Plan du cours

1/ Rappels et compléments Java.

2/ Tableaux, boucles et invariants.

3/ Notions élémentaires de complexité.

4/ Récursion.

5/ Structures de données et introduction aux types abstraits de données. **ICI**

6/ Quelques compléments Java.

Structures déjà vues

- Tableau, File, Liste, Arbre et aujourd'hui `Tas` .

Les Tas

Structures déjà vues

- Tableau, File, Liste, Arbre et aujourd'hui `Tas` .

Définition d'un tas

Un `tas` est un **`arbre binaire`** tassé tel que

- les sommets sont **étiquetés par des clefs**;
- tel que tout sommet possède une **clé supérieure ou égale** aux **cléfs de ses fils**.

Structures déjà vues

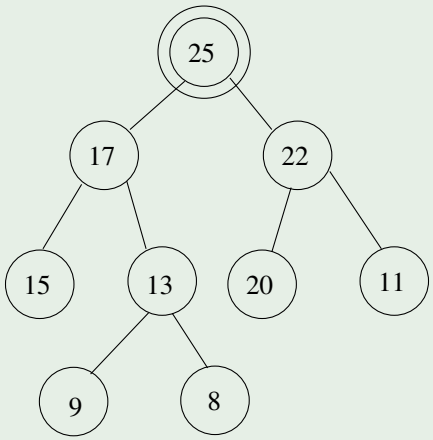
- Tableau, File, Liste, Arbre et aujourd'hui **`Tas`** .

Définition d'un tas

Un **`tas`** est un **`arbre binaire`** tassé tel que

- les sommets sont **étiquetés par des clefs**;
- tel que tout sommet possède une **clé supérieure ou égale** aux **cléfs de ses fils**.

Exemple :



Représentation d'un tas

On représente un tas par un tableau.

Tas et tableaux

Représentation d'un tas

On représente un tas par un tableau.

Parcours et numérotation

1. On (re)numérote les sommets suivant un parcours en largeur.
2. Le sommet dont le rang (la renumérotation) est *i* est rangé à l'indice *i* du tableau de représentation.

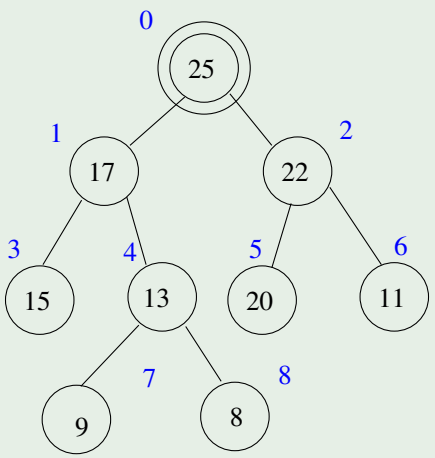
Représentation d'un tas

On représente un tas par un tableau.

Parcours et numérotation

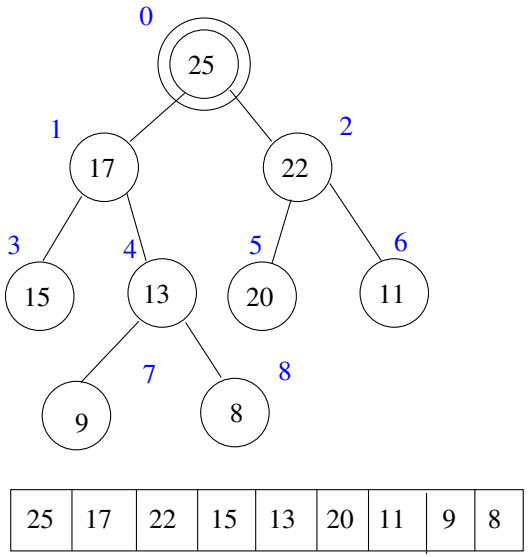
1. On (re)numérote les sommets suivant un parcours en largeur.
2. Le sommet dont le rang (la renumérotation) est *i* est rangé à l'indice *i* du tableau de représentation.

Le même exemple :



25	17	22	15	13	20	11	9	8

Hiérarchie et relations de filiation



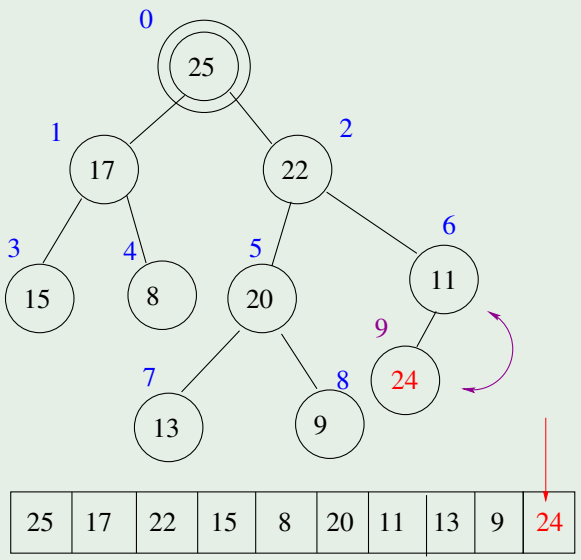
Propriétés

racine	=	sommet 0
père du sommet i	=	sommet $\lfloor (i-1)/2 \rfloor$
fils gauche de i	=	sommet 2i+1
fils droit de i	=	sommet 2i+2
sommet i est une feuille	=	2i+1 $\geq$ n
sommet i a un fils droit	=	2i+2 $<$ n

Insertion dans un tas

- Pour insérer une **nouvelle clef** v:
  1. on insère la clef v à la fin du dernier niveau de l'arbre (à la fin du tableau)
  2. tant que la clef du père de v est plus petite que v on `échange la clef du père de` v `avec` v.

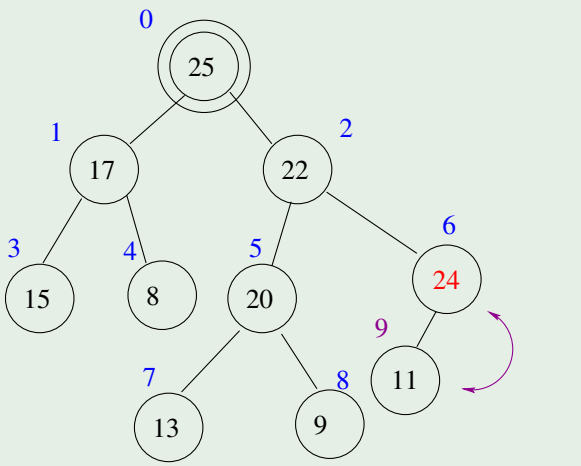
Exemple : insertion de 24



Insertion dans un tas

- Pour insérer une **nouvelle clef** v:
  1. on insère la clef v à la fin du dernier niveau de l'arbre (à la fin du tableau)
  2. tant que la clef du père de v est plus petite que v on `échange la clef du père de` v `avec` v .

Exemple : insertion de 24

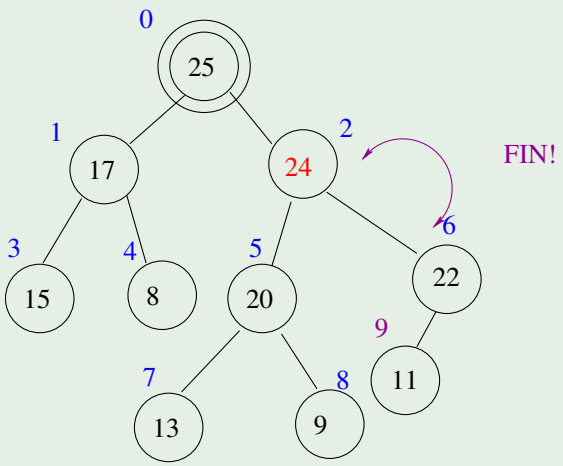


25	17	22	15	8	20	24	13	9	11

Insertion dans un tas

- Pour insérer une **nouvelle clef** v:
  1. on insère la clef v à la fin du dernier niveau de l'arbre (à la fin du tableau)
  2. tant que la clef du père de v est plus petite que v on **échange la clef du père de v avec v**.

Exemple : insertion de 24



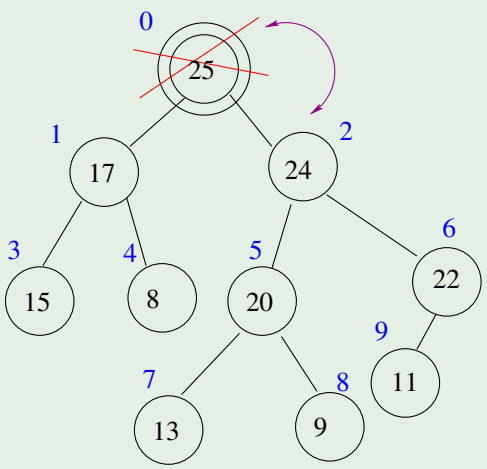
25	17	24	15	8	20	22	13	9	11

Suppression du maximum

- Fait: le maximum c'est la racine!
- Pour enlever la racine,
  1. on remplace la racine par le dernier élément v (celui de la fin du tableau

  2. tant que la clef v est inférieure à celle de l'un de ses fils on **échange la clef de v avec celle du plus grand de ses fils**.

Exemple : suppression de la racine

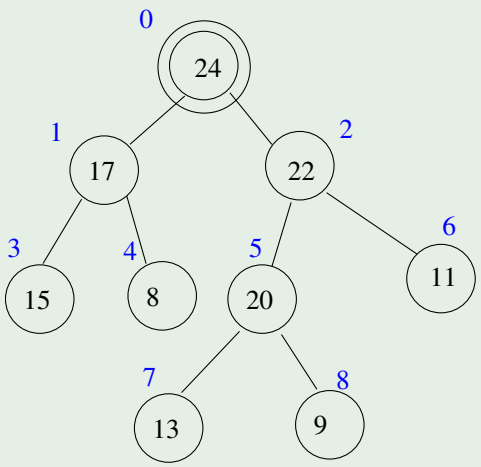


25	17	24	15	8	20	22	13	9	11

Suppression du maximum

- Fait: le maximum c'est la racine!
- Pour enlever la racine,
  1. on remplace la racine par le dernier élément *v* (celui de la fin du tableau
  2. tant que la clef *v* est inférieure à celle de l'un de ses fils on **échange la clef de *v* avec celle du plus grand de ses fils**.

Exemple : suppression de la racine



24	17	22	15	8	20	11	13	9

Utilisations des tas

Complexités moyennes des opérations utilisées

- Rappel de cours (admis!) : la **`hauteur moyenne d'un arbre binaire`** est $O(\log n)$.
- L'insertion d'un élément dans un tas se fait donc en $O(\log n)$.
- La suppression de la racine nécessite $O(\log n)$.
- La construction d'un tas par un algorithme "force brute" (n insertions) coûte donc $O(n \log n)$.

Utilisations des tas

Complexités moyennes des opérations utilisées

- Rappel de cours (admis!) : la **`hauteur moyenne d'un arbre binaire`** est $O(\log n)$.
- L'insertion d'un élément dans un tas se fait donc en $O(\log n)$.
- La suppression de la racine nécessite $O(\log n)$.
- La construction d'un tas par un algorithme "force brute" (n insertions) coûte donc $O(n \log n)$.

Utilisations

Les tas représentent des **`structures de données`** utilisées en algorithmique pour les raisons suivantes.

- Le **`tri par tas`** est l'un des meilleurs algorithmes pouvant triés les éléments **sur place**[a] sans la pénalité quadratique dans le **cas pire** (cf. cours). Ce tri reste en $O(\log n)$.
- En utilisant les tas, trouver le min, le max, le min et le max en même temps ou encore le $k-$ième plus grand élément se fait en temps **linéaire**.
- Ils sont utilisés dans les algorithmes de **graphes** (généralisation des arbres).

[a] l'algorithme ne nécessite pas plus de mémoire que celle contenant déjà les

Le tri par tas (heapsort)

Le principe

- Il s'agît (comme toujours) de trier un tableau (d'entiers par exemple).

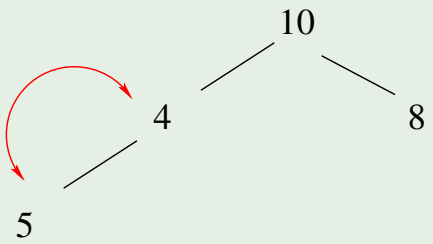
- L'idée consiste à voir ce tableau comme un **`arbre binaire`**. Le premier élément est la racine, le second et le troisième sont les fils de la racine, etc.
- Lors de l'éxécution de l'algorithme, on cherchera à obtenir un tas qui n'est rien d'autre qu'un arbre binaire vérifiant les propriétés du tas.

  Supposons que l'arbre gauche et l'arbre droit soient déjà organisés. L'opération de base consiste à échanger la racine par le plus grand de ses fils si nécessaire, et ainsi récursivement jusqu'à ce que cette ex-racine soit à sa place.

Exemple

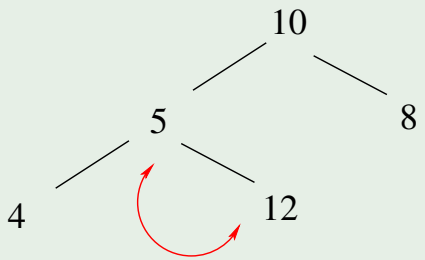
Exemple d'exécution

10	4	8	5	12	2	6	11	3	9	7	1



Exemple d'exécution

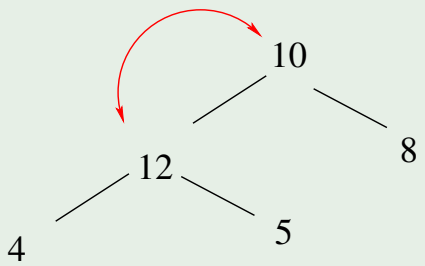
10	5	8	4	12	2	6	11	3	9	7	1



Exemple

Exemple d'exécution

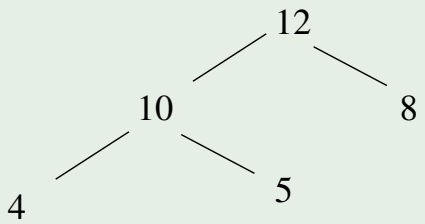
10	12	8	4	5	2	6	11	3	9	7	1



Exemple

Exemple d'exécution

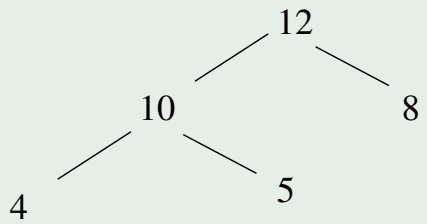
12	10	8	4	5	2	6	11	3	9	7	1



Exemple

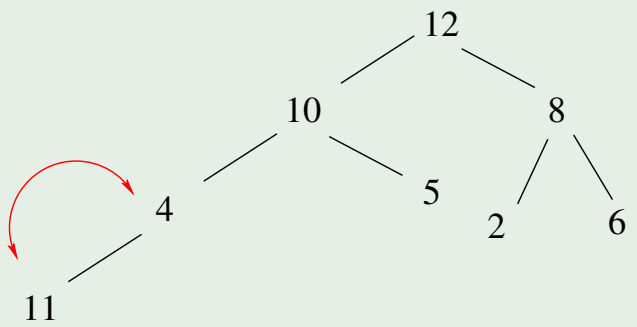
Exemple d'exécution

12	10	8	4	5	2	6	11	3	9	7	1



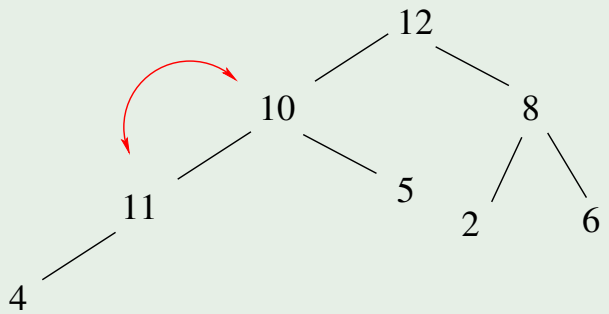
Exemple d'exécution

12	10	8	4	5	2	6	11	3	9	7	1



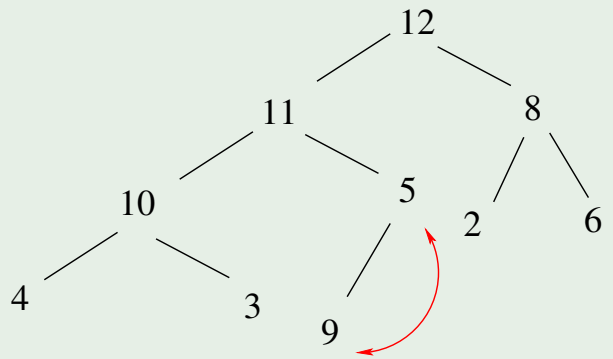
Exemple d'exécution

12	10	8	11	5	2	6	4	3	9	7	1



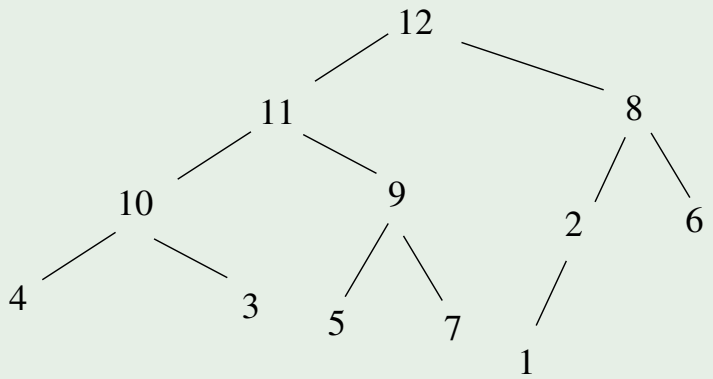
Exemple d'exécution

12	11	8	10	5	2	6	4	3	9	7	1



Exemple d'exécution

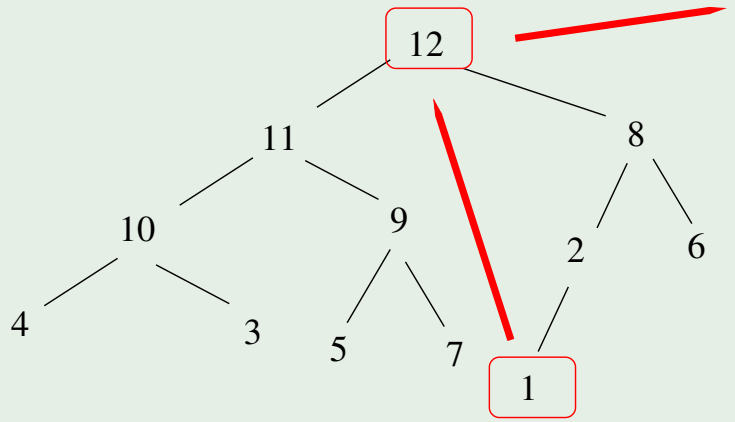
12	11	8	10	9	2	6	4	3	5	7	1



FIN PREMIERE ETAPE D'ORGANISATION

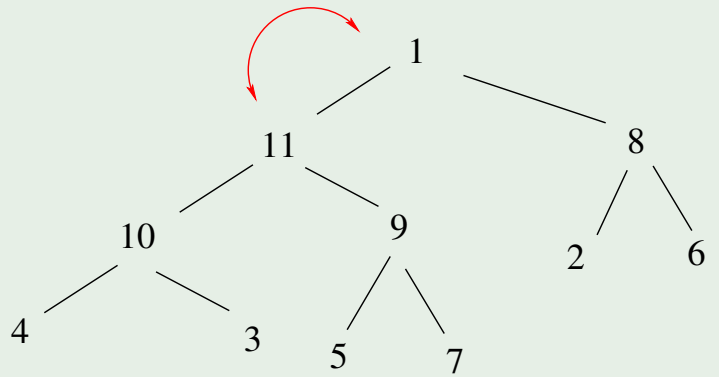
Exemple d'exécution

12	11	8	10	9	2	6	4	3	5	7	1



Exemple d'exécution

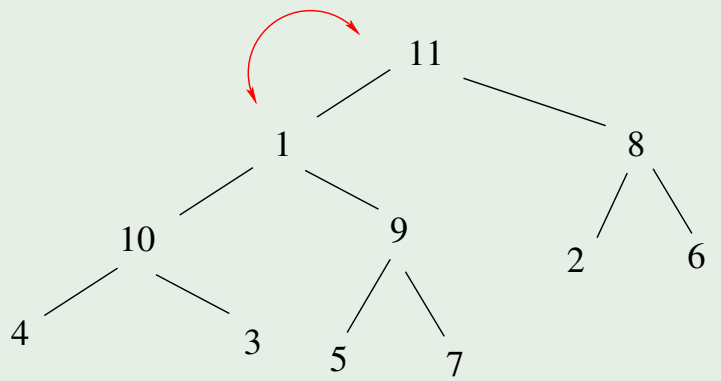
1	11	8	10	9	2	6	4	3	5	7	12



ON RE–ORGANISE

Exemple d'exécution

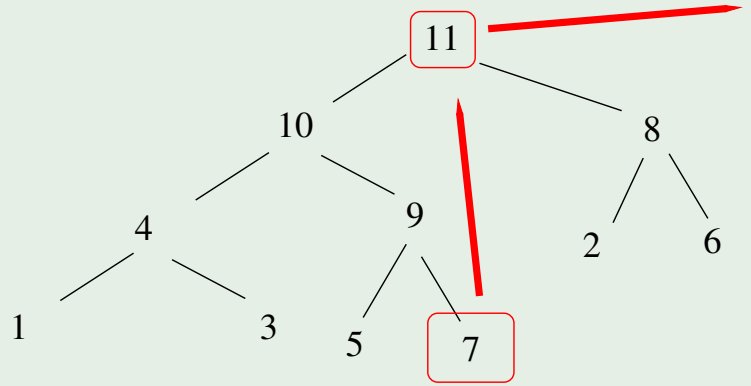
11	1	8	10	9	2	6	4	3	5	7	12



ON RE–ORGANISE

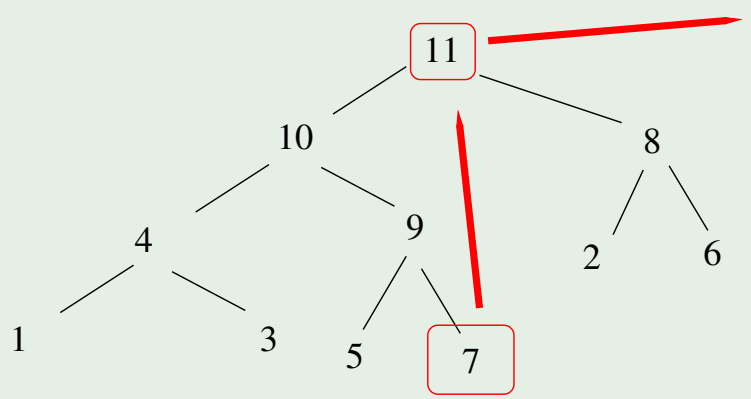
Exemple d'exécution

11	10	8	4	9	2	6	1	3	5	7	12



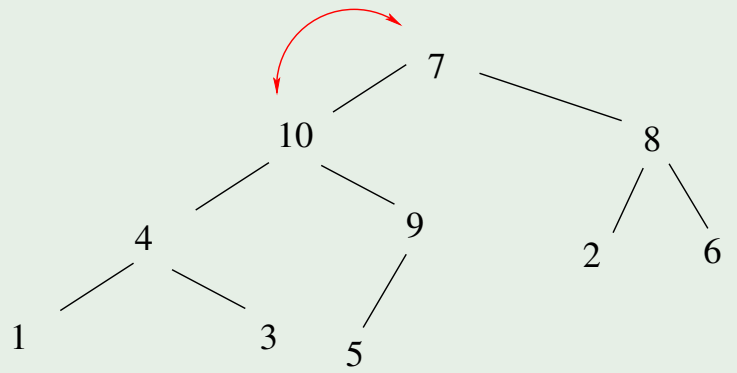
Exemple d'exécution

7	10	8	4	9	2	6	1	3	5	11	12



Exemple d'exécution

7	10	8	4	9	2	6	1	3	5	11	12

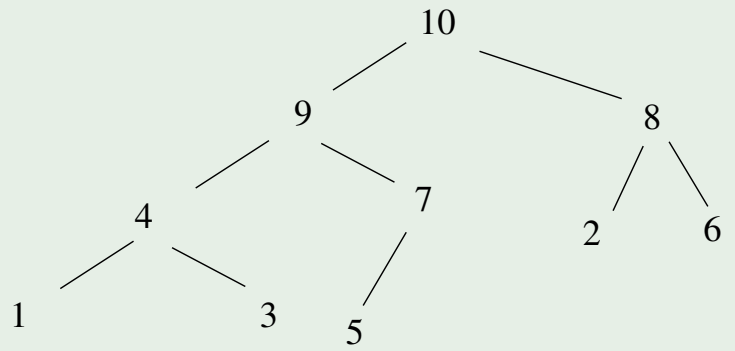


RE–ORGANISATION ...

Exemple

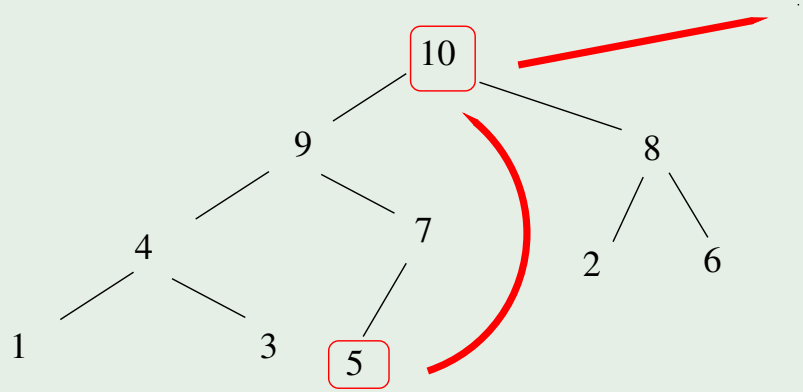
Exemple d'exécution

10	9	8	4	7	2	6	1	3	5	11	12



Exemple d'exécution

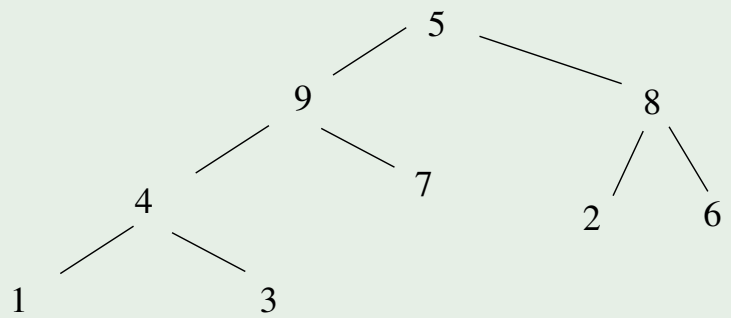
10	9	8	4	7	2	6	1	3	5	11	12



Exemple

Exemple d'exécution

5	9	8	4	7	2	6	1	3	10	11	12

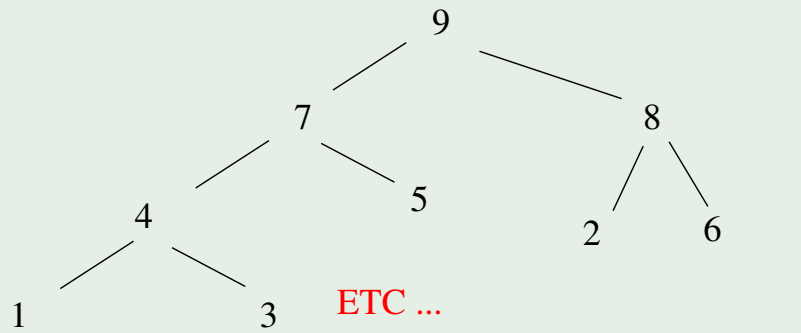


ON REORGANISE TOUT CA

Exemple

Exemple d'exécution

9	7	8	4	5	2	6	1	3	10	11	12



IMPORTANT: CA TRIE SUR PLACE

Un tas en JAVA

Un tas très simple

```
class Tas {
        private int [ ] t; // tableau des sommets
        private int n; // nombre de sommets
        Tas (int taille) {
                t = new int [taille+1];
                n = 0;
        }
        int pere(int i) {return( (i-1)/2 );}
        int filsgauche(int i) { return 2*i+1; }
       int filsdroit(int i) { return 2*i+2;}
        boolean estUneFeuille(int i) { return (2*i+1 >= n);}
        ...
       }
```

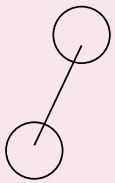
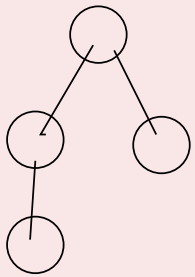
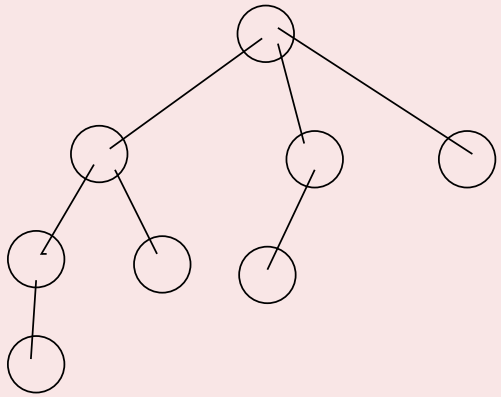
Définition d'unarbre binomial

Un **`arbre binomial`** est défini récursivement par:

- L'arbre binomial d'ordre 0 est un réduit à un nœud.
- L'arbre binomial d'ordre k possède une racine de degré k dont les fils sont **dans l'ordre** un arbre binomial d'ordre $k-1$ suivi d'un arbre binomial d'ordre $k-2$, ..., d'un arbre binomial d'ordre 1 et puis d'un arbre binomial d'ordre 0 (un nœud).

Pourquoi “binomial”?

Un arbre binomial d’ordre 0 a un nœud. D’après la définition précédente, un arbre binomial d’ordre 1 a donc 2 nœuds (la racine et son fils). Un arbre binomial d’ordre 2 a une racine avec deux fils ...

Specimen				
Ordre	0	1	2	3

Soit T_k le nombre de nœuds dans un arbre binomial d’ordre k. Nous avons $T_0 = 1$ et $T_k = 1 + \sum_{i=0}^{k-1} T_i = 2^k$. Et $2^k = \sum_{i=0}^{k} \binom{k}{i}$.

Le tas binomial

Définition

Un tas binomial est un ensemble d'arbres binomiaux tels que

- chaque arbre possède une **structure en tas** : la clef de chaque nœud est inférieure à celle de son père
- pour tout entier naturel i, il existe au plus un tas binomial d'ordre i.

Le tas binomial

Définition

Un tas binomial est un ensemble d'arbres binomiaux tels que

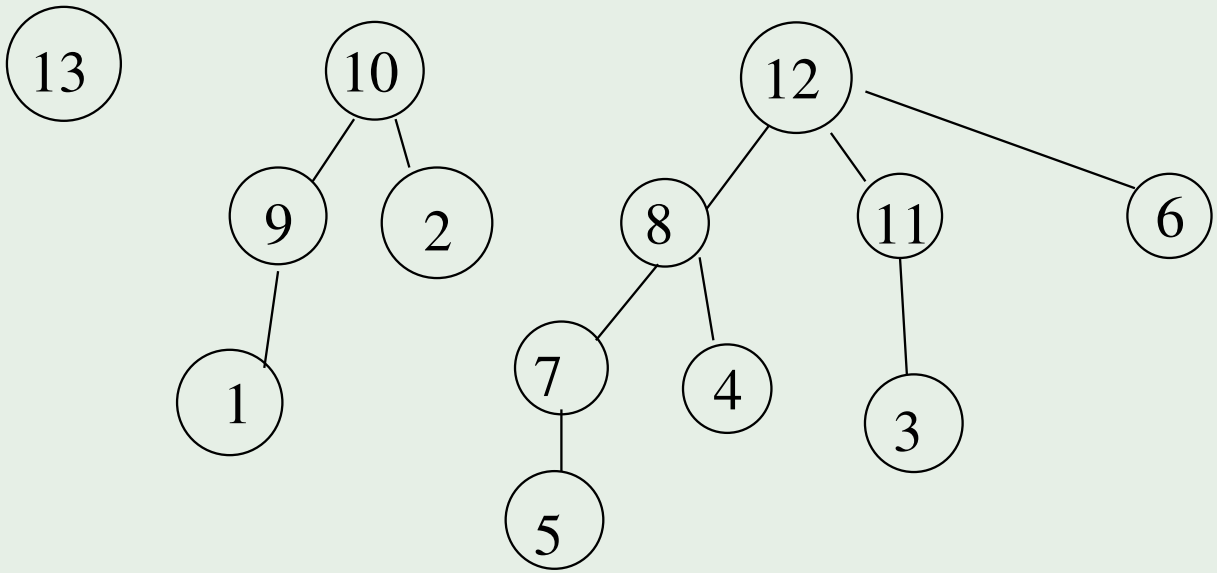
- chaque arbre possède une **structure en tas** : la clef de chaque nœud est inférieure à celle de son père
- pour tout entier naturel i, il existe au plus un tas binomial d'ordre i.

Conséquences

- Chaque racine d'un arbre (première propriété) est un maximum "local".
- D'après la seconde propriété, un tas binomial contenant n éléments consiste en au plus un certain nombre d'arbres bien déterminé.

Exemple $n = 13$

Observons que $13 = 1101$ en binaire. On a $13 = 2^3 + 2^2 + 2^0$. Le tas binomial à 13 éléments consiste à 3 arbres binomiaux d'ordres 0, 2 et 3.



Opérations sur les tas binomiaux

Recherche en temps logarithmique

Le nombre d'arbres d'un tas binomial de taille n est logarithmique (i.e. $O(\log n)$. Exercice!). Pour chercher le plus grand élément dans un tas binomial, il nous suffit de trouver la plus grande des racines des arbres.

Opérations sur les tas binomiaux

Recherche en temps logarithmique

Le nombre d'arbres d'un tas binomial de taille n est logarithmique (i.e. $O(\log n)$. Exercice!). Pour chercher le plus grand élément dans un tas binomial, il nous suffit de trouver la plus grande des racines des arbres.

Fusion

L'algorithme de fusion de **deux tas** est un exemple d'algorithme très élégant et très efficace!! Le principe est le suivant :

- Si seul un des deux tas contient un arbre d'ordre j alors cet arbre est directement rajouté dans le tas fusionné (le résultat).
- Sinon, les deux tas contiennent chacun un et un seul arbre d'ordre j et on les fusionne en un tas d'ordre $j+1$. On rajoute ce nouveau arbre dans le tas fusionné en respectant la structure de tas. Notez que l'on peut avoir besoin de fusionner cet arbre avec un arbre d'ordre $j+1$ dans l'un des deux tas.
- Durant l'exécution de l'algorithme, on examinera donc au plus 3 arbres. Or l'ordre d'un arbre est logarithmique. Donc la complexité de la fusion est en au plus $O(\log n)$.

Suite des opérations

Insertion

Pour insérer un nouvel élément, on crée simplement un nouveau tas contenant cet élément et on fusionne. Donc en temps logarithmique.

Suite des opérations

Insertion

Pour insérer un nouvel élément, on crée simplement un nouveau tas contenant cet élément et on fusionne. Donc en temps logarithmique.

Supprimer le max

Pour **supprimer le plus grand élément**, on trouve cet élément, on l'enlève de son arbre binomial. On obtient alors une liste de sous-arbres que l'on transforme en un autre tas, qui sera fusionné avec le tas précédent.

Suite des opérations

Insertion

Pour insérer un nouvel élément, on crée simplement un nouveau tas contenant cet élément et on fusionne. Donc en temps logarithmique.

Supprimer le max

Pour **supprimer le plus grand élément**, on trouve cet élément, on l'enlève de son arbre binomial. On obtient alors une liste de sous-arbres que l'on transforme en un autre tas, qui sera fusionné avec le tas précédent.

Supprimer un élément quelconque

On remplace cet élément par "$+\infty$". Et on se ramène à l'algorithme de suppression du max!!