Chapitre 1 : Les arbres binaires et tas

Informatique pour tous – MP

Structure arborescente : motivation

- arbres généalogiques
- tournois sportifs
- probabilités
- stockage d'expressions algébriques
- théorie des jeux...

Table des matières

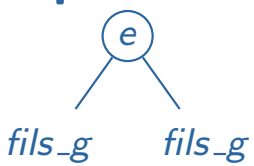
Définition récursive des arbres binaires

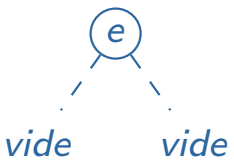
Définition : un arbre binaire étiqueté est

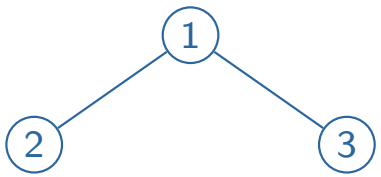
- soit l'arbre vide
- soit de la forme $a = [e, fils\_g, fils\_d]$ où
  - $e \in E$: étiquette de la racine
  - $fils\_g$, $fils\_d$: arbres binaires étiquetés disjoints

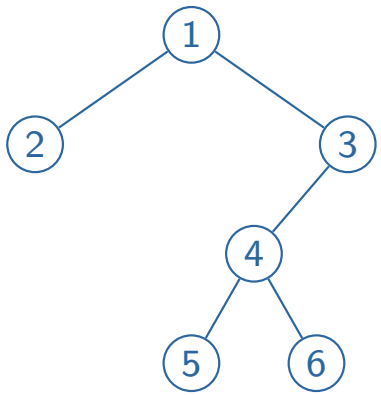
Rq : on peut définir des arbres avec plus de fils (liste).

Représentation graphique :

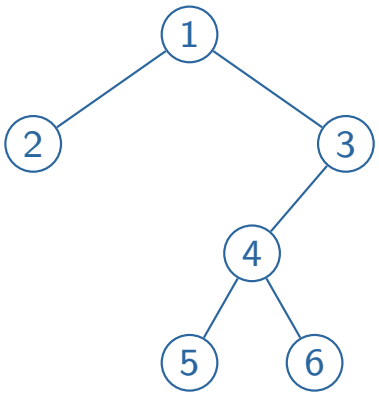




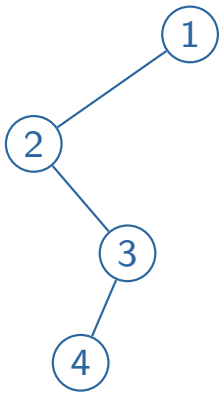




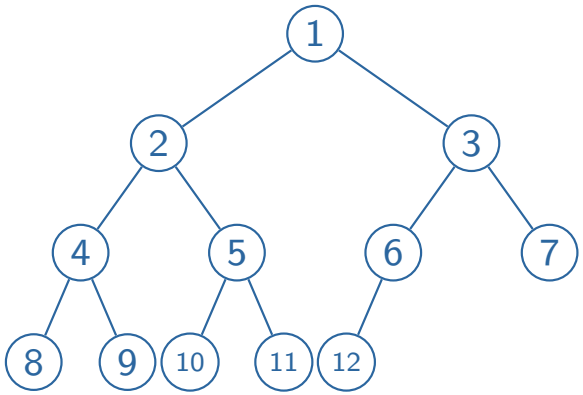
Définition récursive des arbres binaires



Exemple 1



Exemple 2



Exemple 3

Vocabulaire :

- *fils_g* et *fils_d* sont les **fils gauche et droit** de *a*
- *a* est le **père** de *fils_g* et *fils_d*
- les triplets $[e, \mathit{fils\_g}, \mathit{fils\_d}]$ constituant *a* sont les **nœuds**
- le seul nœud qui n'a pas de père est la **racine**
- les nœuds avec fils vides sont les **feuilles**

Notion de hauteur

Definition (Hauteur)

- La hauteur (ou profondeur, ou niveau) d'un nœud est définie par :

$$h(x) = \begin{cases} 0 & \text{si } x \text{ est la racine} \\ 1 + h(y) & \text{si } y \text{ est le père de } x \end{cases}$$

- La hauteur d'un arbre est la hauteur maximale d'un de ces nœuds

Propriété : la hauteur h d'un arbre à n nœuds vérifie

$$\lfloor \log_2 n \rfloor \leq h \leq n - 1$$

Notion de hauteur

Propriété : la hauteur h d'un arbre à n nœuds vérifie

$$\lfloor \log_2 n \rfloor \leq h \leq n-1$$

Preuve

- majoration de h :
  - $h+1$ niveaux
  - au moins $h+1$ nœuds
  - $n \geq h+1$
  - $h \leq n-1$
- minoration de h :
  - Montrons $(\mathcal{P}_k)$: $\forall k \in \mathbb{N}$, il y a au plus 2^k nœuds au niveau k
    (I) $k=0$: une seule racine $(2^0=1)$
    (H) soit $k \in \mathbb{N}$, tq $(\mathcal{P}_k)$, chaque nœud a au plus 2 fils, $(\mathcal{P}_{k+1})$ vraie
    (C) $(\mathcal{P}_k)$ est vrai pour tout $k \in \mathbb{N}$
  - $\sum_{k=0}^{h} 2^k \geq n$, donc $2^{h+1}-1 \geq n$, donc $2^{h+1} > n$
    Ainsi $\log_2 n < h+1$, donc $\lfloor \log_2 n \rfloor \leq h$.

Arbres parfaits

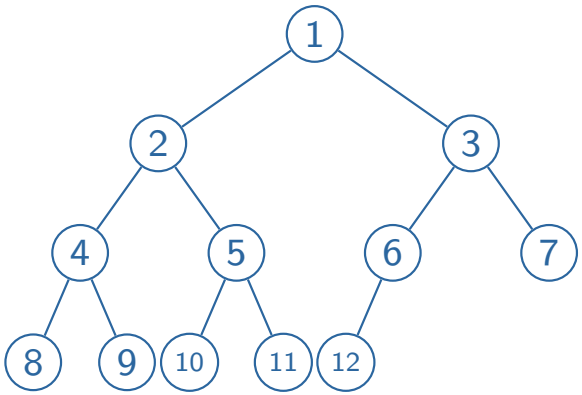
Définition : un arbre est **parfait** :

- si tous ses niveaux sont remplis
- sauf éventuellement le dernier
- le dernier niveau est groupé à gauche

Propriété :

la hauteur d'un arbre parfait à n nœuds est exactement $\lfloor \log_2 n \rfloor$

Exemple 3 :



Primitives du type abstrait `arbre binaire`

- `est_vide(a)` : renvoie `Vrai` si l'arbre `a` est vide, `Faux` sinon
- `arbre_vide()` : renvoie l'arbre vide ;
- `cons(e,g,d)` : renvoie l'arbre d'étiquette `e`, de fils gauche (*resp.* droit) `g` (*resp.* `d`) ;
- `contenu(a)` : renvoie l'étiquette de l'arbre **non vide** `a` ;
- `f_g(a)` (*resp.* `f_d(a)`) : renvoie le fils gauche (*resp.* droit) de l'arbre **non vide** `a`.

Implémentation par une liste

En Python, on peut simplement utiliser la structure de liste :

$$\texttt{cons}(\texttt{e}, \texttt{g}, \texttt{d}) = [\texttt{e}, \texttt{g}, \texttt{d}]$$

- **Exemple 1 :**
  [1, [2, [], []], [3, [4, [5, [], []], [6, [], []]], []]]
- **Exemple 2 :**
  [1, [2, [], [3, [4, [], []], []]], []]
- **Exemple 3 :**
  [1, [2, [4, [8, [], []], [9, [], []]], [5, [10, [], []], [11, [], []]]], [3, [6, [12, [], []], []], [7, [], []]]]

Représentation par un tableau

ID : stocker dans un tableau `t` les étiquettes

Mise en œuvre

- on suppose les nœuds numérotés :
  - de haut en bas
  - de gauche à droite
  - à partir de 1
- dans la case `t[k]`, on stocke l'étiquette du nœud numéro `k` (ou -1)

Conséquences

- niveau `i` : indices $[\![2^i, 2^{i+1} - 1]\!]$
- fils de `t[k]` : `t[2*k]` et `t[2*k+1]`
- père de `t[k]` : `t[k//2]`

Représentation par un tableau : exemple

- **Exemple 1 :**
  $[-1, 1, 2, 3, -1, -1, 4, -1, -1, -1, -1, 5, 6]$
- **Exemple 2 :**
  $[-1, 1, 2, -1, -1, 3, -1, -1, -1, -1, 4]$
- **Exemple 3 :**
  $[-1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]$

Conclusion

- Adapté aux arbres parfaits (donc aux tas...)
- sinon : complexité spatiale exponentielle

Définition des tas

Definition (Tas)

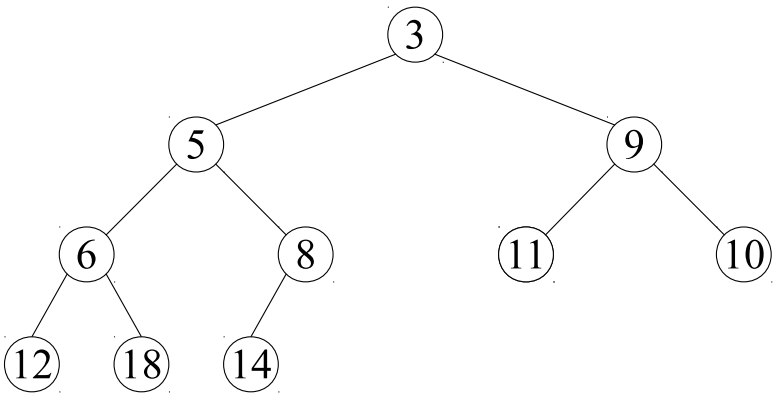
Un **tas** est un arbre binaire :

- parfait
- tq l'étiquette de chaque nœud est supérieure à celle de son père

Conséquence : l'étiquette de la racine est minimale

Exemple : $[-1, 3, 5, 9, 6, 8, 11, 10, 12, 18, 14]$ est un tas

Plus clair comme ça :



Point CG : introduit par J.W.J. Williams en 1964 pour expliciter son algorithme de tri par tas...(sujet du TP)

Affichage des arbres

Objectif

Indications :

- `' '*n` construit une chaîne de `n` espaces
- `'{: 3}'.format(e)` crée une chaîne de 3 caractères où `e` est centré

```
def afficher(t):
    n=len(t)-1
    h=int(log2(n))
    nbsc=[1]
    for i in range(h):
        nbsc=[2*nbsc[0]+1]+nbsc
        nbsi=[1]
    for i in range(h):
        nbsi=[2*nbsi[0]+3]+nbsi
    for i in range(h+1):
        ligne=' '*nbsc[i]
        for j in range(2**i,min(2**(i+1),n+1)):
            ligne=ligne+'{:^3}'.format(t[j])+' '*nbsi[i]
        print(ligne)
```