ALGORITMIQUE

5. Les ARBRES BINAIRES

5.1. Définitions

Un arbre binaire est soit l'arbre vide, soit l'arbre formé d'une racine et de deux arbres binaires disjoints, appelés sous arbres gauche et sous arbre droit.

Exemples :

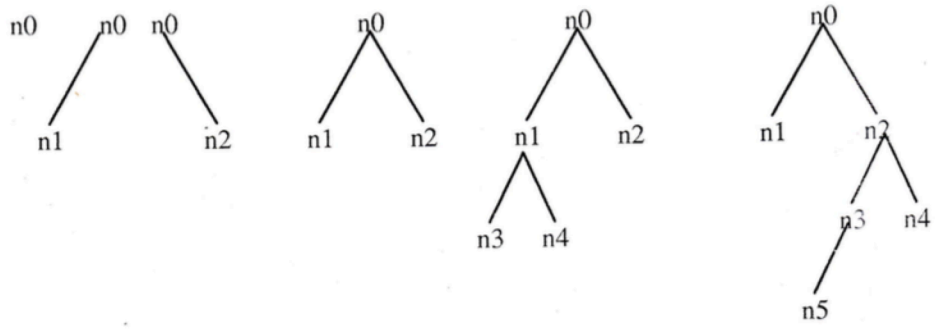


Figure 1. Un exemple d'arbres binaires.

On appelle fils gauche (resp. fils droit) d'un nœud la racine de son sous arbre gauche (resp. de son sous arbre droit).

Un nœud ni est dit le père d'un nœud nj si et seulement si le nœud nj est un des deux fils du nœud ni.

Les deux fils d'un nœud, s'ils existent, sont dits des frères.

Un nœud qui n'a ni fils droit ni fils gauche est dit une feuille.

Un nœud qui n'est pas une feuille est dit nœud interne ou intérieur.

On appelle chemin de n1 à nk une séquence de nœuds nl, n2, ... , nk telle que, ∀ i = l, 2, ... , k - 1, ni+1 est un fils droit ou gauche de ni.

Un nœud ni est dit un ascendant d'un nœud nj si et seulement si ni est soit le père de nj, soit un ascendant du père de nj.

Un nœud ni est dit un descendant d'un nœud nj si et seulement si ni est soit l'un des deux fils de nj, soit un descendant d'un fils de nj.

5.2. Mesures sur les arbres

Afin de pouvoir évaluer la complexité des algorithmes sur les arbres, on introduit quelques opérati6ns sur les arbres. Ces opérations sont, celles qui suivent.

La taille d'un arbre est le nombre de nœuds dans l'arbre.

Ainsi, taille (arbre vide)= 0 et taille (arbre non vide A) = 1 + taille (s-a-g(A)) + taille (S-a-d(A)).

La longueur d'un chemin est donnée par le nombre de nœuds présents dans le chemin auquel on retire 1. Ainsi, la longueur d'un chemin ne passant que par un seul nœud est égale 0.

La hauteur d'un nœud ni est donnée par la longueur du plus long chemin depuis ni jusqu'à une feuille.

La hauteur d'un arbre est donnée par la hauteur de la racine, c.-à-d., par la longueur du plus long chemin depuis la racine jusqu'à une feuille.

Ainsi hauteur (feuille) = 0 et hauteur (nœud interne ni) == 1 + max hauteur (nj),
nj ∈ { fils de ni}

La profondeur ou le niveau d'un nœud ni est donnée par la longueur du chemin depuis la racine jusqu'à ni.

Ainsi, profondeur (racine) = 0 et profondeur (nœud ni) = 1 + profondeur (père de ni).

La profondeur d'un arbre A est donnée par max profondeur (ni).
ni un nœud de A

La longueur de cheminement d'un arbre A est donnée par Σ profondeur (ni)
ni un nœud de A

La longueur de cheminement externe d'un arbre A est donnée Σ profondeur (fi)
fi une feuille de A

La longueur de cheminement interne d'un arbre A est donnée par Σ profondeur (ni)
ni un nœud interne de A

5.3. Arbres binaires particuliers

Un arbre binaire étiqueté est un arbre binaire dans lequel une étiquette ou une valeur est associée à chaque nœud de l'arbre.

Un arbre binaire dégénéré est un arbre formé uniquement de nœuds qui ont un seul fils. Par exemple, l'arbre de la figure 2 est un arbre binaire dégénéré.

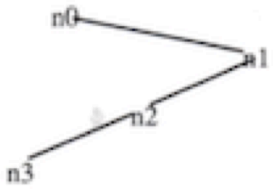


Figure 2. Un exemple d'un arbre binaire dégénéré.

Un arbre binaire est dit complet s'il contient 1 nœud au niveau 0, 2 nœuds au niveau 1, 4 nœuds au niveau 2, … , 2^h nœuds au niveau h. Dans un tel arbre, on dit que chaque niveau est complètement rempli. Par exemple, l'arbre de la figure 3 est un arbre binaire complet.

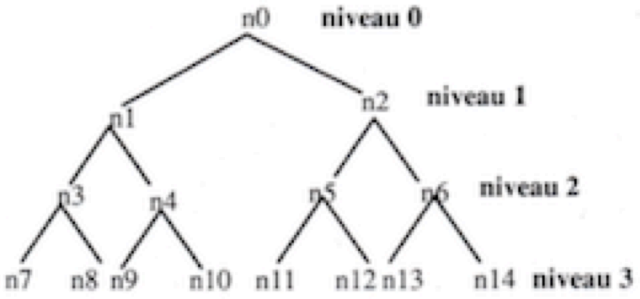


Figure 3. Un exemple d'un arbre binaire complet.

Ainsi, le nombre total de nœuds d'un arbre binaire complet de hauteur h est $n = 2^0 + 2^1 + 2^2 + \dots + 2^h = \sum_{=0}^{h} 2^i = (1 - 2^{h+1} / 1 - 2) = 2^{h+1} - 1$

Un arbre binaire parfait est un arbre binaire dont tous les niveaux sont complètement remplis, sauf éventuellement le dernier niveau, et dans ce cas, les nœuds (c.-à-d., les feuilles) du dernier niveau sont groupés le plus à gauche possible. Par exemple, l'arbre de la figure 4 est parfait. Par contre, les arbres de la figure 5 ne sont pas parfaits.

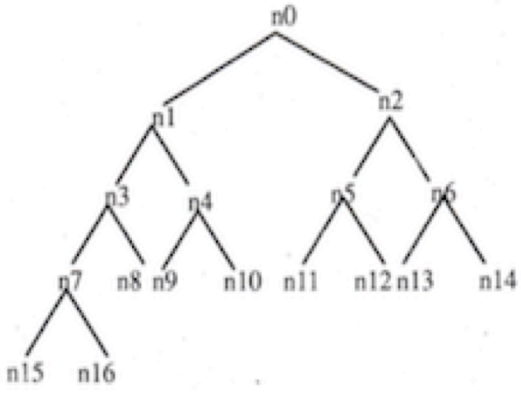


Figure 4. Un exemple d'un arbre binaire parfait.

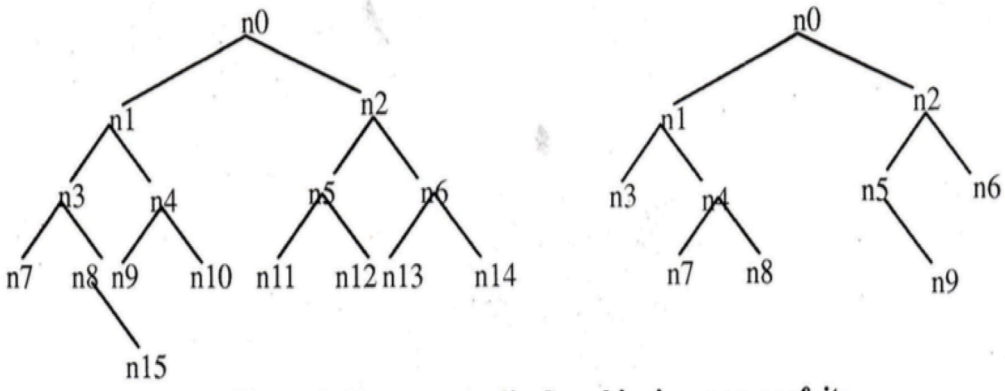


Figure 5. Un exemple d'arbres binaires non parfaits.

Un arbre binaire partiellement ordonné est un arbre binaire étiqueté par des éléments d'un ensemble totalement ordonné que l'on appelle des clés, et tel que la clé de tout nœud est inférieure ou égale aux clés des fils de ce nœud. Par exemple, l'arbre de la figure 6 est un arbre binaire partiellement ordonné. En fait, c'est même un arbre parfait partiellement ordonné.

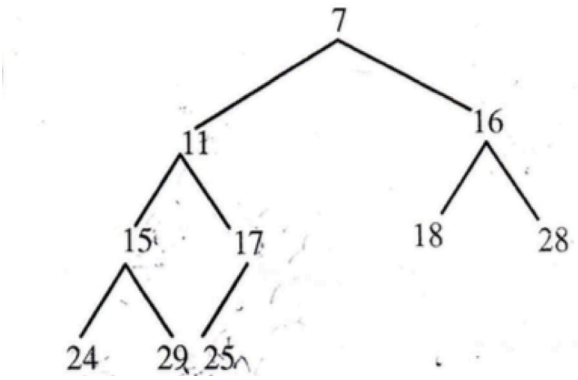


Figure 6. Un exemple d'un arbre parfait partiellement ordonné.

Un arbre binaire localement complet est un arbre binaire non vide dans lequel tous les nœuds autres que les feuilles ont deux fils. Par exemple, l'arbre de la figure 4 est un arbre binaire localement complet. Par contre, les arbres de la figure 5 et 6 ne sont localement complets.

5.4. Propriétés

Comme on le verra par la suite, souvent, la complexité des algorithmes sur les arbres s'exprime en fonction de la hauteur et de la taille. Aussi, dans ce paragraphe, on donnera quelques relations qui existent entre ces deux mesures.

Lemme : Pour un arbre binaire de taille n et de hauteur h on a $\lfloor \log_2(n) \rfloor \leq h \leq n - 1$.

Remarque : Un arbre binaire parfait de taille n a pour hauteur $\lfloor \log_2(n) \rfloor$.

Corollaire : Tout arbre binaire non vide ayant f feuilles a une hauteur h supérieure ou égale à $\lceil \log_2(f) \rceil$.

5.5. Récursivité sur les arbres

Il existe un bon nombre d'algorithmes sur les arbres binaires qui peuvent être décrits récursivement. Le schéma des récursivités sur les arbres binaires est le suivant.

Début

action a0 ;

appel récursif sur le sous arbre gauche ;

action a1 ;

appel récursif sur le sous arbre droit ;

action a2 ;

Fin

5.6. Les tas

On appelle tas un tableau T[l ... p] représentant un arbre parfait partiellement ordonné où p dénote le nombre de nœuds de l'arbre. Les clés des nœuds de l'arbre parfait partiellement ordonné apparaissent dans T niveau par niveau, en commençant par la racine et de la gauche vers la droite au sein de chaque niveau. Par exemple, la racine de l'arbre parfait partiellement ordonné est dans T[l], le fils gauche de la racine est dans T[2] et le fils droit de la racine est dans T[3].

En général, on a les règles qui suivent.

- La racine est dans T[l].

- Le père de T[i] est dans T[i / 2] pour i > 1.

- Les deux fils de T[i] sont dans T[2 * i] et T[2 * i + 1] si 2 * i < p. Par contre,

  - Si 2 * i = p, alors T[i) n'a qu'un seul fils situé en 2 * i = p, soit en T[p].
  - Si 2 * i > p, alors T[i] est une feuille.

Exemple

L'arbre de la figure 6 est représenté par le tableau :

i	1	2	3	4	5	6	7	8	9	10
T[i]	7	11	16	15	17	18	28	24	29	25

5.7. Représentation

La représentation d'un arbre binaire se fait comme suit.

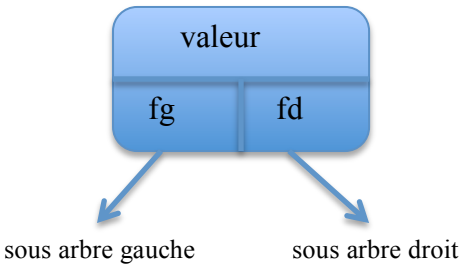
```
type Arbre = ^noeud

noeud = enregistrement

    etiquette_nœud : t

    fg,fd : Arbre

Fin ;
```



5.8. Parcours des arbres binaires

Le parcours d'un arbre consiste à visiter tous les nœuds de l'arbre et ce, tout en traitant chacun des nœuds une et une seule fois à un instant donné.

II existe deux stratégies de parcours d'un arbre, le parcours en profondeur et le parcours en largeur.

5.8.1. Parcours en profondeur

Soit A un arbre dont tous les nœuds sont initialement non marqués. Le principe du parcours en profondeur est le suivant. On part de la racine s de l'arbre, on la marque et on suit un chemin issu de s, aussi loin que possible, en marquant les nœuds au fur et à mesure qu'on les rencontre. Arrivée en fin de chemin, on revient au dernier choix fait, et on prend une autre direction.

Le parcours en profondeur peut se faire soit,

- ✓ En utilisant l'ordre préfixe dit également pré-ordre (On visite la racine, puis le sous arbre gauche et enfin le sous arbre droit);
- ✓ En utilisant l'ordre suffixe dit également post-ordre (On visite le sous arbre gauche, puis le sous arbre droit et enfin la racine) ;
- ✓ En utilisant l'ordre infixe dit également in-ordre (On visite le sous arbre gauche, puis la racine, et enfin le sous arbre droit).

```
Procédure préfixe (d a : Arbre) ;
        {…}
        Début

        Fin ;
```

Procédure suffixe (d a : Arbre) ;

{…}

Début

Fin ;

Procédure infixe (d a : Arbre) ;

{…}

Début

Fin ;

5.8.1. Parcours en largeur

Soit A un arbre dont tous les nœuds sont initialement non marqués. Appelons distance de s à x la longueur d'un plus court chemin de s à x. Le principe du parcours en largeur est le suivant. On choisit un nœud de départ s et on le marque. Ensuite, on marque tous les nœuds qui se trouvent à une distance 1 de s, puis tous ceux qui se trouvent à une distance

2 de s et ainsi de suite. Enfin, on réitère le processus et ce, jusqu'à ce que tous les nœuds de A soient marqués.

Ainsi, dans le parcours en largeur, on ne peut traiter les nœuds de niveau i+1 que si on a traité tous les nœuds du niveau i.

Aussi, le parcours en largeur s'appuie sur le principe de fonctionnement d'une file, à savoir, le FIFO.

```
Procédure largeur (dr a : Arbre) ;
        {…}
        var F : File
        Début
                Si a ≠ nil alors
                        initialiserfile (F)
                        enfiler (F, a) {on ajoute la racine de l'arbre à la file}
                        Tant que (filevide(F)=faux) faire
                                a := F^.valeur {a prend la valeur de la tête de la file}
                                defiler (F) {On supprime le contenu de la tête de la file}
                                Si a ≠ nil alors
                                        Affiche(a^. etiquette_nœud)
                                        Si a^. fg ≠ nil alors
                                                enfiler (F, a^. fg)
                                        Fin si
                                        Si a^. fd ≠ nil alors
                                                enfiler (F, a^. fd)
                                        Fin si
                                Fin si
                        Fin tant que
                Fin si

        Fin ;
```

4.9. Autres algorithmes

```
Procédure initialiserarbre (dr a : Arbre)
{On initialise l'arbre a à nil}
Début
        a :=nil
Fin ;
```

```
Fonction arbrevide (d a : Arbre) : booléenne;
{On teste si l'arbre a est vide ou pas}
Début
        arbrevide:= (a= nil)
Fin ;

Fonction creenoeud (d e: t) : Arbre;
{On crée un arbre de racine le nœud d'étiquette e}
var a : Arbre
Début

Fin ;

Fonction estfeuille (d a : Arbre) : booléenne;
{On teste si l'arbre a est une feuille ou pas}
Début

Fin ;
```

Procédure ajoutfg (dr a : Arbre, d e: t)
{On ajoute à l'arbre a un fils gauche d'étiquette e}

Début

Fin ;

Procédure ajoutfd (dr a : Arbre, d e: t)
{On ajoute à l'arbre a un fils droit d'étiquette e}

Début

Fin ;

Fonction taille (d a : Arbre) : entier;
{On calcul la taille de l'arbre}

Début

```
Si (a=nil) alors
        taille:= 0
Sinon
        taille:= 1 + taille(a^. fg) + taille(a^. fd)
```

Fin ;

Fonction nbfeuilles (d a : Arbre) : entier;
{On calcul le nombre de feuilles de l'arbre}

Début

Fin ;

Fonction calhauteur (d a : Arbre) : entier;

{On calcul la hauteur de l'arbre A cette fin on considère que le type Arbre contient en plus le champs hauteur qui est de type entier}

var hdroit : entier

Début

Fin ;

Fonction rechercher (d a : Arbre, d e :t) : booléenne;
{On retourne vrai si e est dans l'arbre a et faux sinon}

Début

Fin ;

Fonction dupliquearbre (d a: Arbre) : Arbre;
{On retourne l'arbre copie de l'arbre a}

var b : Arbre

Début

Fin ;