Files de priorité

MPCI Spécialité informatique

Stéphane Grandcolas

Aix-Marseille Université

Contact : stephane.grandcolas@univ-amu.fr

Files de priorité

Structure de donnée opérant sur une ensemble dont les éléments sont marqués par des priorités, et qui implémente les opérations :

- ajouter un nouvel élément,
- trouver un élément de priorité maximum,
- supprimer un élément de priorité maximum,
- augmenter/diminuer la priorité (increase-key).

priorité : clé, avec une relation d'ordre sur les clés.

Par exemple : $(\mathbb{N}, \leq)$. Souvent l'élément le plus prioritaire est celui qui a la clé la plus petite (extract-min, decrease-key)

Files de priorité

	ajout	trouver min	supprimer min
tableau[1]	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
tableau ordonné[2]	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
liste chaînée	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$[3]
liste chaînée ordonnée	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

[1] ou tableau redimensionnable
[2] i.e. ordonnés par priorités croissantes
[3] en supposant qu'on sait où il est

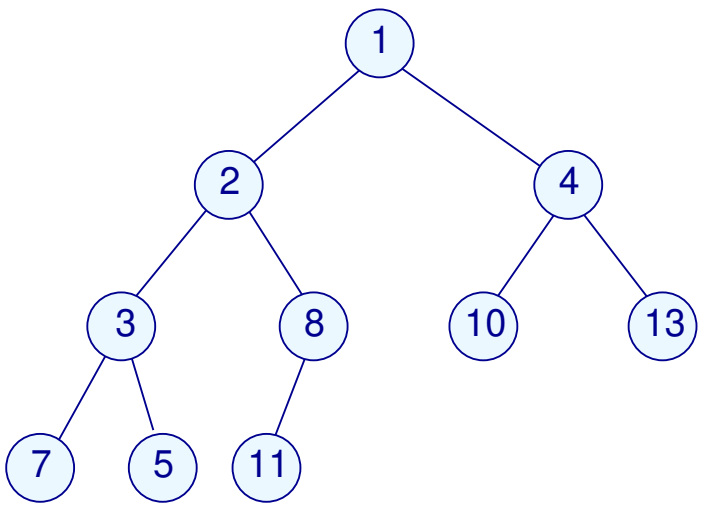
Files de priorité

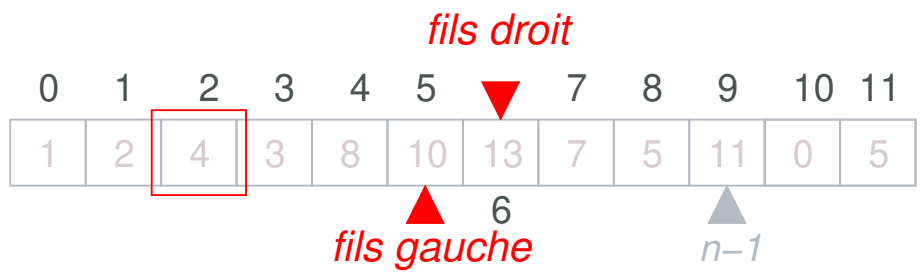
	ajout	trouver min	supprimer min
tableau	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
tableau ordonné	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
liste chainée	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$
liste chainée ordonnée	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
tas binaire	$\mathcal{O}(\log\ n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log\ n)$
tas de Fibonacci[1]	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log\ n)$

[1] coûts amortis

Tas binaire (*heap*)

Arbre binaire représenté dans un tableau satisfaisant la *propriété des tas* : le père a une priorité plus forte que ses fils





(seules les clés sont représentées, la priorité est donnée à l'élément qui a la plus petite clé)

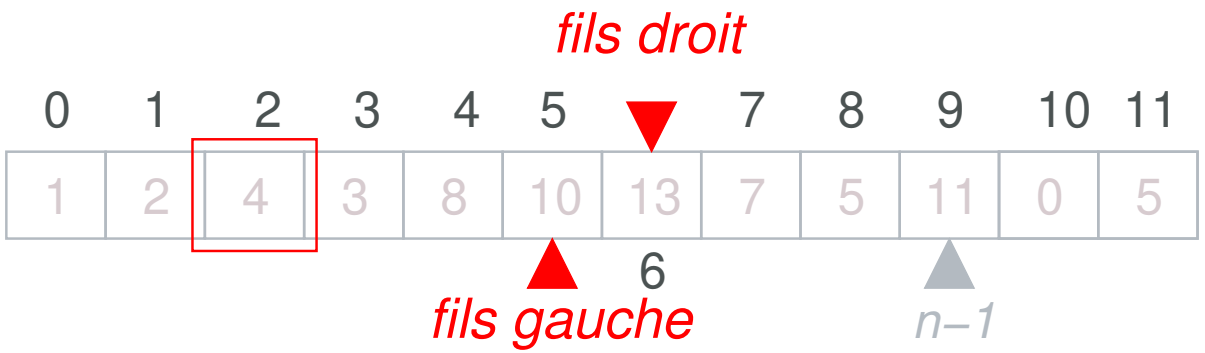
Tas binaire : représentation

Tableau indexé : les n éléments du tas sont $T[0]$, $T[1]$,..., $T[n-1]$.

Convention :

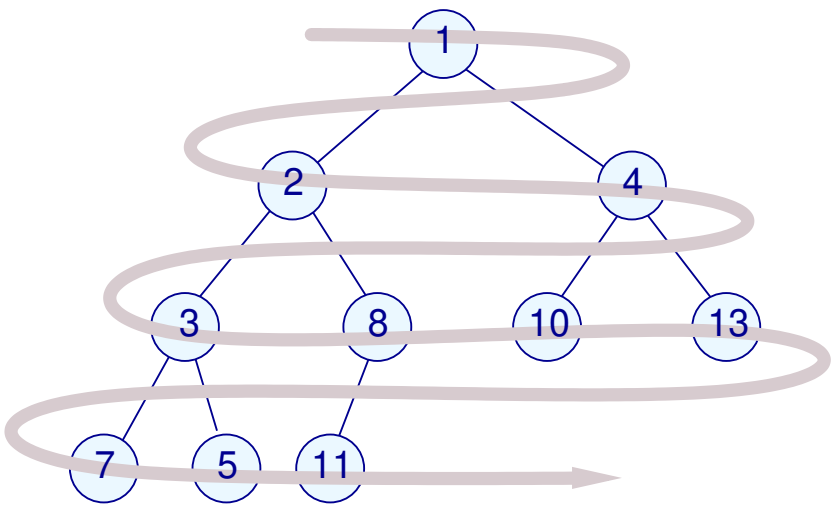
- indice du fils gauche de $T[i]$: $2 \times i + 1$,
- indice du fils droit de $T[i]$: $2 \times i + 2$.

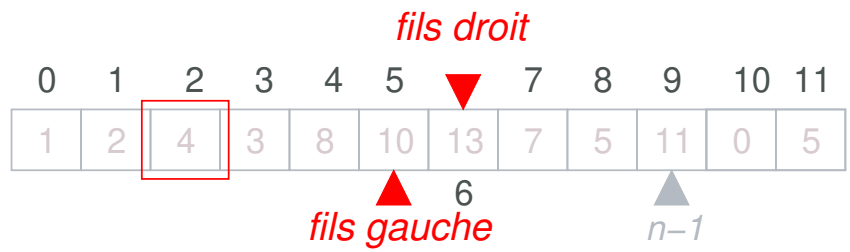
L'arbre est *naturellement* équilibré (les feuilles sont toutes à la même profondeur à un niveau près, les plus longues branches sont à gauche)



Tas binaire : représentation

Le parcours en largeur de l'arbre de gauche à droite correspond au parcours des éléments du tableau dans l'ordre des indices.



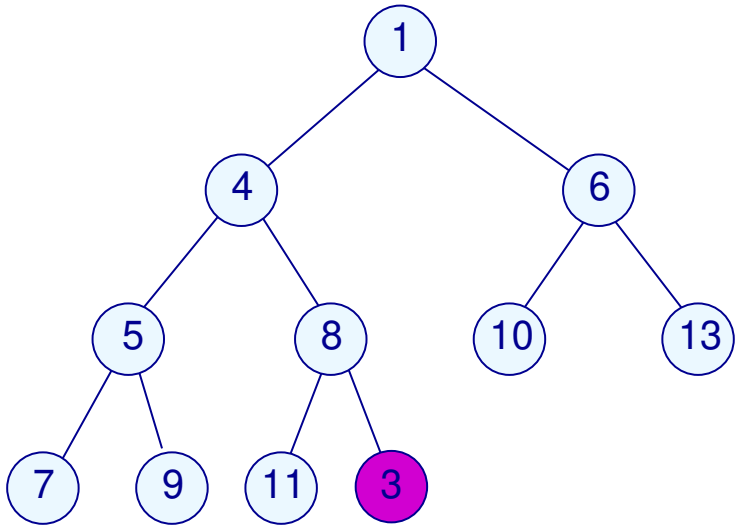


Tas binaire : opérations

- chercher le min (l'élément le plus prioritaire)
- ajouter un nouvel élément
- extraire le min
- diminuer la clé d'un élément donné
- construire un tas

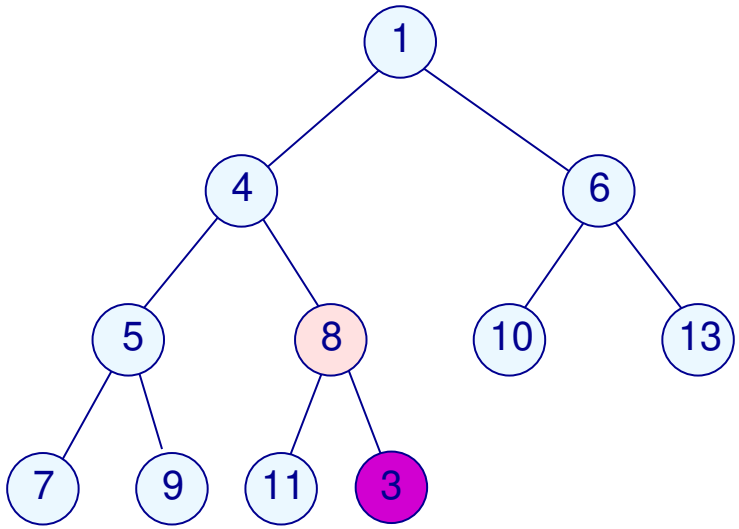
Tas binaire : ajout d'un nouvel élément

- placer le nouvel élément à la fin du tableau



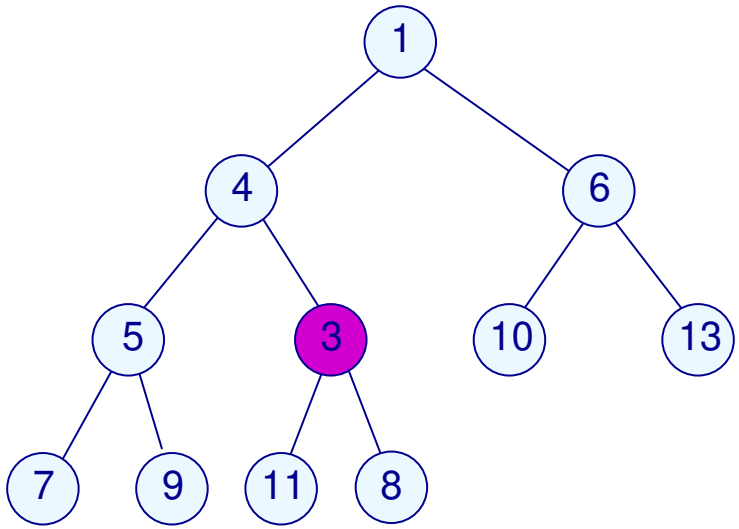
Tas binaire : ajout d'un nouvel élément

- (1) comparer avec le père



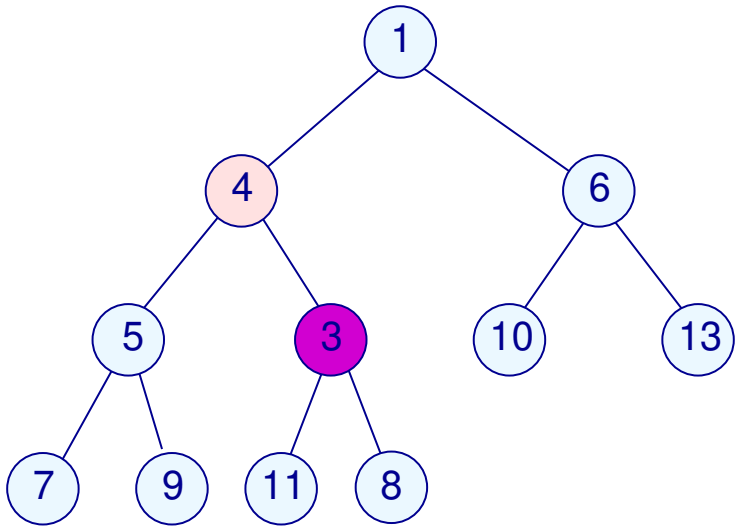
Tas binaire : ajout d'un nouvel élément

- (2) permuter avec le père si besoin



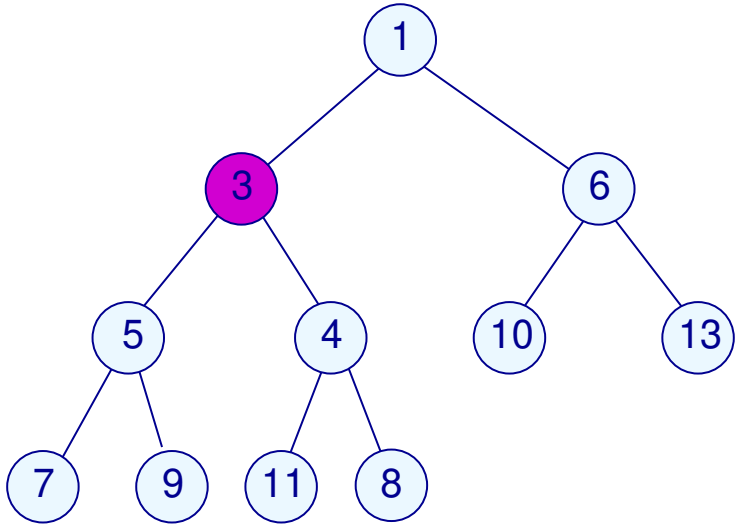
Tas binaire : ajout d'un nouvel élément

- réitérer : (1) comparer avec le père



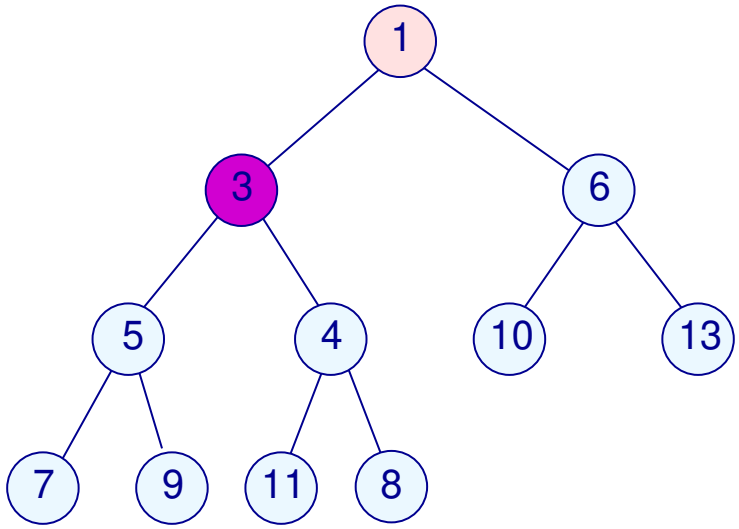
Tas binaire : ajout d'un nouvel élément

- (2) permuter si besoin



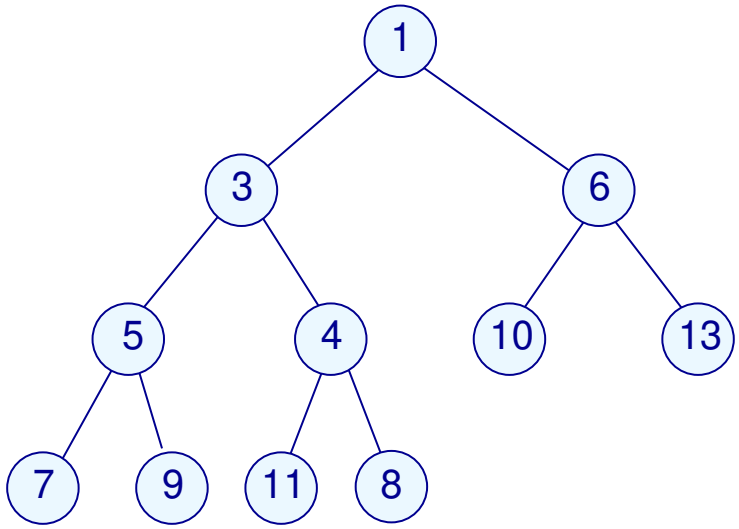
Tas binaire : ajout d'un nouvel élément

- comparer avec le père



Tas binaire : ajout d'un nouvel élément

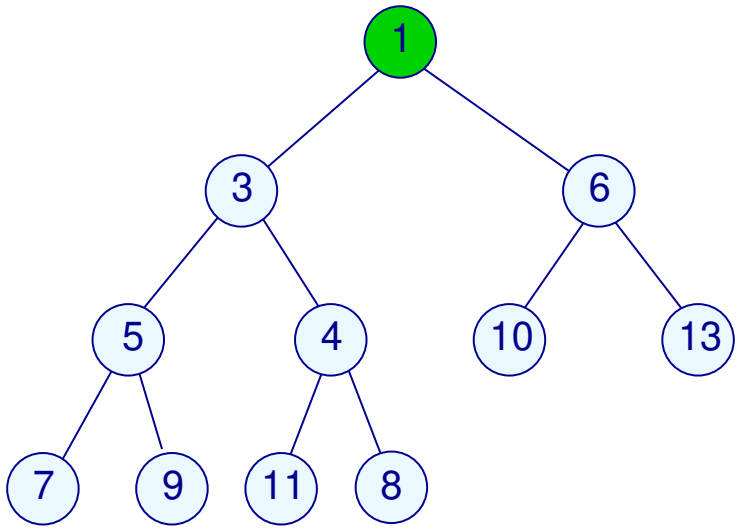
- stoppe dès que la propriété des tas est rétablie



Coût $\mathcal{O}(\log\ n)$, la hauteur de l'arbre

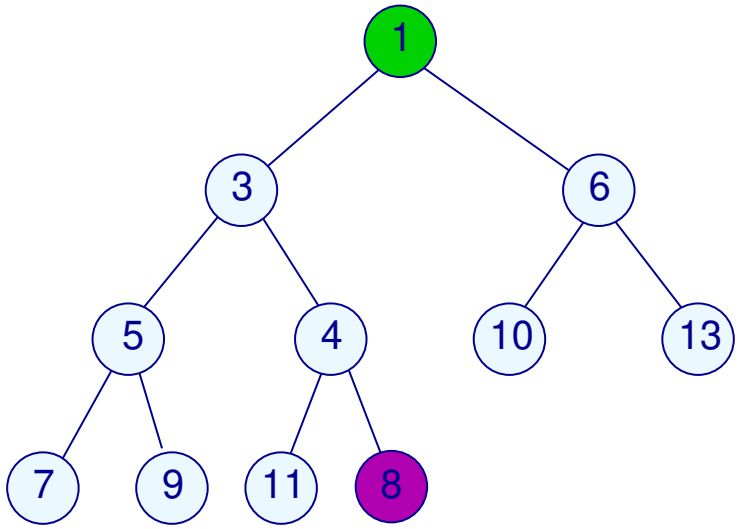
Tas binaire : extraction du min

- l'élément minimal est à la racine (indice 0 du tableau)



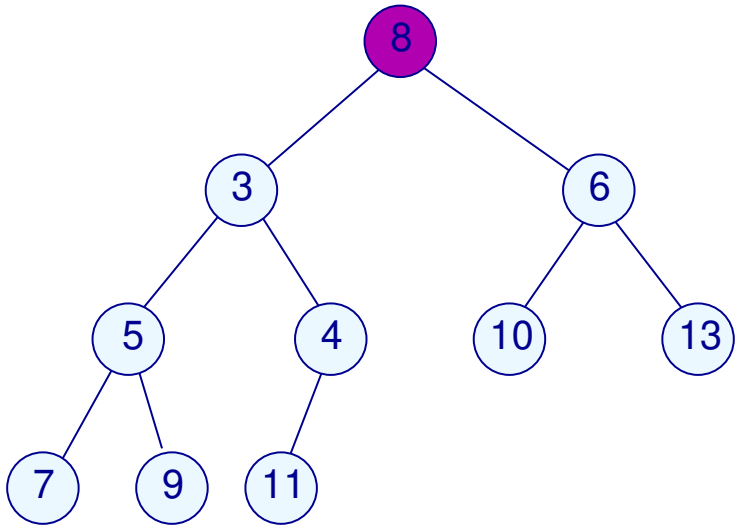
Tas binaire : extraction du min

- permuter avec le dernier élément du tableau



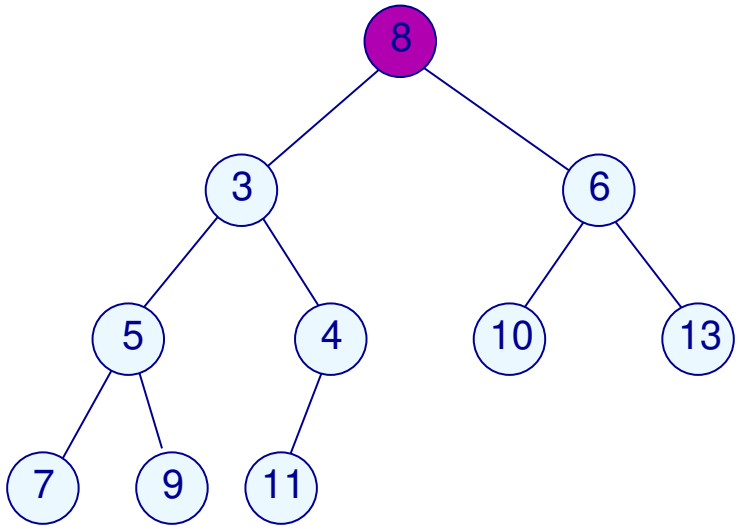
Tas binaire : extraction du min

- supprimer le dernier



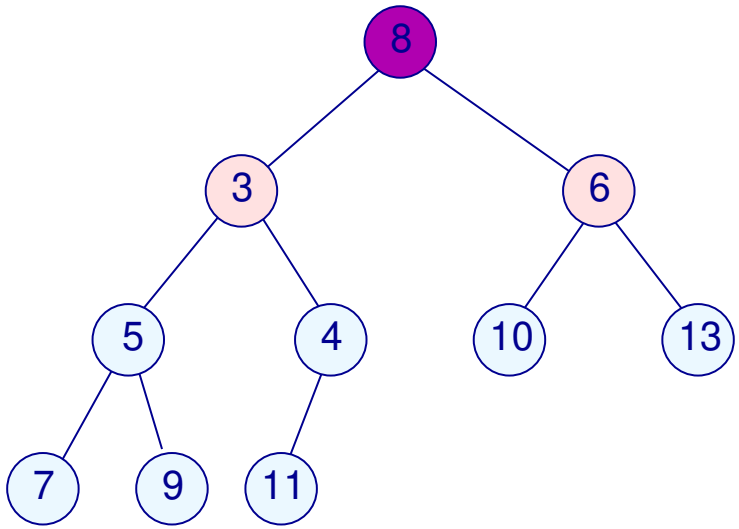
Tas binaire : extraction du min

- rétablir la propriété des tas : *entasser*, *heapify*



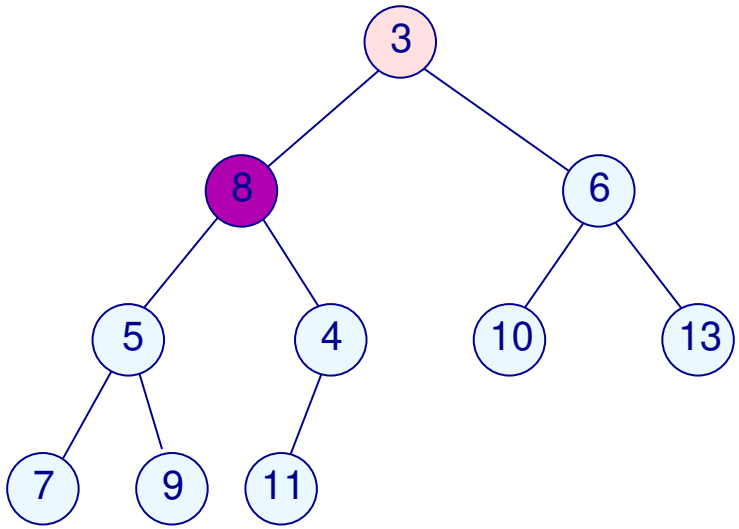
Tas binaire : extraction du min

- (1) comparer avec les fils



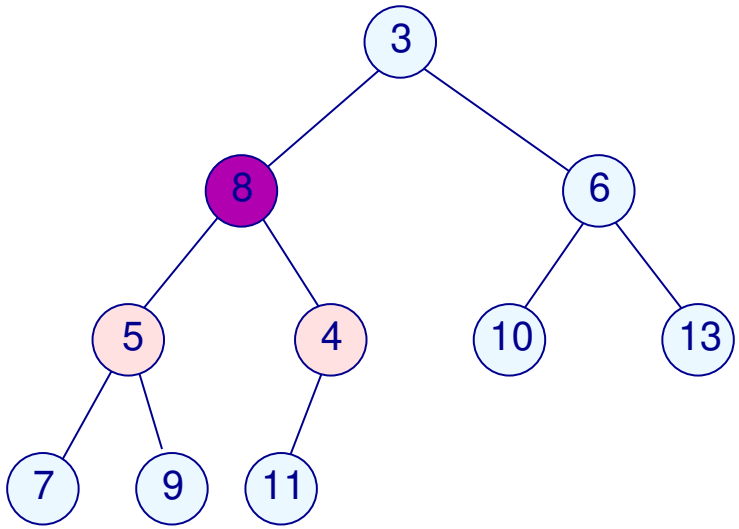
Tas binaire : extraction du min

- (2) permuter avec le plus prioritaire



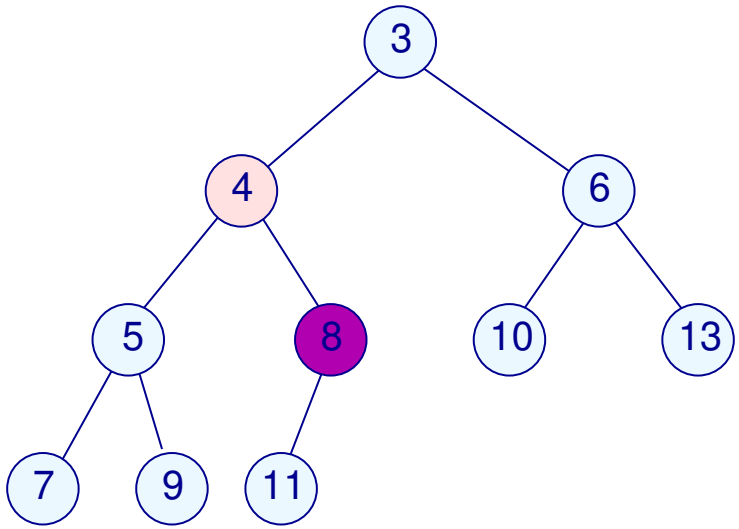
Tas binaire : extraction du min

- réitérer : (1) comparer avec les fils



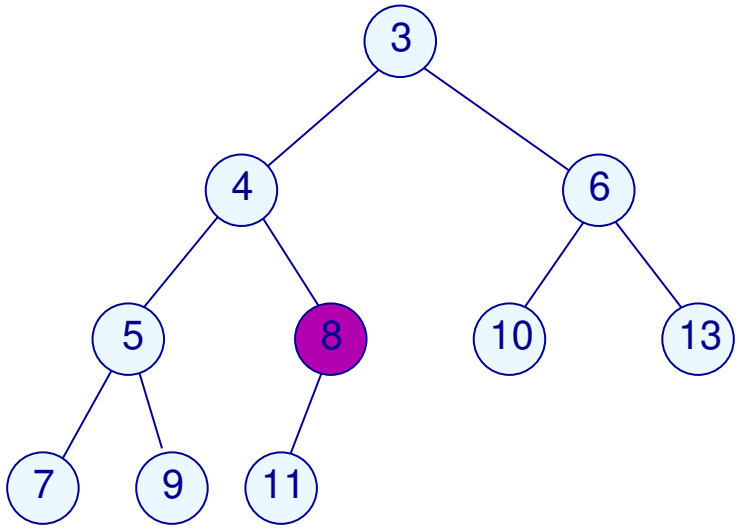
Tas binaire : extraction du min

- (2) permuter avec le plus prioritaire



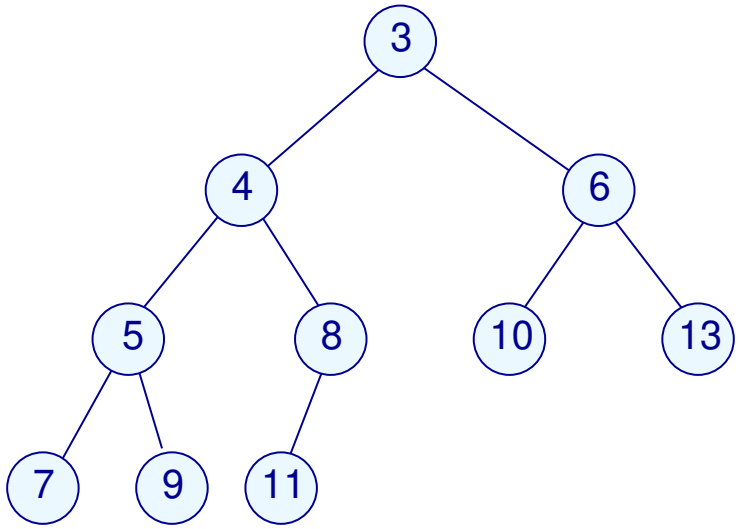
Tas binaire : extraction du min

- stoppe dès que la propriété des tas est rétablie



Tas binaire : extraction du min

- *entasser*, *heapify*



Coût $\mathcal{O}(\log\, n)$, la hauteur de l'arbre

Tas binaire : entasser

fonction ENTASSER(i, T, n)

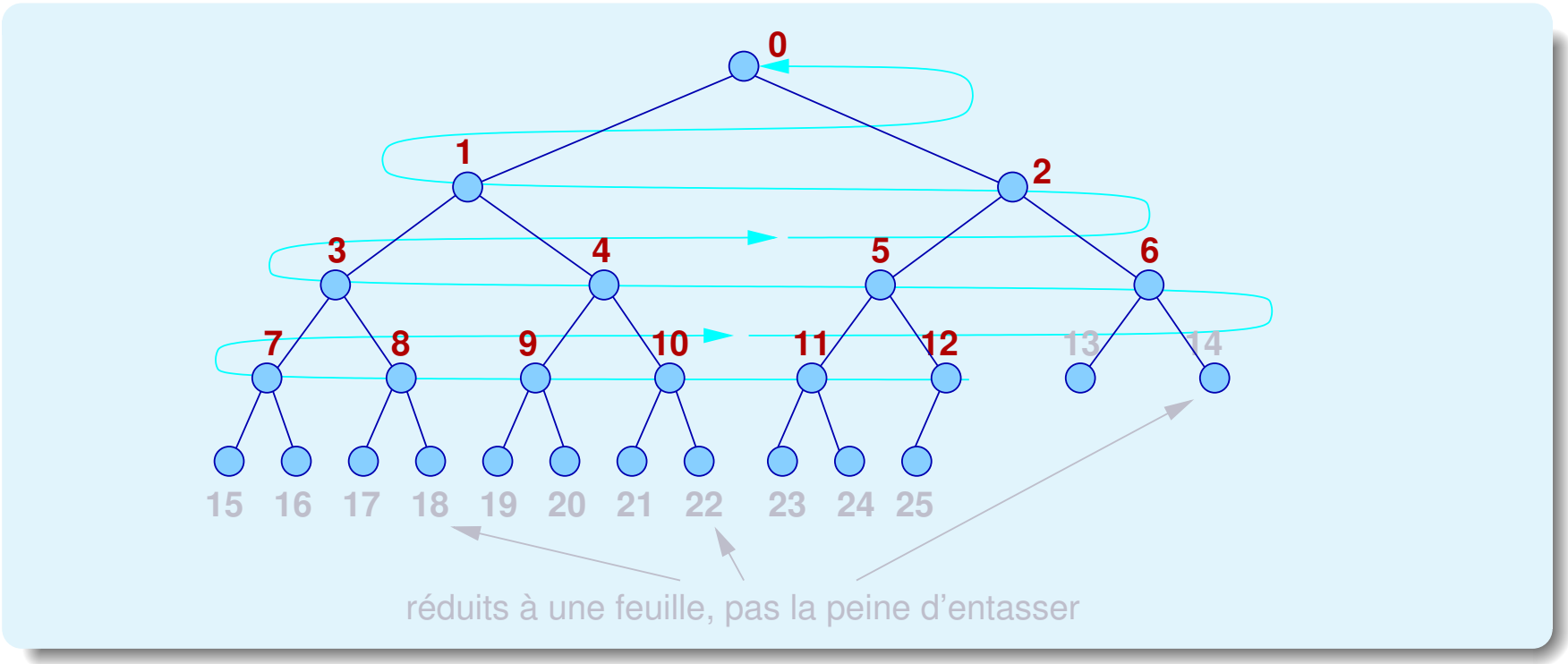
Entrées : i indice, T tableau contenant n éléments
Avant : $\mathrm{FilsG}(i)$ et $\mathrm{FilsD}(i)$ sont des tas
Après : i est un tas

i_{min} := indice du plus prioritaire entre i et ses fils,
si $i_{min} \neq i$ **alors**
    PERMUTER(i, i_{min}, T),
    ENTASSER(i_{min}, T, n),
fin si
fin fonction

Complexité : $\mathcal{O}(\log\ n)$

Tas binaire : construction

Principe : établir la propriété de tas en partant du bas



- initialement tous les éléments sont dans le tableau
- appliquer ENTASSER

Tas binaire : construction

Etat initial : les $n/2$ derniers éléments du tableau sont des tas (feuilles)
Itération : $\mathrm{FilsG}(i)$ et $\mathrm{FilsD}(i)$ sont des tas, après Entasser(i, T, n), l'arbre enraciné en i est un tas
Etat final : l'arbre enraciné en 0 est un tas

fonction CONSTRUIRE_TAS(T, n)
    pour $i := n/2 - 1$ **jusqu'à** 0 **faire**
        ENTASSER(i, T, n),
fin fonction

Complexité : $\mathcal{O}(n \log n)$ de façon évidente, en fait $\mathcal{O}(n)$

Tas binaire : construction (preuve $\mathcal{O}(n)$)

- à la hauteur $h = 0$ au plus $\lceil \frac{n}{2^1} \rceil$ noeuds
- à la hauteur $h = 1$ au plus $\lceil \frac{n}{2^2} \rceil$ noeuds
- à la hauteur h au plus $\lceil \frac{n}{2^{h+1}} \rceil$ noeuds

Entasser à la hauteur h requiert au plus h permutations

Tas binaire : construction (preuve $\mathcal{O}(n)$)

nombre de permutations

$$n_{perm}(n) \leq \sum_{h=1}^{\log\ n} \left\lceil \frac{n}{2^{h+1}} \right\rceil \times h \leq n \times \sum_{h=1}^{\log\ n} \frac{h}{2^h}$$

or

$$\sum_{i=1}^{\infty} i \times \left(\frac{1}{2}\right)^i = \frac{\sum_{i=1}^{\infty} \left(\frac{1}{2}\right)^i}{1 - 1/2} \simeq 2$$

et donc

$$n_{perm}(n) \leq n \times \sum_{h=0}^{\infty} \frac{h}{2^h} \leq 2n = \mathcal{O}(n)$$

Tas binaire : diminuer la clé

- trouver l'élément dans le tas,
- faire *remonter* l'élément dans le tas jusqu'à ce que la propriété des tas soit rétablie (comme pour l'insertion)

Coût : $\mathcal{O}(\log\, n)$

Remarque. Suppose qu'on peut trouver l'élément dans le tas en temps constant, sinon l'opération est très couteuse. On peut par exemple adjoindre à chaque élément sa position dans le tas, position qui sera mise à jour après chaque permutation.

Tas binaire : coûts des opérations

opération	*coût*
trouver le min	$\mathcal{O}(1)$
ajouter	$\mathcal{O}(\log\ n)$
supprimer le min	$\mathcal{O}(\log\ n)$
diminuer la clé	$\mathcal{O}(\log\ n)$
construire un tas	$\mathcal{O}(n)$

Tas binaire : coûts des opérations

opération	*coût*
trouver le min	$\mathcal{O}(1)$
ajouter	$\mathcal{O}(\log\ n)$
supprimer le min	$\mathcal{O}(\log\ n)$
diminuer la clé	$\mathcal{O}(\log\ n)$
construire un tas	$\mathcal{O}(n)$

Tas de Fibonacci

opération	*coût tas binaire*	*coût tas de Fibonacci*
trouver le min	$\mathcal{O}(1)$	$\mathcal{O}(1)$
ajouter	$\mathcal{O}(\log\ n)$	$\mathcal{O}(1)$
extraire le min	$\mathcal{O}(\log\ n)$	$\mathcal{O}(\log\ n)$
diminuer la clé	$\mathcal{O}(\log\ n)$	$\mathcal{O}(1)$
union	$\mathcal{O}(n)$	$\mathcal{O}(1)$