

L'implémentation de FP la plus utilisée est un **tas binaire max** :

1. un **arbre binaire**...
2. ... **presque complet** : tous les niveaux sont complets, sauf éventuellement le dernier niveau ...
3. ... dont **chaque sommet est supérieur à ses éventuels fils**.

Tas max

L'implémentation de FP la plus utilisée est un **tas binaire max** :

1. un **arbre binaire**...
2. ... **presque complet** : tous les niveaux sont complets, sauf éventuellement le dernier niveau ...
3. ... dont **chaque sommet est supérieur à ses éventuels fils**.

À ne pas confondre avec un ABR !

La racine contient

Tas max

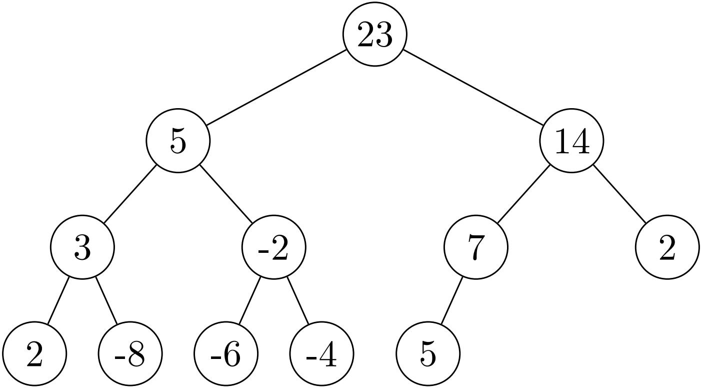
L'implémentation de FP la plus utilisée est un **tas binaire max** :

1. un **arbre binaire**...
2. ... **presque complet** : tous les niveaux sont complets, sauf éventuellement le dernier niveau ...
3. ... dont **chaque sommet est supérieur à ses éventuels fils**.

À ne pas confondre avec un ABR !

La racine contient le maximum.
(le minimum est une feuille).

Tas max



Le dernier niveau est rempli de gauche à droite.

Tas max

On considère un arbre binaire à n sommets et de hauteur h.

S'il est complet :

Tas max

On considère un arbre binaire à n sommets et de hauteur h.

S'il est complet :

$$n = \sum_{k=0}^{h} 2^k$$

$$\boxed{n = 2^{h+1} - 1}$$

Tas max

Un arbre binaire presque complet a son nombre de sommets n compris entre un arbre complet de hauteur $h-1$ et un arbre complet de hauteur h :

Tas max

Un arbre binaire presque complet a son nombre de sommets n compris entre un arbre complet de hauteur $h-1$ et un arbre complet de hauteur h :

$$
\begin{aligned}
& 2^h - 1 < n \leq 2^{h+1} - 1 \\
\implies\ & 2^h \leq n < 2^{h+1} \\
\implies\ & h \leq \log_2(n) < h + 1 \\
\implies\ & \boxed{h = \lfloor \log_2(n) \rfloor}
\end{aligned}
$$

Donc $\boxed{h = \mathsf{O}(\log(n))}$.

Représentation des tas max

On peut représenter efficacement un arbre binaire a presque complet (donc aussi un tas max) par un tableau a tel que :

1. `a.(0)` est la racine de `a`.
2. `a.(i)` a pour fils `a.(2*i + 1)` et `a.(2*i + 2)`, si ceux-ci sont définis.

Représentation des tas max

On peut représenter efficacement un arbre binaire a presque complet (donc aussi un tas max) par un tableau a tel que :

1. `a.(0)` est la racine de a.
2. `a.(i)` a pour fils `a.(2*i + 1)` et `a.(2*i + 2)`, si ceux-ci sont définis.

Le père de `a.(j)` est donc

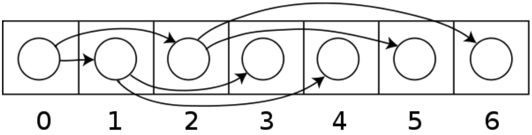
Représentation des tas max

On peut représenter efficacement un arbre binaire a presque complet (donc aussi un tas max) par un tableau a tel que :

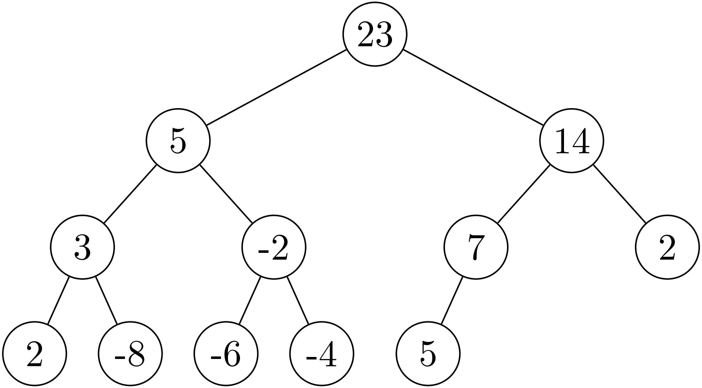
1. `a.(0)` est la racine de a.
2. `a.(i)` a pour fils `a.(2*i + 1)` et `a.(2*i + 2)`, si ceux-ci sont définis.

Le père de `a.(j)` est donc `a.((j - 1)/2)` (si j $\neq 0$)

Ainsi, on accède au père et au fils d'un sommet en O(1).

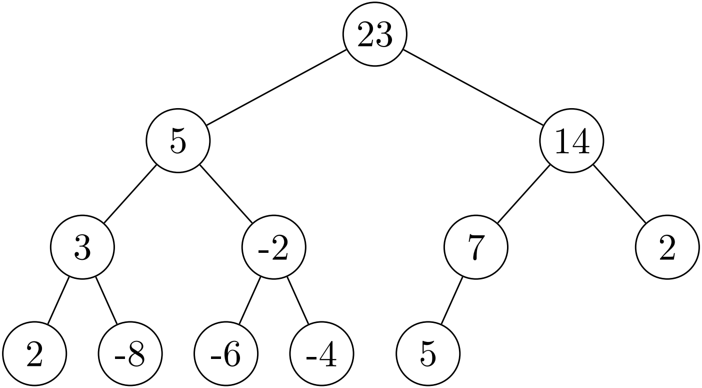


Représentation des tas max



est représenté par :

Représentation des tas max

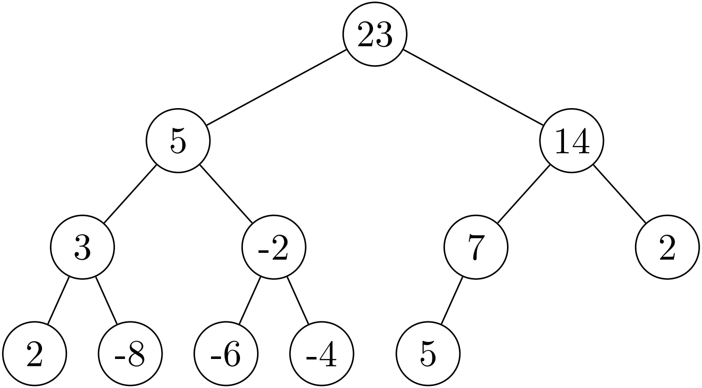


est représenté par :

`[|23; 5; 14; 3; -2; 7; 2; 2; -8; -6; -4; 5; ...|]`

C'est

Représentation des tas max



est représenté par :

`[|23; 5; 14; 3; -2; 7; 2; 2; -8; -6; -4; 5; ...|]`

C'est le **parcours en largeur du tas** !

Représentation des tas max

On peut utiliser des fonctions utilitaires de manipulation de tas :

```
type 'a heap = { a : 'a array; mutable n : int }

let swap h i j =
    let tmp = h.a.(i) in
    h.a.(i) <- h.a.(j);
    h.a.(j) <- tmp
```

Représentation des tas max

On peut utiliser des fonctions utilitaires de manipulation de tas :

```
type 'a heap = { a : 'a array; mutable n : int }

let swap h i j =
    let tmp = h.a.(i) in
    h.a.(i) <- h.a.(j);
    h.a.(j) <- tmp
```

n est le nombre d'éléments du tas (les indices de a après n sont ignorés).

Les feuilles sont d'indices

Représentation des tas max

On peut utiliser des fonctions utilitaires de manipulation de tas :

```
type 'a heap = { a : 'a array; mutable n : int }

let swap h i j =
    let tmp = h.a.(i) in
    h.a.(i) <- h.a.(j);
    h.a.(j) <- tmp
```

n est le nombre d'éléments du tas (les indices de a après n sont ignorés).
Les feuilles sont d'indices $\left\lfloor \frac{n}{2} \right\rfloor$ à $n - 1$.

Opérations de tas max

On utilise deux fonctions auxiliaires pour implémenter les opérations sur un tas max `h` et un indice `i` de `h.a` :

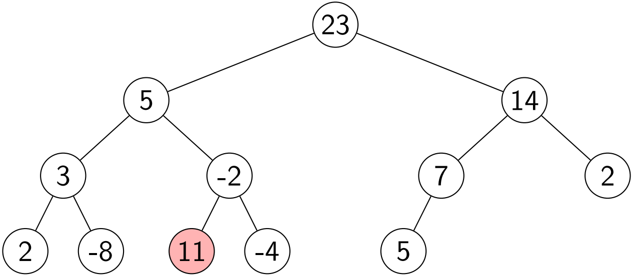
1. `up h i` : suppose que `h` est un tas max sauf `h.a.(i)` qui peut être supérieur à son père.
   Fait monter `h.a.(i)` de façon à obtenir un tas max.

Opérations de tas max

On utilise deux fonctions auxiliaires pour implémenter les opérations sur un tas max `h` et un indice `i` de `h.a` :

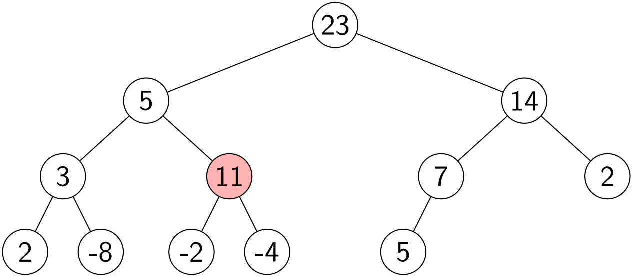
1. `up h i` : suppose que `h` est un tas max sauf `h.a.(i)` qui peut être supérieur à son père.
   Fait monter `h.a.(i)` de façon à obtenir un tas max.
2. `down h i` : suppose que `h` est un tas max sauf `h.a.(i)` qui peut être inférieur à un fils.
   Fait descendre `h.a.(i)` de façon à obtenir un tas max.

up



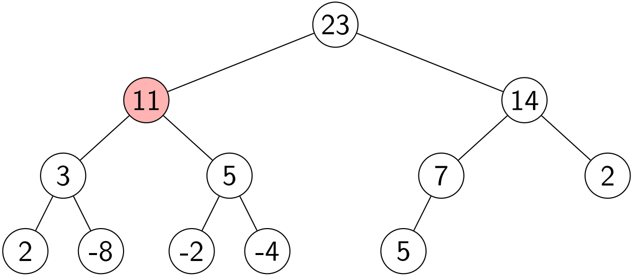
[|23; 5; 14; 3; -2; 7; 2; 2; -8; 11; -4; 5; ... |]

up



[|23; 5; 14; 3; 11; 7; 2; 2; -8; -2; -4; 5; ... |]

up



[|23; 11; 14; 3; 5; 7; 2; 2; -8; -2; -4; 5; ... |]

up

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

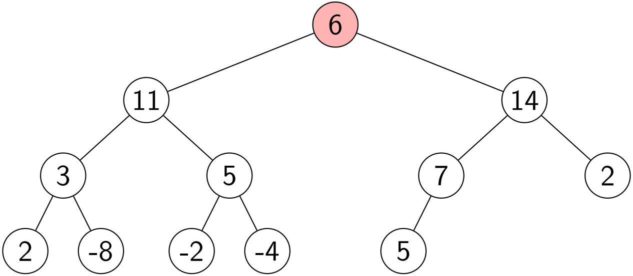
Complexité :

up

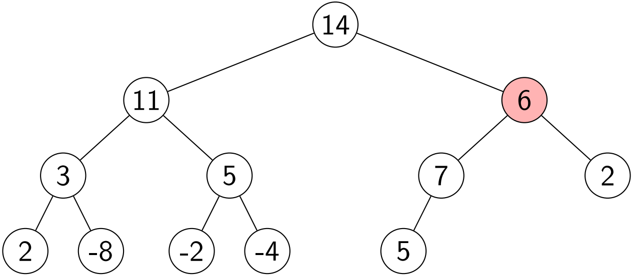
```
let rec up h i =
    let p = (i - 1)/2 in
    if i <> 0 && h.a.(p) < h.a.(i) then (
        swap h i p;
        up h p
    )
```

Complexité : $\mathrm{O}(h) = \mathrm{O}(\log(n))$.

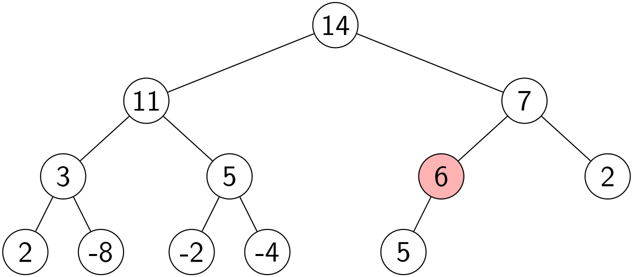
down



down



down



down

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

Complexité :

down

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

Complexité : $O(h) = O(\log(n))$.

Ajouter élément

Pour ajouter un élément (tant qu'il reste de la place dans le tableau) :

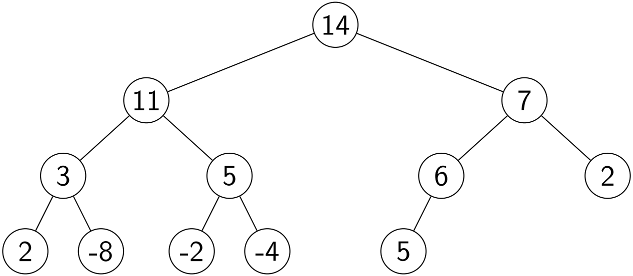
Ajouter élément

Pour ajouter un élément (tant qu'il reste de la place dans le tableau) :

1. l'ajouter en tant que feuille la plus à droite (dernier indice du tableau)
2. le faire remonter.

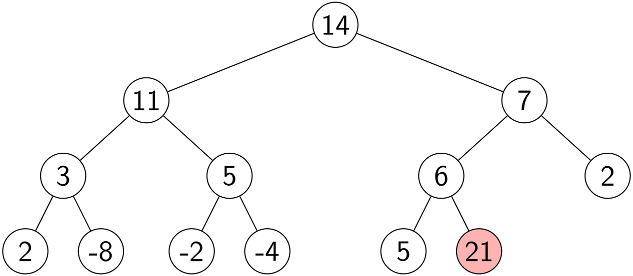
Ajouter élément

Insertion de 21 :



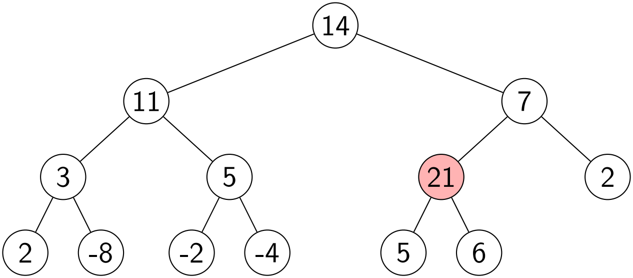
Ajouter élément

Insertion de 21 :



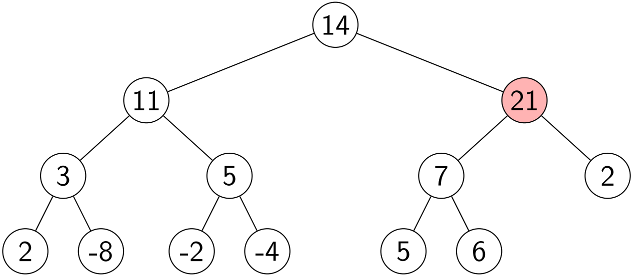
Ajouter élément

Insertion de 21 :



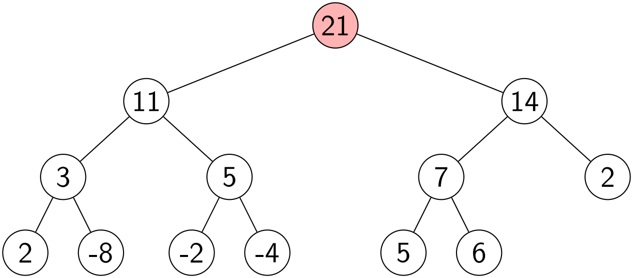
Ajouter élément

Insertion de 21 :



Ajouter élément

Insertion de 21 :



Ajouter élément

Code pour ajouter un élément :

```
let add h e =
    h.a.(h.n) <- e;
    up h h.n;
    h.n <- h.n + 1
```

Complexité :

Ajouter élément

Code pour ajouter un élément :

```
let add h e =
    h.a.(h.n) <- e;
    up h h.n;
    h.n <- h.n + 1
```

Complexité : $\mathsf{O}(h) = \mathsf{O}(\log(n))$.

Construire un tas à partir d'un tableau

Conversion d'un tableau quelconque en tas :

Construire un tas à partir d'un tableau

Conversion d'un tableau quelconque en tas :

```
let array_to_heap a =
    let h = { a=a; n=0 } in
    Array.iter (add h) a;
    h
```

Correction :

Construire un tas à partir d'un tableau

Conversion d'un tableau quelconque en tas :

```
let array_to_heap a =
    let h = { a=a; n=0 } in
    Array.iter (add h) a;
    h
```

Correction :
« au début de la boucle, les `i` premiers éléments de `tas.t` forment un tas » est un **invariant de boucle**.

Construire un tas à partir d'un tableau

Conversion d'un tableau quelconque (de taille n) en tas :

```
let array_to_heap a =
    let h = { a=a; n=0 } in
    Array.iter (add h) a;
    h
```

Complexité :

add est en

Construire un tas à partir d'un tableau

Conversion d'un tableau quelconque (de taille n) en tas :

```
let array_to_heap a =
    let h = { a=a; n=0 } in
    Array.iter (add h) a;
    h
```

Complexité :
add est en $O(\log(n))$ donc `array_to_tas` est en

Construire un tas à partir d'un tableau

Conversion d'un tableau quelconque (de taille n) en tas :

```
let array_to_heap a =
    let h = { a=a; n=0 } in
    Array.iter (add h) a;
    h
```

Complexité :
`add` est en $\mathsf{O}(\log(n))$ donc `array_to_tas` est en $\mathsf{O}(n \log(n))$.

Construire un tas à partir d'un tableau

Conversion d'un tableau quelconque (de taille n) en tas :

```
let array_to_heap a =
    let h = { a=a; n=0 } in
    Array.iter (add h) a;
    h
```

Complexité :
`add` est en O($\log(n)$) donc `array_to_tas` est en O($n \log(n)$).

Plus précisément : `add h a.(i)` est en O(p), où p est la profondeur de l'élément rajouté.

Construire un tas à partir d'un tableau

Conversion d'un tableau quelconque (de taille n) en tas :

```
let array_to_heap a =
    let h = { a=a; n=0 } in
    Array.iter (add h) a;
    h
```

Complexité plus précise :
Dans le pire des cas, chaque élément ajouté à une profondeur p est remonté en racine : p échanges.
Le nombre de swaps est donc, dans le pire cas :

$$\sum_{p=0}^{h} p2^p = ... = \Theta(h2^h) = \Theta(\log(n)n)$$

Extraire le maximum

On veut supprimer et renvoyer la racine, en conservant la structure de tas.

Extraire le maximum

On veut supprimer et renvoyer la racine, en conservant la structure de tas.

On peut :

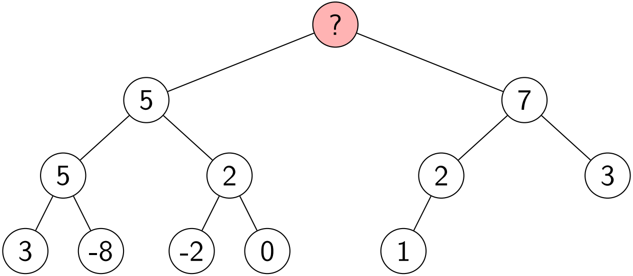
Extraire le maximum

On veut supprimer et renvoyer la racine, en conservant la structure de tas.

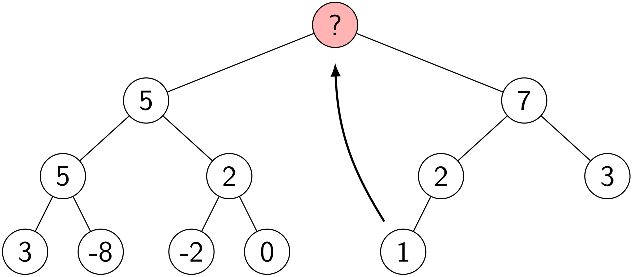
On peut :

1. Remplacer la racine par la dernière feuille.
2. Appeler down dessus.

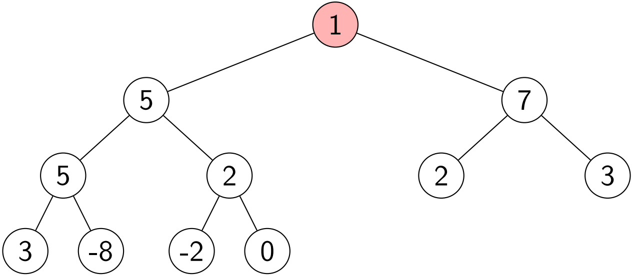
Extraire le maximum



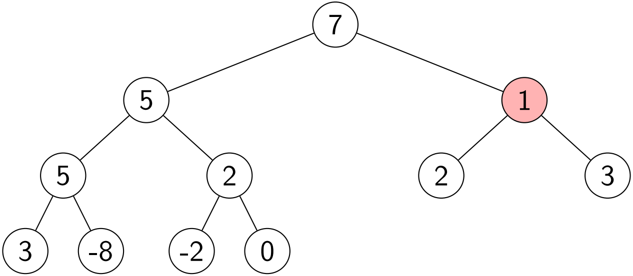
Extraire le maximum



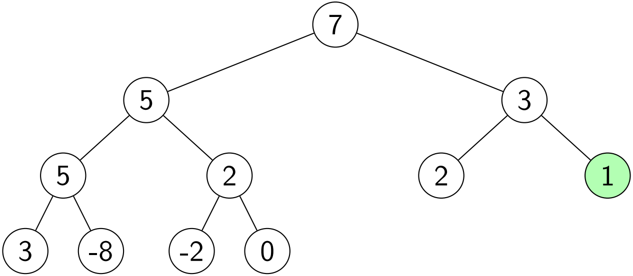
Extraire le maximum



Extraire le maximum



Extraire le maximum



Extraire le maximum

Code pour extraire la racine d'un tas :

Extraire le maximum

Code pour extraire la racine d'un tas :

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

Complexité :

Extraire le maximum

Code pour extraire la racine d'un tas :

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

Complexité : $O(\log(n))$.

Remarques :

- on met le maximum à la fin
- cette méthode ne permet que de supprimer la racine (maximum), pas un élément quelconque

Mettre à jour un élément

Pour mettre à jour un élément :

Mettre à jour un élément

Pour mettre à jour un élément :

1. Si on augmente son étiquette : on le monte.
2. Sinon : on le descend.

Mettre à jour un élément

Pour mettre à jour un élément :

Mettre à jour un élément

Pour mettre à jour un élément :

```
let update h i v =
    let p = h.a.(i) in
    h.a.(i) <- v;
    if v > p then up h i
    else down h i
```

Complexité :

Mettre à jour un élément

Pour mettre à jour un élément :

```
let update h i v =
    let p = h.a.(i) in
    h.a.(i) <- v;
    if v > p then up h i
    else down h i
```

Complexité : O($\log(n)$).

Tas max : résumé

Opération	Tas max
ajouter élément	O($\log(n)$)
extraire maximum	O($\log(n)$)
valeur du maximum	O(1)
mettre à jour	O($\log(n)$)
créer à partir d'un tableau de taille n	O(n)

File de priorité

Comparaison des implémentations de files de priorités :

Opération	Liste triée	Tas	ABR équilibré
ajouter	$\mathsf{O}(n)$	$\mathsf{O}(\log(n))$	$\mathsf{O}(\log(n))$
extraire max	$\mathsf{O}(1)$	$\mathsf{O}(\log(n))$	$\mathsf{O}(\log(n))$
valeur du max	$\mathsf{O}(1)$	$\mathsf{O}(1)$	$\mathsf{O}(\log(n))$
`update`	$\mathsf{O}(n)$	$\mathsf{O}(\log(n))$	$\mathsf{O}(\log(n))$
Conversion depuis **`array`**	$\mathsf{O}(n\log(n))$	$\mathsf{O}(n)$	$\mathsf{O}(n\log(n))$

Tri par tas

Toute FP avec ajout et extraction du maximum en $\mathrm{O}(f(n))$ donne un algorithme de tri en $\mathrm{O}(nf(n))$: on ajoute un à un les éléments extraits dans une nouvelle liste.

Tri par tas

Toute FP avec ajout et extraction du maximum en $O(f(n))$ donne un algorithme de tri en $O(nf(n))$: on ajoute un à un les éléments extraits dans une nouvelle liste.

1. FP implémenté avec tas $\implies$ tri en $O(n \log(n))$
2. FP implémenté avec AVL/ARN $\implies$ tri en $O(n \log(n))$
3. ...

Mais avec un tas on peut éviter de créer un nouveau tableau (complexité $O(1)$ **en mémoire**).

Tri par tas

Code pour trier avec un tas :

```
let heap_sort a =
    let h = array_to_heap a in
    for i = 0 to h.n - 1 do
        extract_max h
    done
```

Correction : « au début de la boucle, les éléments de `h` d'indices $n - i$ à $n - 1$ sont les i plus grands éléments triés ».

Tri par tas

Code pour trier avec un tas :

```
let heap_sort a =
    let h = array_to_heap a in
    for i = 0 to h.n - 1 do
        extract_max h
    done
```

Complexité : $\mathsf{O}(n + n\log(n)) = \mathsf{O}(n\log(n))$ (optimal pour un tri).

Tri par tas

Code pour trier avec un tas :

```
let heap_sort a =
    let h = array_to_heap a in
    for i = 0 to h.n - 1 do
        extract_max h
    done
```

Complexité en mémoire (espace utilisé en plus de l'entrée) : $O(1)$
On dit que le tri est **en place** : pas besoin de créer un nouveau tableau.

Tri par tas

Exercice

Comment trier partiellement un tableau (seulement les k plus grands ou plus petits) ?

Tri par tas

Exercice

Comment trier partiellement un tableau (seulement les k plus grands ou plus petits) ?

Il suffit d'arrêter la boucle au bout de k itérations.
<u>Complexité</u> :

Tri par tas

Exercice

Comment trier partiellement un tableau (seulement les k plus grands ou plus petits) ?

Il suffit d'arrêter la boucle au bout de k itérations.
Complexité : $\mathrm{O}(n + k\log(n))$.

Ceci donne un algorithme linéaire pour trouver le $k$ème plus petit élément d'un tableau, pour $k \leq \dfrac{n}{\log(n)}$.