File de priorité / tas binaire

File de priorité

Type abstrait d'une **file de priorité**/file à priorités (*priority queue*)

Objets : ensembles d'objets avec clés comparables (abstraction : nombres naturels)

Opérations :
insert(x) ou add : insertion (addition) de l'élément x
deleteMin() : supprime et retourne l'élément de valeur minimale

Opérations parfois supportées :
merge : fusionner deux files
findMin : retourne (mais ne supprime pas) l'élément minimal ("*peek*")
decreaseKey/increaseKey : ajustememnt de priorité d'un élément

(ou définitions équivalentes avec deleteMax et findMax — mais non pas max et min en même temps)

Applications

- $\star$ simulations d'événements discrets
  (p.e., collisions, mutations génétiques, épidémiques) :
  priorité = ordonnancement temporaire
- $\star$ systèmes d'exploitation (partage multiple du CPU et d'autres ressources :
  interruptions entre procéssus multiples, ordonnancement de tâches) :
  priorité = spécification explicite
- $\star$ algorithmes sur graphes, recherche opérationnelle
  (plus courts chemins, arbre couvrant minimal)
- $\star$ statistiques : maintenir l'ensemble des m meilleurs éléments dans un algorithme
  en-ligne

```
MEILLEUR-EMTS(T[0..n − 1], m)
                                   // (maintient les m plus grand éléments
B1  initialiser min-tas PQ
B2  for i ← 0, . . . , n − 1 do PQ.insert(T[i]) ; if i ≥ m then PQ.deleteMin()
B3  return les éléments de PQ
```

Implémentations pour quelques éléments

⋆ tableau / liste chaînée avec des éléments non-ordonnés — approche paresseuse

Opération insert(x) *// en* $O(1)$

I1 head.insertNext(x) *// (insertion à la tête)*

Opération deleteMin() *// en* $\Theta(n)$ *au pire*

D1 $N \leftarrow$ head; M $\leftarrow$ null; $v \leftarrow \infty$

D2 **while** N.next $\neq$ null

D3 $w \leftarrow N$.next.info

D4 **if** $w < v$

D5 **then** $v \leftarrow w$; $M \leftarrow N$ *//(M contient min)*

D6 $N \leftarrow N$.next

D7 M.deleteNext() *//(suppression après nœud M)*

D8 **return** v

⋆ tableau / liste chaînée avec des éléments ordonnés — approche impatiente

insert en $O(n)$, deleteMin en $O(1)$

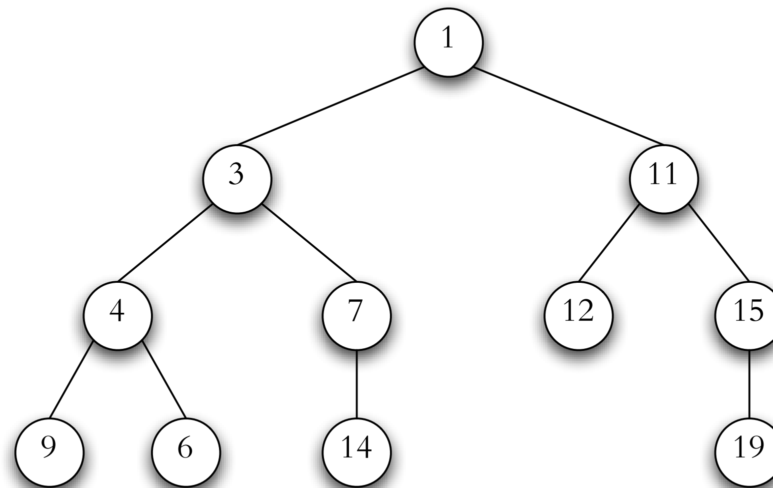
On veut une meilleure solution. . .

Ordre de tas

arbre dans l'**ordre de tas** : si nœud x n'est pas la racine, alors

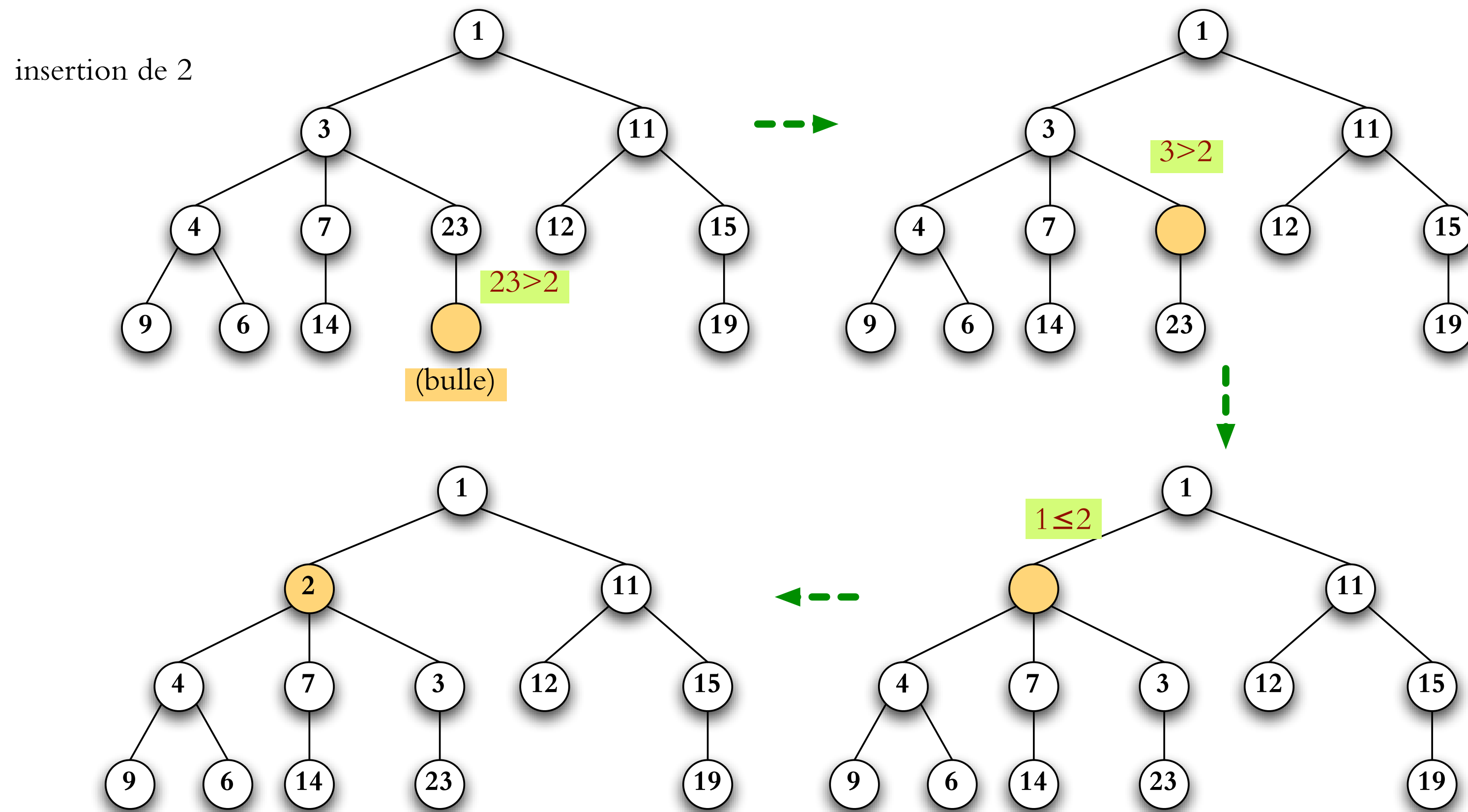
$$x.\mathsf{parent.priorite} \leq x.\mathsf{priorite}.$$

Opération findMin en $O(1)$: c'est à la racine.



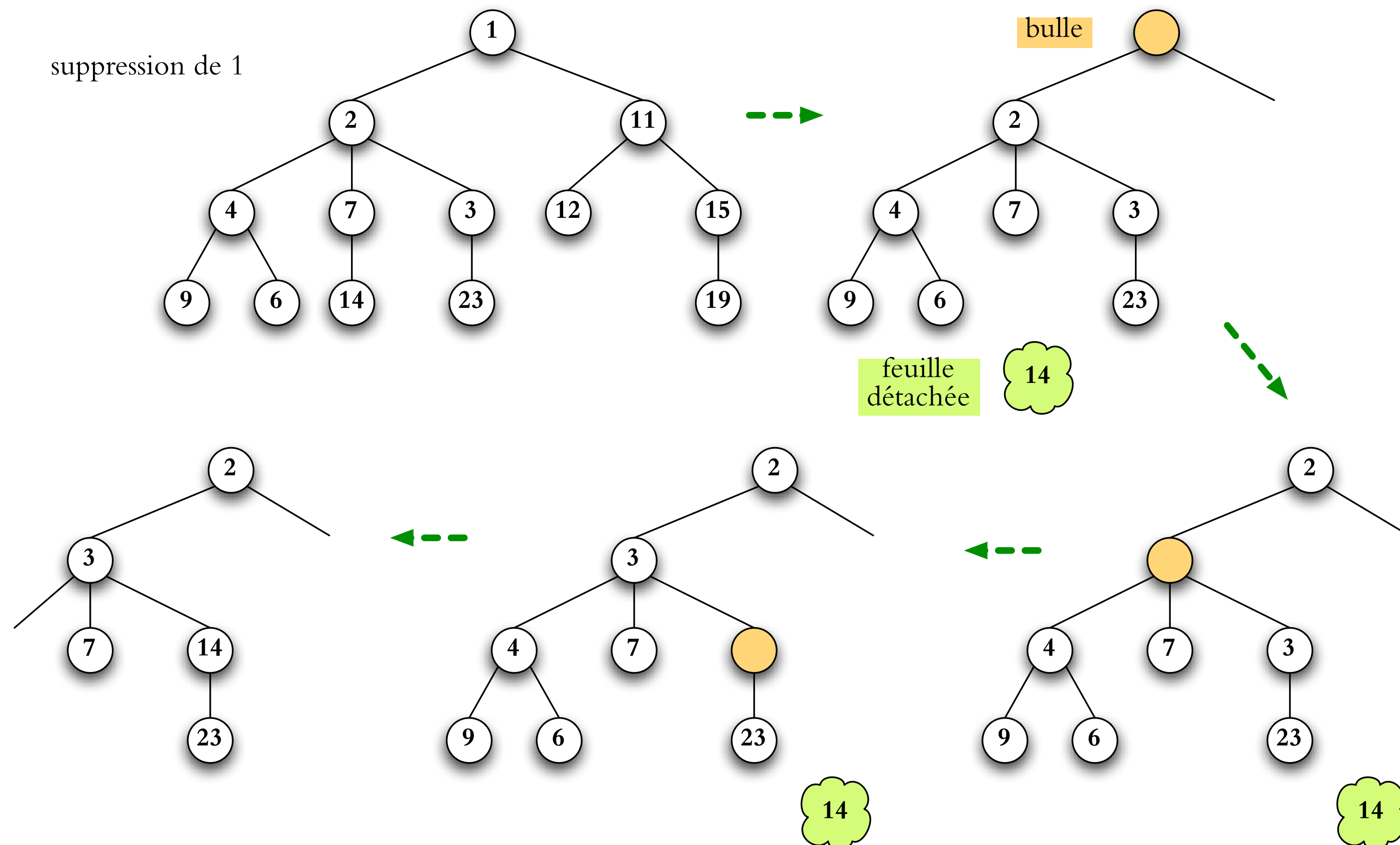
Insertion

ajouter une feuille vide («bulle») + monter la bulle vers la racine jusqu'à ce qu'on trouve la place pour la nouvelle valeur (**swim**/nager)



Suppression

remplacer le nœud par une «bulle», enlever une feuille et pousser la bulle vers les feuilles jusqu'à ce qu'on trouve la place pour la nouvelle valeur (**sink**/couler)

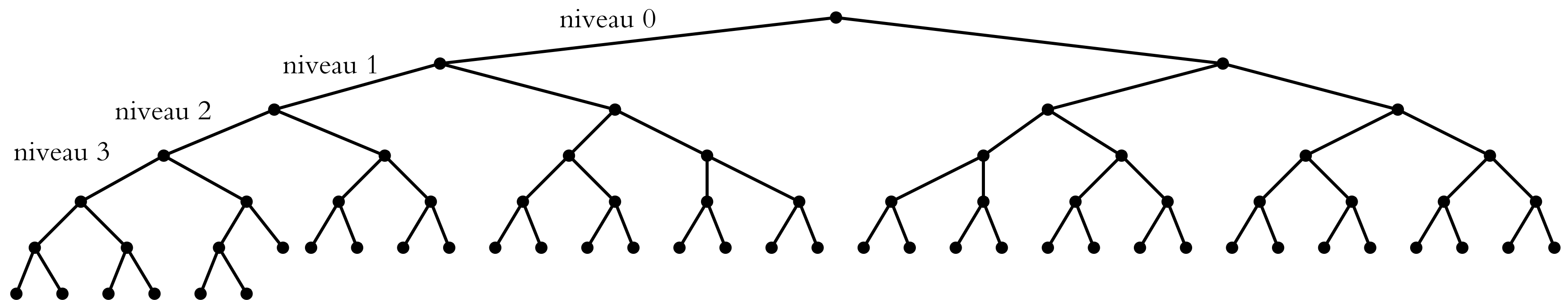


Tas — efficacité

Temps pour insertion : dépend du niveau où on crée la bulle

Temps pour suppression : dépend du nombre des enfants des nœuds échangés avec la bulle

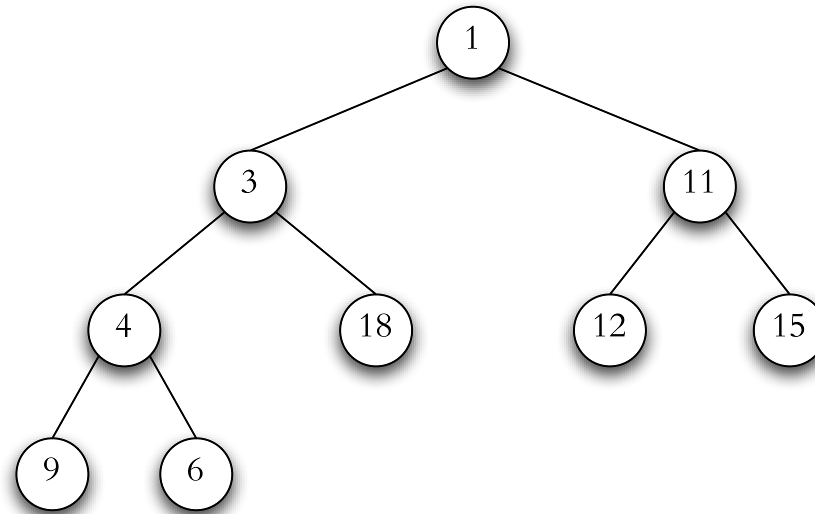
Hauteur minimale pour n éléments atteinte par **arbre binaire complet** :
il y a 2^i nœuds à chaque niveau $i = 0, \ldots, h-1$; les niveaux «remplis» de gauche à droit



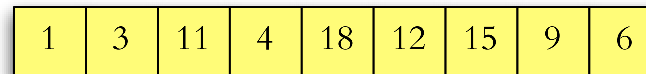
Tas binaire

Arbre binaire complet $\rightarrow$ pas de pointeurs parent, left, right !

Tableau $H[1..n]$, enfant gauche est à $2i$, enfant droit est à $2i+1$, parent de nœud i à $\lceil (i-1)/2 \rceil$.
ordre de tas : $H[i] \leq H[2i], H[2i+1]$.



indice dans le tableau: 1 2 3 4 5 6 7 8 9



niveau: 0 1 2 3

Tas binaire : insertion et suppression

```
    INSERT(v, H, n)                                   // en O(lg n)
I1  SWIM(v, n+1, H)                    // // tas binaire dans H[1..n]

    SWIM(v, i, H)                         // placement de v en H[1..i]
N1  p ← ⌊i/2⌋
N2  while p ≠ 0 et H[p] > v do H[i] ← H[p] ; i ← p ; p ← ⌊i/2⌋
N3  H[i] ← v
```

```
    DELETEMIN(H, n)                                   // en O(lg n)
D1  r ← H[1]                                    // tas dans H[1..n]
D2  v ← H[n] ; H[n] ← null ; if n > 1 then SINK(v, 1, H, n − 1)
D3  retourner r
```

Tas binaire : sink