Files de priorité

File de priorité

Type abstrait d'une **file de priorité** ou **tas** ou **monceau**

(*priority queue* & *heap*)

Objets : ensembles de valeurs naturelles (abstraction de clés comparables)

Opérations :
$\mathsf{insert}(x, H)$: insertion de l'élément x dans H
$\mathsf{deleteMin}(H)$: enlever l'élément de valeur minimale dans H

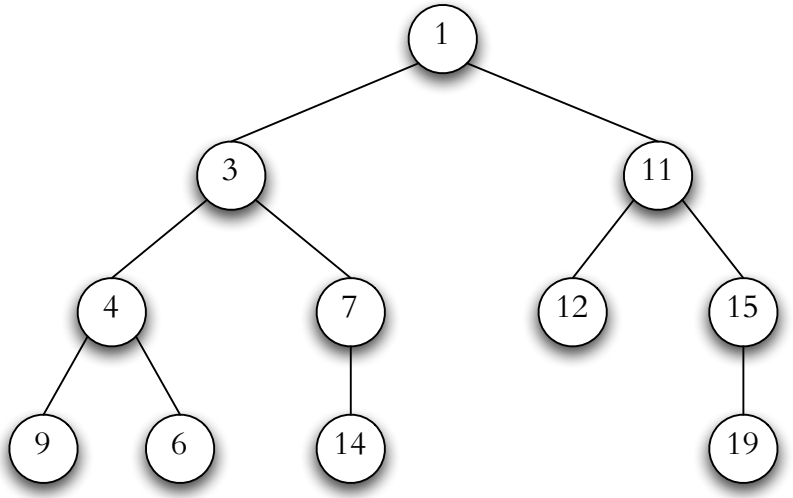
Opérations parfois supportées :
$\mathsf{merge}(H_1, H_2)$: fusionner deux files
$\mathsf{findMin}(H)$: retourne (mais ne supprime pas) l'élément minimal
$\mathsf{delete}(x, H)$: supprimer élément x

Notez qu'un arbre équilibré peut offrir toutes ces fonctionnalités en $O(\log n)$

Monceau

on va implanter la file de priorité par une arborescence dont les nœuds sont dans l'**ordre de monceau** :

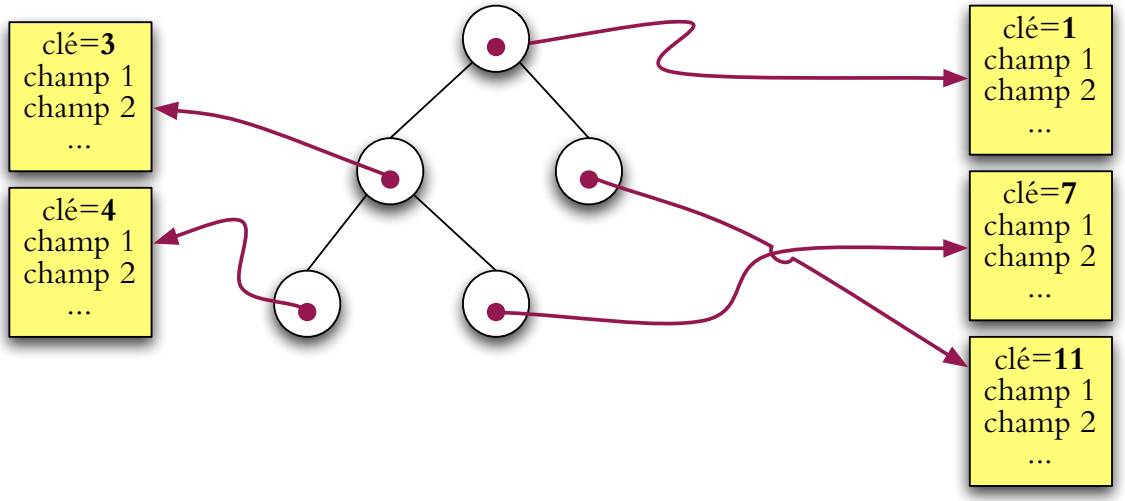
si x n'est pas la racine, alors $\mathsf{val}\Big(\mathsf{parent}(x)\Big) \leq \mathsf{val}(x)$.



Opération findMin en $O(1)$: retourner val(racine)

Monceau (cont)

Les valeurs ne sont pas stockées avec les nœuds mais plutôt un pointeur vers les données associées (en Java, il n'y a pas de grande différence : `val` est un objet `Comparable`)

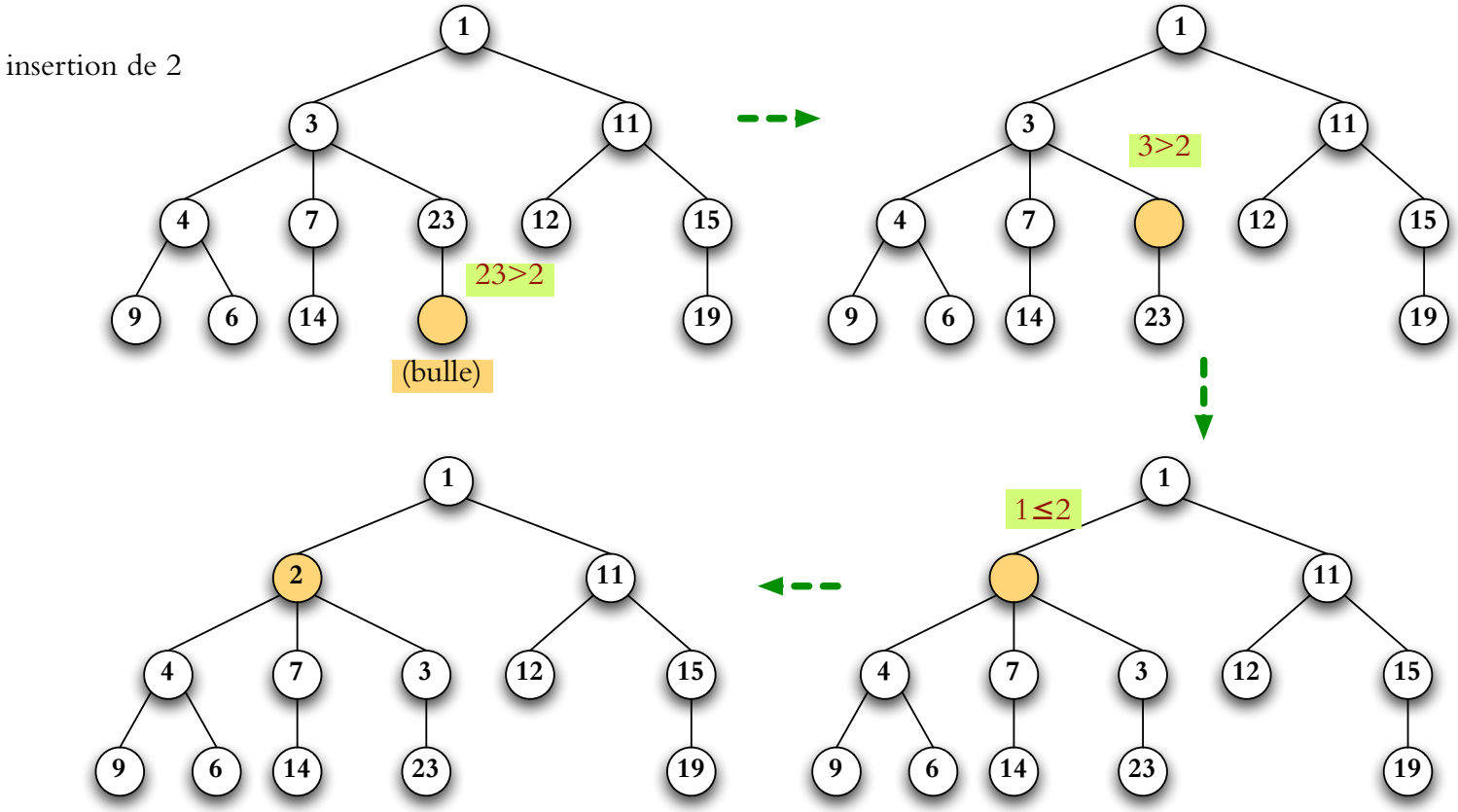


Comment insérer et supprimer ?

Idée de base : on ne change pas la structure de l'arbre
- affectation de pointeurs de données seulement

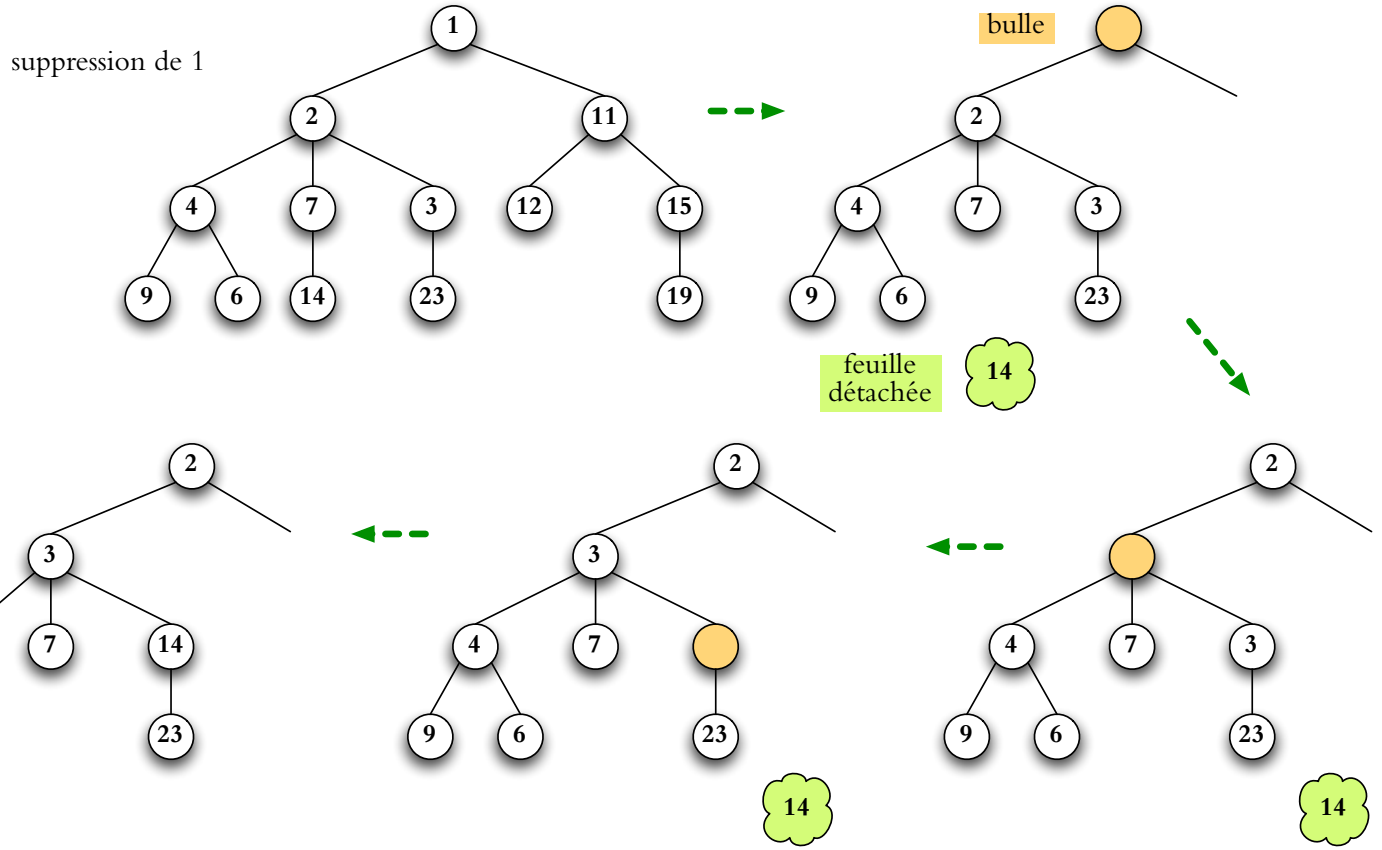
Monceau — insertion

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



Monceau — 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

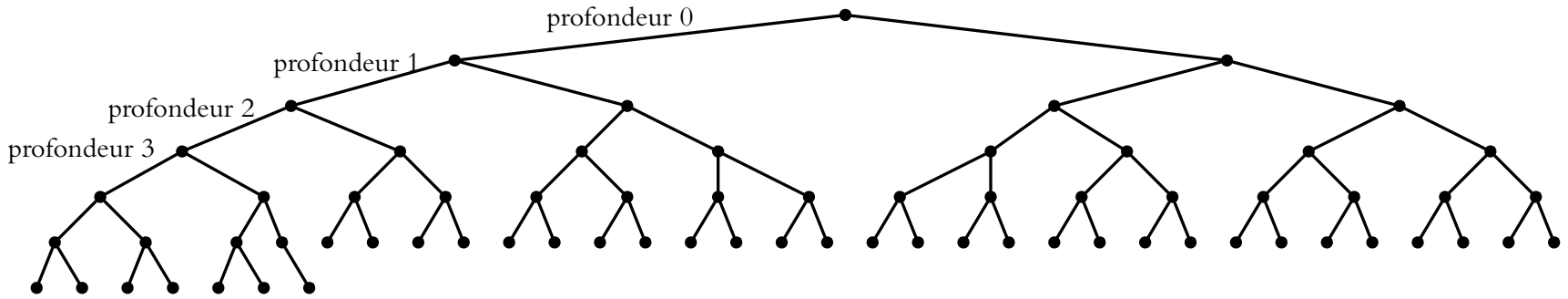


Monceau — efficacité

Temps pour insertion : dépend de la profondeur où on crée la bulle

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

Une solution simple : utiliser un **arbre binaire complet** de hauteur h :
il y a 2^i nœuds de chaque profondeur $i = 0, \ldots, h-1$; les niveaux sont «remplis» de gauche à droit

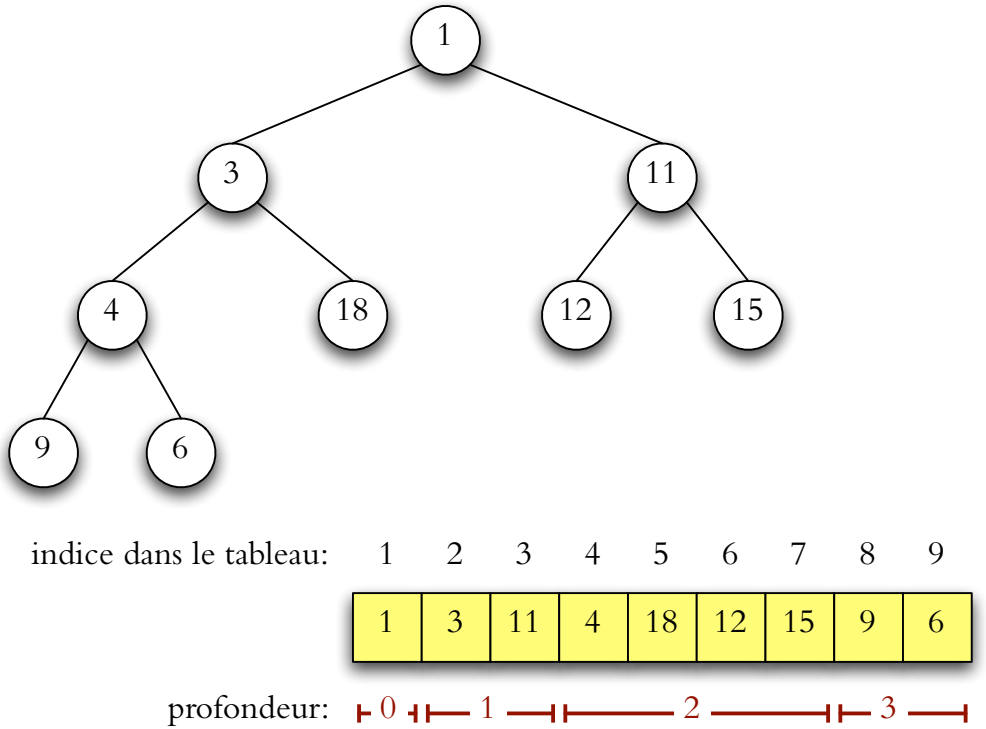


Tas binaire

Arbre binaire complet $\longrightarrow$ pas de pointeurs parent, left, gauche !

Les n clés sont stockés dans un tableau $H[1..n]$.

Parent de nœud i est à $\lceil (i-1)/2 \rceil$, enfant gauche est à $2i$, enfant droit est à $2i+1$.



Tas binaire — insertion

```
    INSERT(v, H) // tas binaire dans H[1..|H|]
I1  NAGER(v, |H| + 1, H)
```

```
     NAGER(v, i, H) // tas binaire dans H[1..|H|]
N1  p ← ⌈(i − 1)/2⌉
N2  while p ≠ 0 et H[p] > v do
N3      H[i] ← H[p]
N4      i ← p
N5      p ← ⌈(i − 1)/2⌉
N6  H[i] ← v
```

(NAGER est «percolate up» dans le livre)

en N1 et N5, on peut juste faire un décalage binaire (`p=i>>1` en Java) — très rapide