Arbres binaires

Quentin Fortier

December 16, 2021

Résumé des structures de données : Linéaires/séquentielles

Structures de données linéaires/séquentielles (éléments rangés les uns après les autres) :

1. **Tableau** : permet l'accès à un élément quelconque en O(1)
   $\widehat{\text{T}}$ableau dynamique, table de hachage...
2. **Liste chaînée** : permet suppression/ajout en O(1)
   $\widehat{\text{L}}$iste doublement chaînée, cyclique...

Résumé des structures de données : Hiérarchiques

Structures de données hiérarchiques (éléments rangés par niveau) :

1. **Arbre**
2. **Arbre équilibré (AVL, arbre rouge-noir...)** : permet d'ajouter et supprimer en $\mathrm{O}(\log(n))$
3. **Tas** : permet d'implémenter une file de priorité
4. **Trie (arbre préfixe)** : pour chercher des chaînes de caractères
5. ...

Arbre binaire

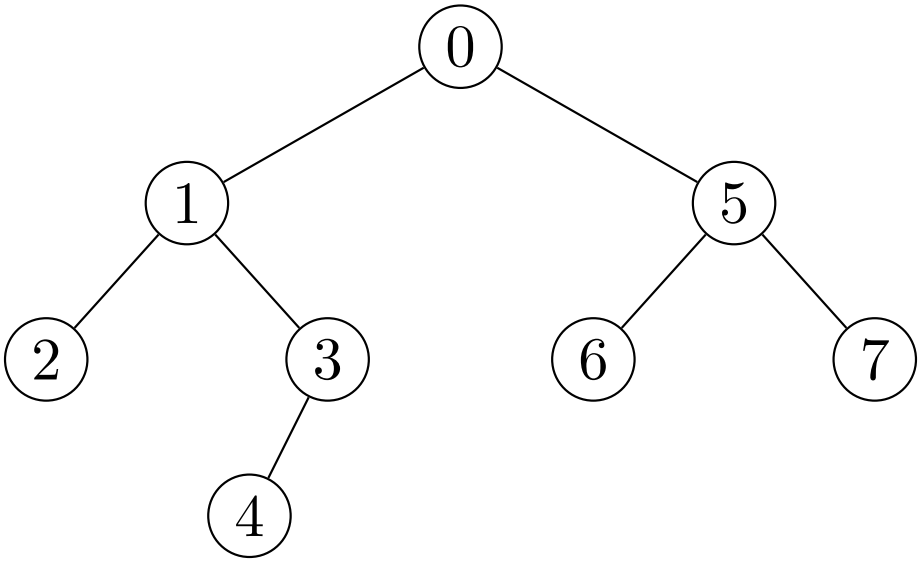
On définit un **arbre binaire** récursivement.

Définition

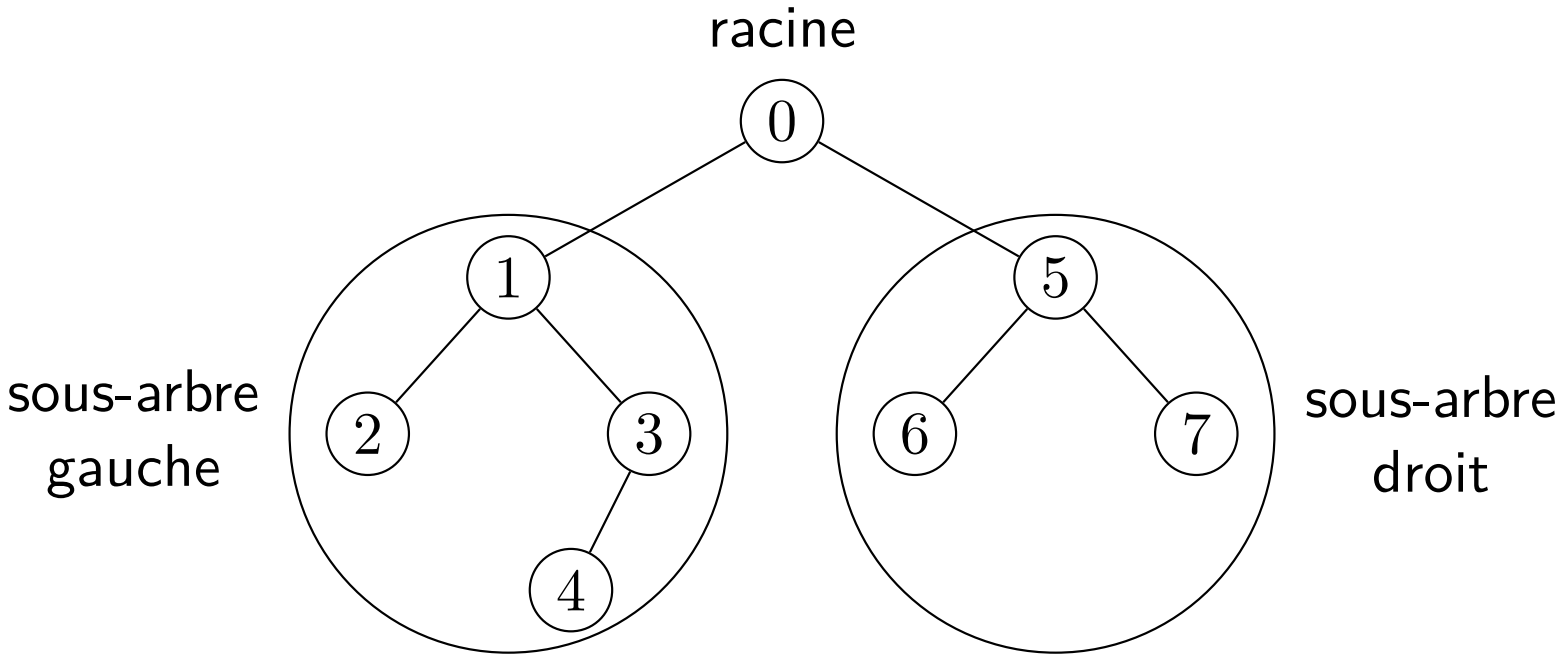
Un arbre binaire est :

- soit vide,
- soit constitué d'un **noeud** (la **racine**) et de deux sous-arbres binaires (gauche et droit)

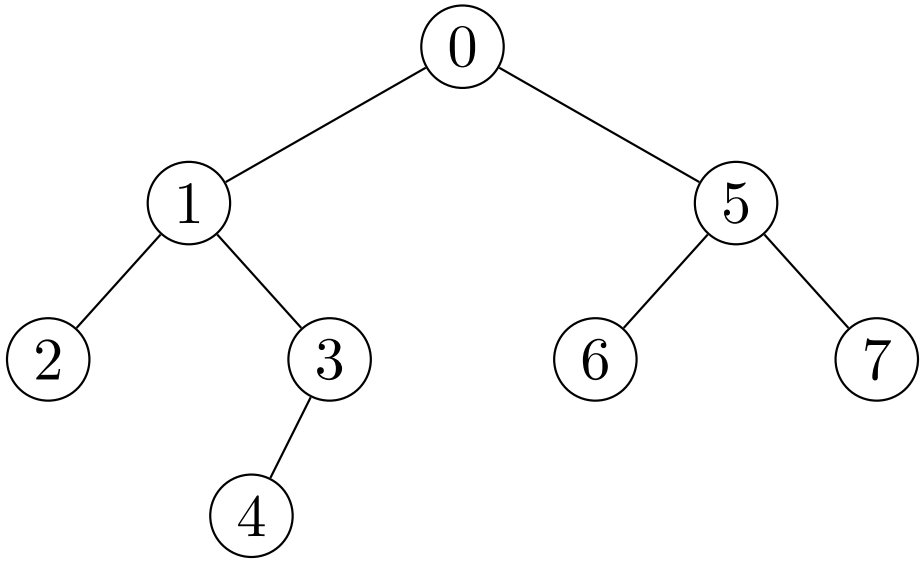
Arbre binaire



Arbre binaire



Arbre binaire



- 1 est le **père** de 2
- 2 est le **fils** de 1
- 2, 4, 6, 7 sont des **feuilles** (noeuds sans fils).
  Les autres noeuds sont des **noeuds internes**.

Arbre binaire

Exemple : arborescence des fichiers.

Arbre binaire : Nombre d'arêtes

Exercice

Soit un arbre binaire à n noeuds. Quel est son nombre d'arêtes (trait reliant un noeud à son fils) ?

Arbre binaire : Nombre d'arêtes

Exercice

Soit un arbre binaire à n noeuds. Quel est son nombre d'arêtes (trait reliant un noeud à son fils) ?

Réponse : $\boxed{n-1}$ (si $n \geq 1$) car la fonction qui à un noeud autre que la racine associe son père est une bijection.

Autre solution, par récurrence :

H_n : « un arbre binaire à $n \geq 1$ noeuds possède $n-1$ arêtes »

Arbre binaire : Nombre d'arêtes

Démontrons, par récurrence forte :

H_n : « un arbre binaire à $n \geq 1$ noeuds possède $n-1$ arêtes »

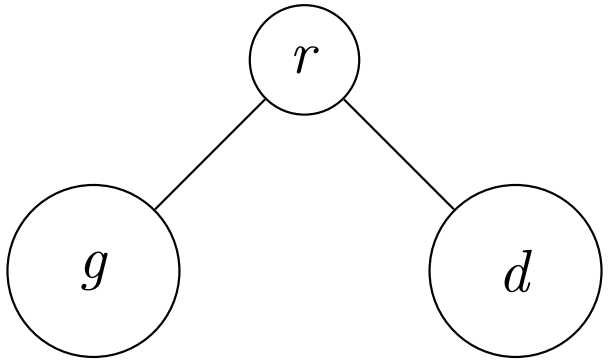
1. H_1 est clairement vraie.
2. Supposons H_n vraie et soit a un arbre binaire à $n+1$ noeuds.

Arbre binaire : Nombre d'arêtes

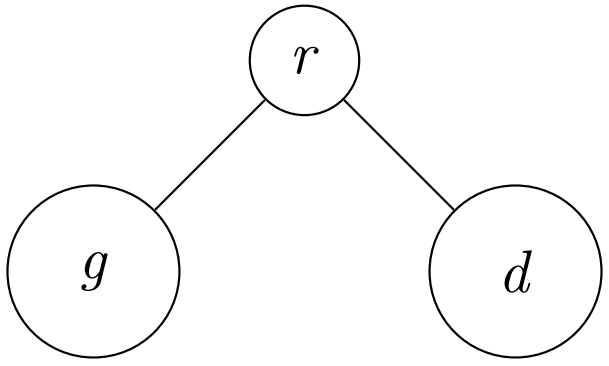
Démontrons, par récurrence forte :

H_n : « un arbre binaire à $n \geq 1$ noeuds possède $n-1$ arêtes »

1. H_1 est clairement vraie.
2. Supposons H_n vraie et soit a un arbre binaire à $n+1$ noeuds.
   a se décompose comme une racine r, un sous-arbre gauche g et un sous-arbre droit d :



Arbre binaire : Démonstration par récurrence



- Si $g = \emptyset$, alors d a n noeuds donc $n-1$ arêtes d'après H_n. Avec l'arête de r vers la racine de d, il y a donc bien $\boxed{n \text{ arêtes}}$ au total.
- De même si $d = \emptyset$.
- Sinon, soit k le nombre de noeuds de g. d possède alors $n+1-k$ noeuds. D'après H_k, g possède $k-1$ arêtes. D'après H_{n-k}, d possède $n-k-1$ arêtes. Donc a possède :

$$\underbrace{2}_{r} + \underbrace{k-1}_{g} + \underbrace{n-k-1}_{d} = \boxed{n \text{ arêtes}}$$

Arbre binaire : Démonstration par récurrence

Il est très courant de démontrer un théorème sur les arbres par récurrence (sur le nombre de noeuds ou la hauteur).

On peut aussi utiliser une **induction structurelle** pour démontrer une propriété $\mathcal{P}$ sur les arbres binaires. Il faut montrer :

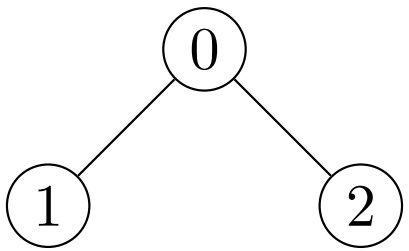
1. (cas de base) $\mathcal{P}$ est vraie sur l'arbre vide
2. (hérédité) $\mathcal{P}$ vraie pour g et d $\implies$ $\mathcal{P}$ est vraie pour un arbre de sous-arbre gauche g et de sous-arbre droit d.

Arbre binaire : Exemple en OCaml

```
1  type 'a binary_tree =
2       | E (* arbre vide *)
3       | N of 'a * 'a binary_tree * 'a binary_tree;;
```

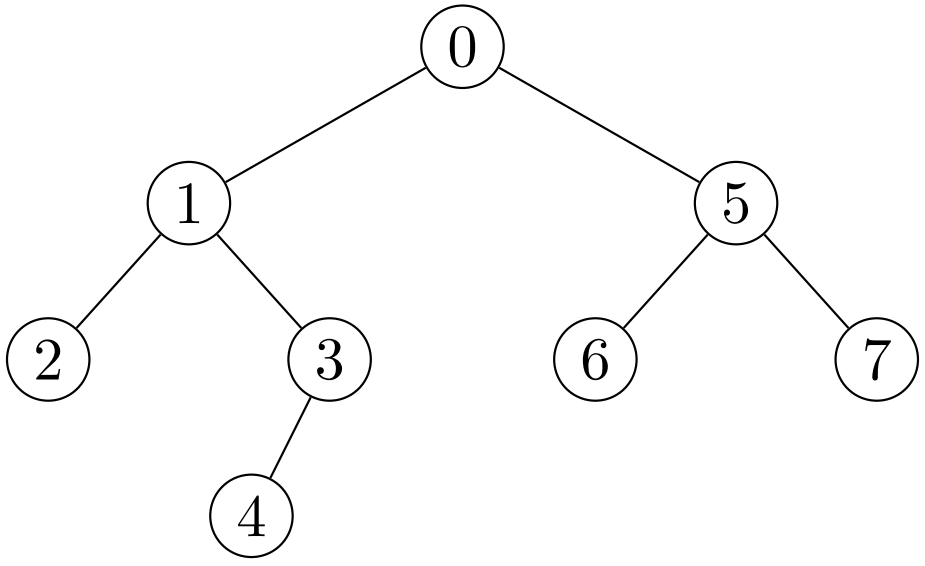
Arbre binaire : Exemple en OCaml

```
1  type 'a binary_tree =
2      | E (* arbre vide *)
3      | N of 'a * 'a binary_tree * 'a binary_tree;;
```




```
let a = N(0, N(1, E, E), N(2, E, E));;
```

Arbre binaire : Exemple en OCaml




```
1  let a_ex = N(0,
2      N(1,
3          N(2, E, E),
4          N(3, N(4, E, E), E)),
5      N(5,
6          N(6, E, E),
7          N(7, E, E)));;
```

Arbre binaire : Exemple en OCaml

Fonction pour renvoyer le nombre de noeuds d'un `binary_tree` :

```
1  let rec size t = match t with
2      | E -> 0
3      | N(r, g, d) -> 1 + size g + size d;;
```

Complexité :

Arbre binaire : Exemple en OCaml

Fonction pour renvoyer le nombre de noeuds d'un `binary_tree` :

```
1  let rec size t = match t with
2      | E -> 0
3      | N(r, g, d) -> 1 + size g + size d;;
```

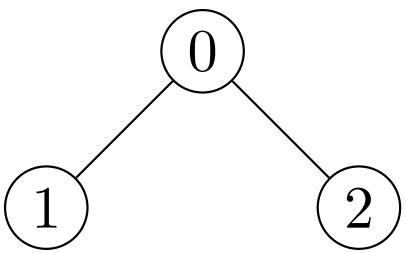
Complexité : O(n), avec n le nombre de noeuds.
En effet, il y a un appel récursif sur chaque noeud et chaque appel récursif se fait en O(1).

Arbre binaire : Exemple en C

```c
typedef struct tree {
    tree *g; // sous-arbre gauche (NULL si vide)
    tree *d; // sous-arbre droit (NULL si vide)
    int val;
} tree;
```

Arbre binaire : Exemple en C

```c
typedef struct tree {
    tree *g; // sous-arbre gauche (NULL si vide)
    tree *d; // sous-arbre droit (NULL si vide)
    int val;
} tree;
```

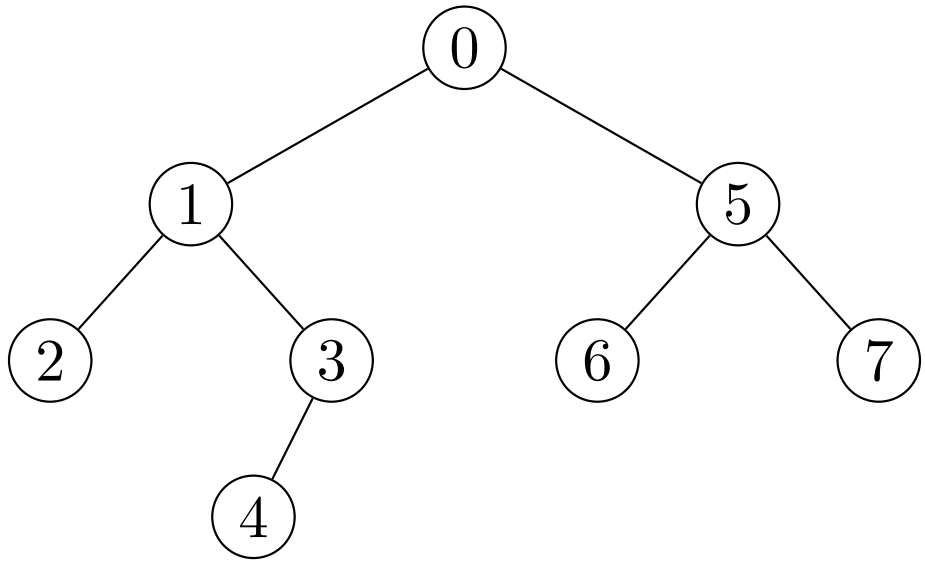



```c
tree* make_node(int r) {
    tree *t = malloc(sizeof(tree));
    t->g = t->d = NULL;
    t->val = r;
    return t;
}
tree* t = make_node(0);
t->g = make_node(1);
t->d = make_node(2);
```

Hauteur, profondeur

Définition

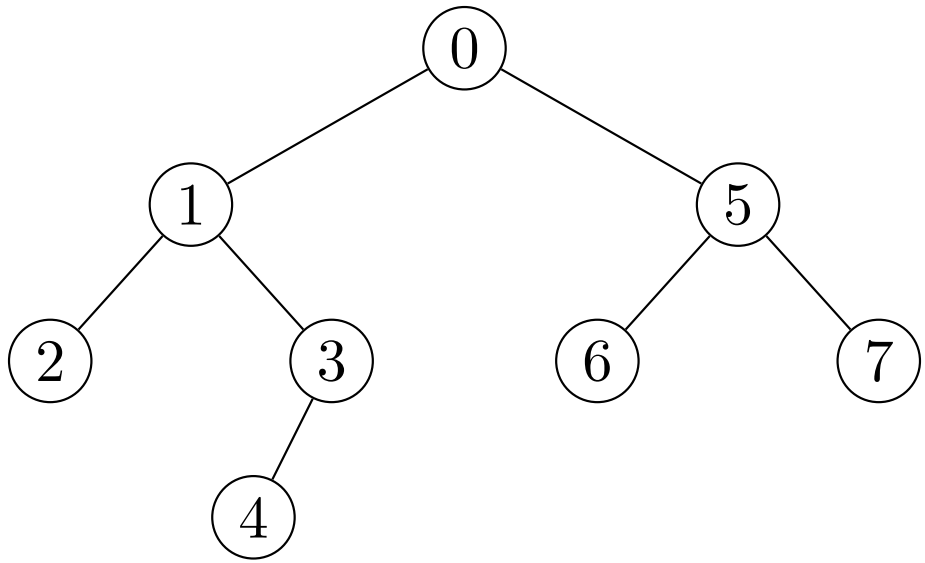
- La **profondeur** d'un noeud v est le nombre d'arêtes du chemin de la racine à v.
- La **hauteur** d'un arbre T est la profondeur maximale d'un noeud de T.
  Si T est vide, sa hauteur est -1.



Hauteur, profondeur

Définition

- La **profondeur** d'un noeud v est le nombre d'arêtes du chemin de la racine à v.
- La **hauteur** d'un arbre T est la profondeur maximale d'un noeud de T.
  Si T est vide, sa hauteur est -1.

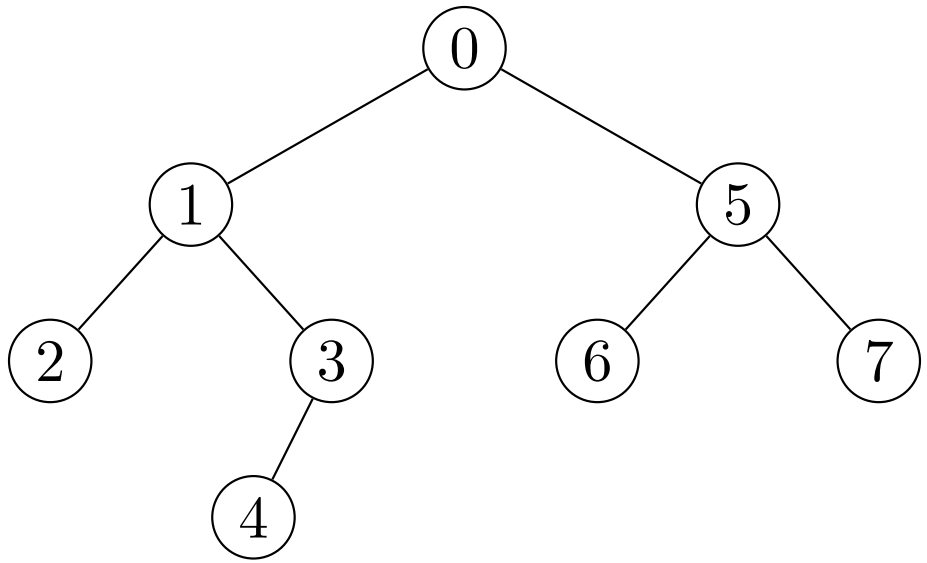


Profondeur du noeud 6 : 2

Hauteur, profondeur

Définition

- La **profondeur** d'un noeud v est le nombre d'arêtes du chemin de la racine à v.
- La **hauteur** d'un arbre T est la profondeur maximale d'un noeud de T.
  Si T est vide, sa hauteur est -1.

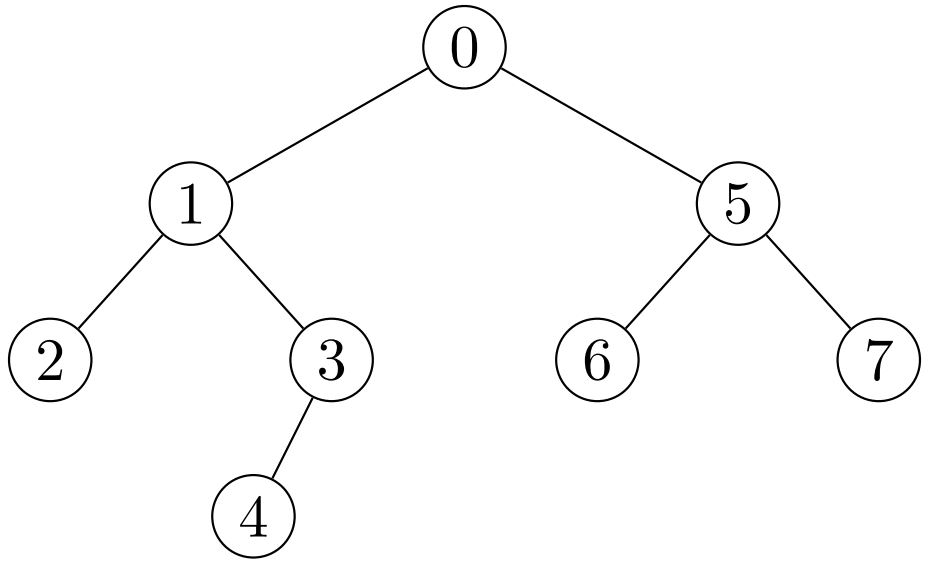


Profondeur du noeud 4 : 3

Hauteur, profondeur

Définition

- La **profondeur** d'un noeud v est le nombre d'arêtes du chemin de la racine à v.
- La **hauteur** d'un arbre T est la profondeur maximale d'un noeud de T.
  Si T est vide, sa hauteur est -1.

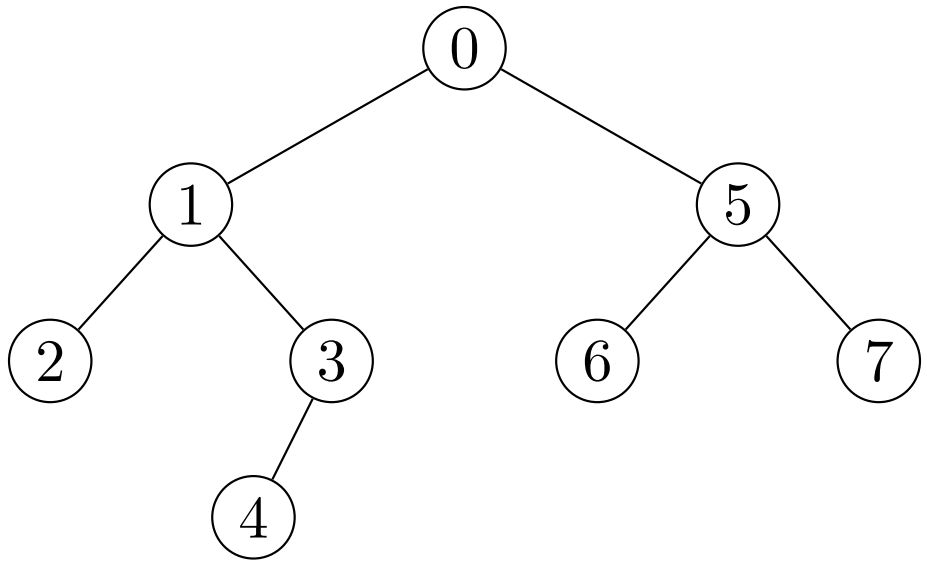


Profondeur du noeud 0 : 0

Hauteur, profondeur

Définition

- La **profondeur** d'un noeud v est le nombre d'arêtes du chemin de la racine à v.
- La **hauteur** d'un arbre T est la profondeur maximale d'un noeud de T.
  Si T est vide, sa hauteur est -1.



Hauteur : 3

Hauteur, profondeur

```
let rec height = function
    | E -> -1
    | N(r, g, d) -> 1 + max (height g) (height d);;
```

```
int max(int a, int b) { return a > b ? a : b; }

int height(tree* t) {
    if(t == NULL)
        return -1;
    return 1 + max(height(t->g), height(t->d));
}
```

Complexité :

Hauteur, profondeur

```
let rec height = function
    | E -> -1
    | N(r, g, d) -> 1 + max (height g) (height d);;
```

```
int max(int a, int b) { return a > b ? a : b; }

int height(tree* t) {
    if(t == NULL)
        return -1;
    return 1 + max(height(t->g), height(t->d));
}
```

Complexité : $O(n)$

Hauteur, profondeur

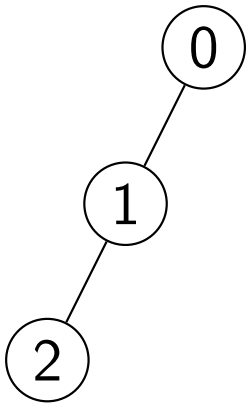
Exercice

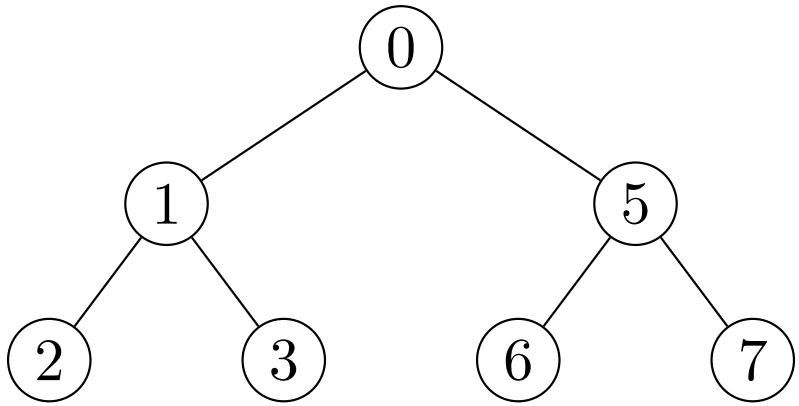
Quel est le nombre minimum de noeuds d'un arbre binaire de hauteur h ? Et le nombre maximum ?

Hauteur, profondeur

Exercice

Quel est le nombre minimum de noeuds d'un arbre binaire de hauteur h ? Et le nombre maximum ?





Minimum : $h+1$

Maximum : $\sum_{p=0}^{h} 2^p = 2^{h+1} - 1$

Théorème

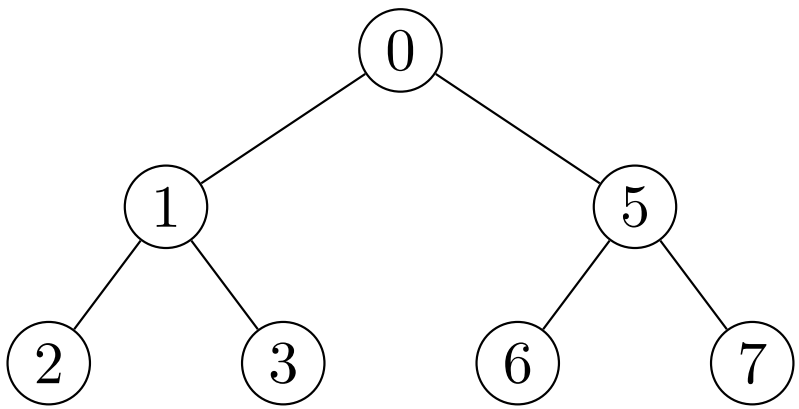
Le nombre n de noeuds et la hauteur h d'un arbre binaire vérifie :

$$\boxed{h+1 \leq n \leq 2^{h+1} - 1}$$

Arbre binaire complet

Définition

Un arbre binaire est **complet** si tous les niveaux sont remplis : il y a 2^p noeuds à profondeur p.
Le nombre total de noeuds est donc $\sum_{p=0}^{h} 2^p = 2^{h+1} - 1$.



Arbre binaire complet

Exercice

Écrire une fonction pour déterminer si un arbre binaire est complet.

```
let rec complet = function
    | E -> true
    | N(r, g, d) -> complet g && complet d
                    && height g = height d;;
```

Complexité : $O(n^2)$

Exercice

Réécrire cette fonction en $O(n)$.

Représentation d'un arbre avec un tableau

```
type 'a tree =
    | E
    | N of 'a * 'a tree * 'a tree;;
```

```
struct tree {
    tree *g;
    tree *d;
    int val;
};
```

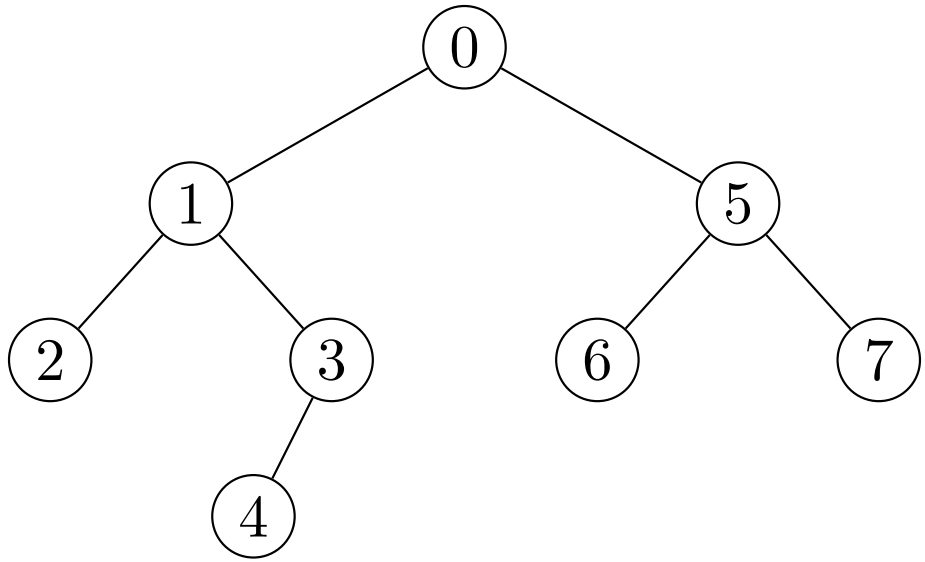
Cette représentation est pratique pour avoir accès aux fils, mais pas pour retrouver le père d'un noeud...

Représentation d'un arbre avec un tableau

Si on veut pouvoir obtenir le père d'un noeud, on peut numéroter les noeuds de 0 à $n-1$ utiliser un tableau `pred` tel que le $i$ème élément de `pred` donne le père du noeud i.

Représentation d'un arbre avec un tableau

Si on veut pouvoir obtenir le père d'un noeud, on peut numéroter les noeuds de 0 à $n-1$ utiliser un tableau `pred` tel que le $i$ème élément de `pred` donne le père du noeud i.




```
let pred = [|-1; 0; 1; 1; 3; 0; 5; 5|];;
```

Par convention, on peut mettre `-1` pour la racine.

Représentation d'un arbre avec un tableau

Exercice

En utilisant une représentation dans un tableau, écrire une fonction en C pour calculer la profondeur d'un sommet. Complexité ?

Représentation d'un arbre avec un tableau

Exercice

En utilisant une représentation dans un tableau, écrire une fonction en C pour calculer la profondeur d'un sommet. Complexité ?

Exercice

Écrire une fonction en OCaml pour renvoyer le chemin de la racine à un noeud donné.

Représentation d'un arbre avec un tableau : Cas binaire complet

On peut représenter un arbre binaire complet (donc aussi un tas max) par un tableau `t` tel que:

1. `t.(0)` est la racine
2. `t.(i)` a pour fils `t.(2*i + 1)` et `t.(2*i + 2)`, si ceux-ci sont définis.

Le père de `t.(j)` est donc

Représentation d'un arbre avec un tableau : Cas binaire complet

On peut représenter un arbre binaire complet (donc aussi un tas max) par un tableau `t` tel que:

1. `t.(0)` est la racine
2. `t.(i)` a pour fils `t.(2*i + 1)` et `t.(2*i + 2)`, si ceux-ci sont définis.

Le père de `t.(j)` est donc `t.((j - 1)/2)` (si `j` $\neq 0$)