Introduction à l'Algorithmique II

Chapitre

Arbres Binaires

Tas et Tri par Tas

(Heaps and Heap sort)

A.U: 2018 - 2019

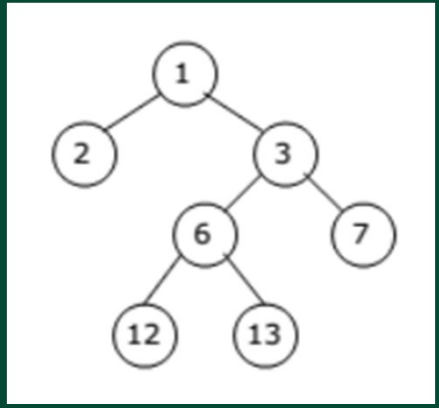
Arbres

- Un arbre est schématisé (modélisé) par des nœuds dont chacun possède éventuellement des descendants (fils).
- Chaque nœud possède au plus un père.
- Le nœud ne possédant pas de père s'appelle **racine de l'arbre**. (en haut de l'arbre)
- Un nœud ne possédant pas de fils est une **feuille**.
- **Arbre binaire**: arbre où chaque nœud possède au plus deux fils. On parle de fils gauche (FG) et fils droit (FD).
- Le fils gauche (resp. droit) est la racine du sous-arbre gauche (resp. droit).

Arbres

- La **profondeur** d'un nœud p dans un arbre est la longueur (nombre d'arêtes) du chemin à partir de la racine jusqu'au nœud p.

- Ex:
  - La profondeur du nœud valué par 7 est égale à 2
  - La profondeur du nœud valué par 13 est égale à 3
  - La profondeur du nœud valué par 2 est égale à 1
  - La profondeur du nœud valué par 1 est égale à 0



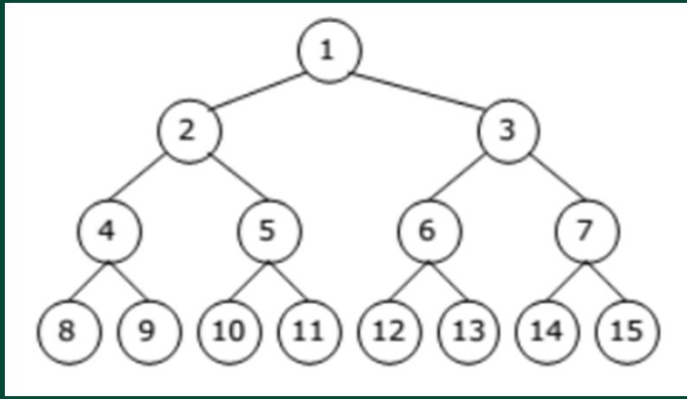
- La **hauteur** d'un arbre binaire est la profondeur maximale d'un nœud, ou -1 si l'arbre est vide.

- Ex: La hauteur de l'arbre ci-haut est égale à 3.

Arbres Binaires Complet (ABC)

- Un **arbre binaire complet** est un arbre dont la profondeur de chaque feuille est égale à la hauteur h de l'arbre.

- Exemple:



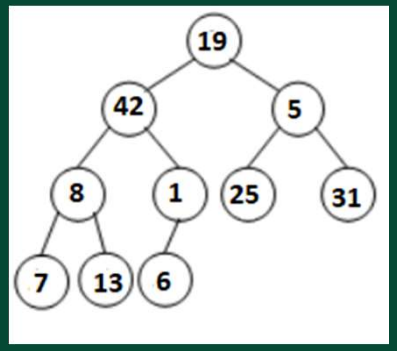
- Le nombre de **nœuds** d'un arbre binaire complet est $2^{h+1} - 1$,

- Le nombre de **feuilles** d'un arbre binaire complet est 2^h

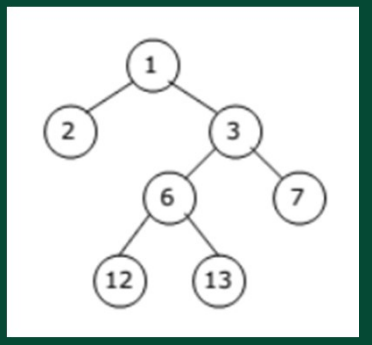
Arbres Binaires Presque-Complet (ABPC)

- Un **arbre binaire presque-complet** est un arbre de hauteur h tel que:
  1. Pour d : 0,…,h-1 il y a 2^d nœud(s) de profondeur d
  2. Les feuilles de profondeur h sont les feuilles les plus à gauche.
- La condition 2 ci-dessus, peut être remplacée par:
  - Si un nœud p de profondeur h-1 a un FG, alors chaque nœud de profondeur h-1 à gauche de p possède 2 fils.
  - Si un nœud de profondeur h-1 possède un FD, alors il a aussi un FG.
- La profondeur des feuilles d'un ABPC est égale à h ou h-1
- Nomenclature anglophone:
  - Nearly complete binary tree, or
  - Almost complete binary tree

- Exemple d'AB Presque-complet:



- Contre Exemple (AB non presque complet):



ABPC : Implémentations

- Par tableau A[]:
  - Utilisation des indices
  - Le fils gauche de A[i], s'il existe, est à la position 2*i + 1
  - Le fils droit de A[i], s'il existe, est à la position 2*i + 2
  - Le père de A[i], exceptée la racine, est à la position: (i-1)/2

- Par structure récursive (Pointeurs) :

```
Struct ABPC
Début
      Val : Entier
      FG : ^ Struct ABPC
      FD : ^ Struct ABPC
Fin
```

ABPC : Implémentations

- Par structure récursive :
  - Un arbre est identifié par l'adresse (POINTEUR) de sa racine.
  - Un nœud est une feuille si les PFG et PFD sont des pointeurs nuls.

- Exemples: (on suppose que l'arbre A est déjà rempli)

```
Variable P : ^ Struct ABPC
P ← &A
// Accès au fils gauche:
P ← P -> FG

// Accès à la feuille la plus à gauche, si elle existe
TantQue (P -> FG)
      P ← P -> FG
FinTantQue
```

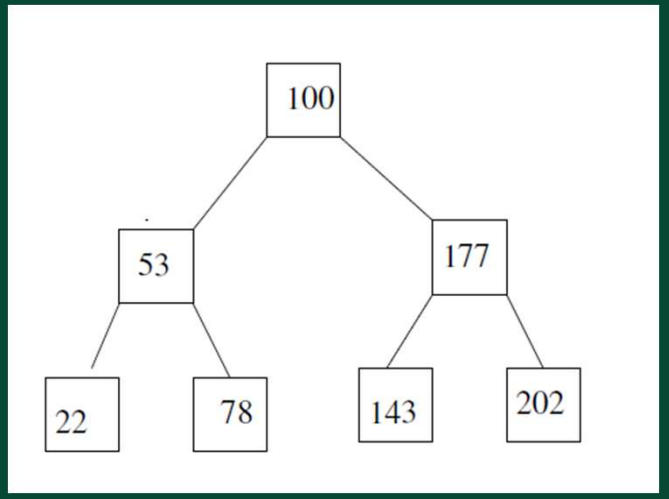
ABPC : Parcours

- En largeur – par niveau de hauteur (cf tas):
  - 100 – 53 – 177 – 22 – 78 – 143 - 202

- En profondeur:
  - prefixe (RGD):
  - 100 – 53 – 22 – 78 – 177 – 143 - 202
  - (RGD ~ 1er passage)

  - Infixe (GRD):
  - 22 - 53 - 78 - 100 - 143 -177 - 202
  - (GRD ~ 2ème passage)

  - suffixe (GDR dern pass):
  - 22 – 78 – 53 – 143 – 202 – 177 - 100
  - (GDR ~ dernier passage)



Arbre Binaire de Recherche (ABR)

- Un arbre binaire de recherche de racine A est un arbre binaire tel que:
  - chaque élément du sous-arbre gauche est inférieur ou égal à A,
  - chaque élément du sous-arbre droit est supérieur ou égal à A
  - selon la mise en œuvre de l'ABR, on pourra interdire ou non des éléments de valeur égale.
  - Les éléments que l'on ajoute deviennent des feuilles de l'arbre.

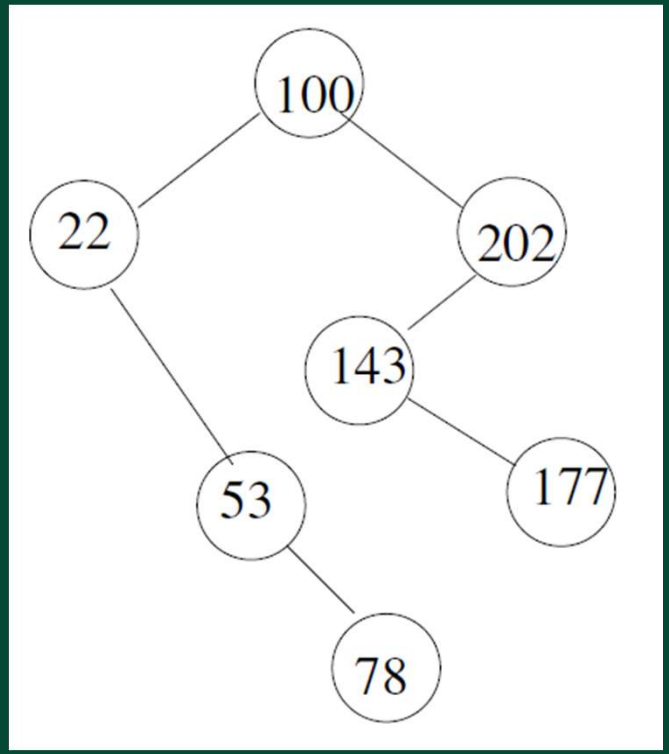
- Utilité: rapidité de
  - Recherche d'un élément (spécialement du Min ou du Max)
  - insérer un élément
  - supprimer un élément

ABR : Parcours

- En profondeur: (Attention aux nœuds qui n'ont pas de fils gauche)
  - prefixe (RGD):
  - 100 - 22 - 53 - 78 - 202 - 143 – 177
  - (RGD ~ 1er passage)

  - Infixe (GRD):
  - 22 - 53 - 78 - 100 - 143 -177 - 202
  - (GRD ~ 2ème passage)

  - suffixe (GDR dern pass):
  - 78 - 53 - 22 - 177 - 143- 202 – 100
  - (GDR ~ dernier passage)



Tas

- Un tas binaire est schématisé par un arbre binaire presque-complet, où chaque nœud est **prioritaire** par rapport à ses fils,

- Implémenté par un tableau T tel que pour chaque élément T[i] :
  - Le fils gauche, s'il existe, est à la position 2*i + 1
  - Le fils droit, s'il existe, est à la position 2*i + 2
  - Le père, exceptée la racine, est à la position: (i-1)/2

- Si la priorité est représentée par une relation de **supériorité**, on parle de **tas max**, tel que:

$$T[\text{parent}(i)] \geq T[i]$$

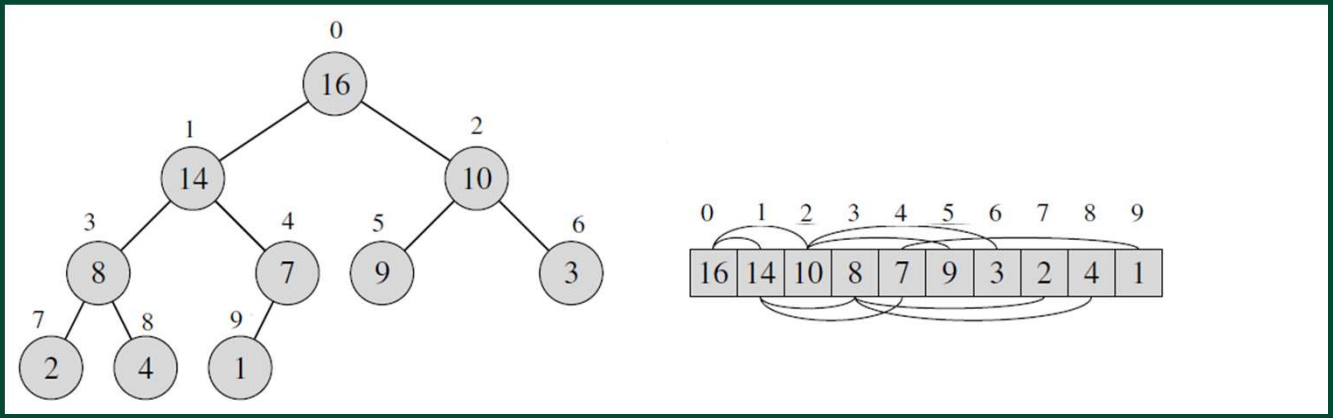
Tq la fonction parent(i) retourne (i-1)/2

- Si la priorité est représentée par une relation d'**infériorité**, on parle de **tas min**, tel que:

$$T[\text{parent}(i)] \leq T[i]$$

Tas

- Exemple de tas max:



– Noter que pour chaque nœud, il n y a pas nécessairement une relation d'ordre entre le fils gauche et le fils droit:
  - FG(0) > FD(0)
  - FG(3) < FD(3)

– Noter aussi qu'on peut remplir le tableau T à partir de l'arbre en faisant un parcours horizontal de chaque niveau de profondeur, en commençant par la racine (niveau 0, puis 1, …).

Tas

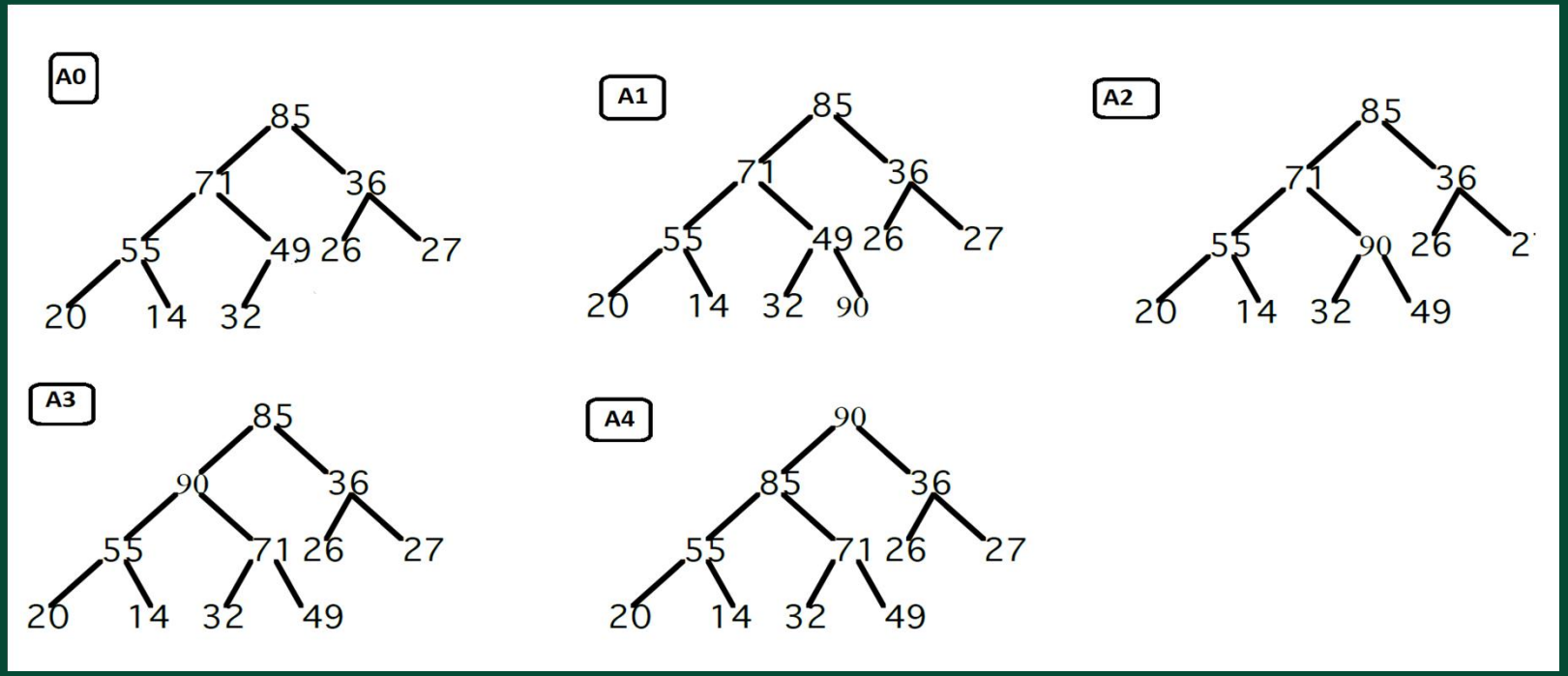
- Un tas (binaire) T a deux attributs :
  - longueur[T]: nombre d'éléments du tableau T,
  - taille[T]: nombre d'éléments du tas rangés dans le tableau T.

  (cf application plus loin)

Attention: Ne pas confondre le tas étudié dans ce chapitre avec le tas qui est une portion d'espace mémoire récupérable.

Tas: Ajout d'un élément

- Ajout d'un élément v à un tas T de taille n (dernier élément du tas est à l'indice n-1)
  - Insérer le nouvel élément à l'indice n,
  - Tant qu'on n'a pas atteint la racine, faire pour le nœud v :
    - Si la propriété du tas max est vérifié avec le père, alors **arrêt**
    - Sinon Permuter le nœud en question avec son père.
- Exemple: Ajout de 90 au tas T: 85 71 36 55 49 26 27 20 14 32



Tas: Ajout d'un élément:: Pseudo-code

```
Fonction permuter(a: ^ Entier, b: ^Entier): vide
Variable tmp: Entier
Début
    tmp ← a^
    a^ ← b^
    b^ ← tmp
Fin

Fonction aj_Elm_Tas(T[]: Entier, taille: Entier, v: Entier): vide
Variables: indfils, indpere : Entier
Début
    T[taille] ← v
    indfils ← taille
    indpere ← (indfils -1)/2
    TantQue (T[indpere] < T[indfils] ET indpere ≥ 0) alors
        permuter(&T[indpere],&T[indfils])
        indfils ← indpere
        indpere ← (indfils -1)/2
    FinTantQue
Fin
```

Construction d'un Tas max

- Nous allons construire un tas max à partir d'un tableau T de taille n

- Idée:

  Pour chaque élément T[i], tq i ≥ 1
  - Supposer que le tableau T[0 … i-1] constitue déjà un tas max
  - Ajouter l'élément T[i] au tas T[0 … i-1] de taille i

- Pseudo-code:

```
Fonction Const_Tas(T[] : Entier, n: Entier): vide
Variable i : Entier
Début
   Pour i allant de 1 à n-1 Faire
     aj_Elm_Tas(T, i, T[i])
   FinPour
Fin
```

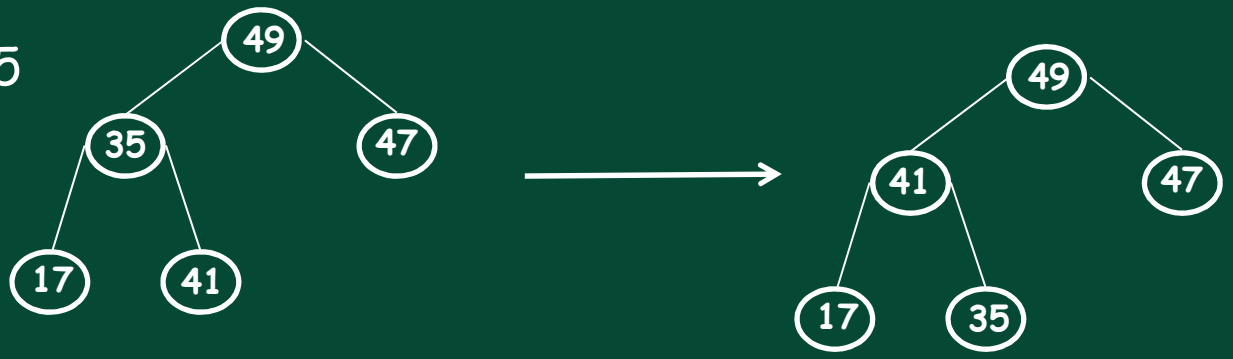
- Ex: construire un tas max à partir du tableau T: { 75, 50, 80, 90, 70, 20, 45, 60, 30, 111 }

Tas max: Descente d'un élément

- Soit T un tas max de taille n avec une exception: " L'élément T[i] représente une **violation singulière** de la propriété du tas max "

- Solution: Faire descendre T[i] dans l'arborescence jusqu'à un nœud où la propriété du tas max soit respectée.
- Comment ?
  1. Tant qu'on n'a pas atteint une feuille:
  2. Echanger le nœud qui représente la violation avec le plus grand de ses fils, et vérifier:
  3. S'il n'y a plus de violation, alors **arrêt**
  4. Sinon, retour à 1

- Exemple: nœud 35



Tas max: Descente d'un élément :: Pseudo-code

```
Fonction Desc_Elm(T[]: Entier, i: Entier, taille: Entier): vide
Variables indFG, indFD, posTouve : Entier
indFG ← 2*i+1 , indFD ← 2*i+2 , posTouve ← 0
TantQue (indFG < taille ET indFD < taille ET posTouve = 0)
    Si (T[i] >= T[indFG] ET T[i] >= T[indFD])
       posTouve ← 1
    Sinon si (T[indFG] < T[indFD])    // Permuter avec le plus grand des fils
          permuter(&T[i],&T[indFD])
          i ← 2*i+2
          sinon
              permuter(&T[i],&T[indFG])
              i ← 2*i+1
          FinSi
    FinSi
    indFG ← 2*i+1
    indFD ← 2*i+2
FinTantQue
Si (posTouve = 0 ET indFG < taille ET T[i] < T[indFG] )
    permuter(&T[i],&T[indFG])
FinSi
Fin
```

Tri par Tas

- Soit T un tableau de longueur n, que l'on veut trier dans un ordre croissant, le tri par tas consiste à :
  - Transformer T en un tas max
  - Répéter n-1 fois (extraction de la racine):
    1. Décrémenter la taille du tas (et non la longueur du tableau)
    2. Permuter la racine T[0] avec T[taille] (la racine est le max du tas T)
    3. Descente de la racine (nouveau T[0])
- Pseudo-code

```
Fonction Tri_Tas (Tab[]: Entier, n: Entier): vide
Variable taille : Entier
Début
Const_Tas(Tab,n)
taille ← n
Pour i allant de 0 à n-2 Faire
   taille ← taille-1
   permuter(&Tab[0],&Tab[taille])
   Desc_Elm(Tab,0,taille)
FinPour
Fin
```

references

- Introduction à l'algorithmique. Cours et exercices. Cormen et al. 2e édition.
- http://examradar.com/binary-trees/
- https://en.wikipedia.org/wiki/Binary_tree
- http://tdinfo.phelma.grenoble-inp.fr/1Apet/td/td8b.pdf
- http://homepages.math.uic.edu/~leon/cs-mcs401-s08/handouts/nearly_complete.pdf