

Structures de données non linéaires

I. Graphes

Définition

Un graphe (simple) orienté G est un couple (S, A) , où :

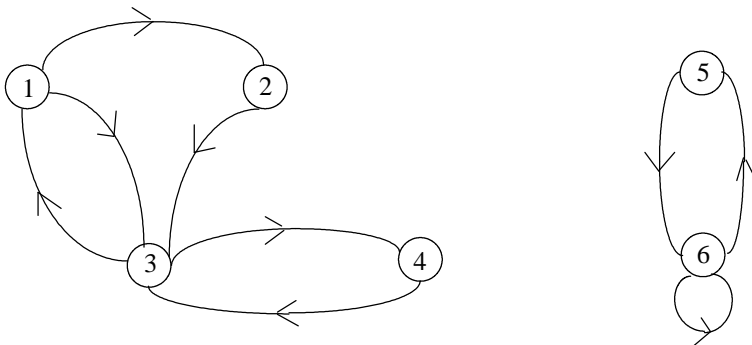
S est un ensemble dont les éléments sont appelés les sommets.

A est un ensemble de couples (ordonnés) d'éléments de S , ces couples sont appelés arcs.

Exemple 1

Soient $S = \{1,2,3,4,5,6\}$ et $A = \{(1;2),(1;3),(2;3),(3;1),(3;4),(4;3),(5;6),(6;5),(6;6)\}$.

(S, A) est un graphe orienté qui peut être représenté par :



Applications

Les graphes orientés peuvent servir à modéliser, entre autres :

- Un réseau routier à petite échelle : chaque intersection est un sommet, chaque tronçon de rue entre deux intersections est un arc. L'orientation peut alors permettre de gérer les sens interdits.
- Dans le même ordre d'idée : un réseau de bus.
- Un site web : chaque page est un sommet, chaque lien hypertexte est un arc (de la page qui le contient vers la page pointée).

Définition

Soit $G = (S, A)$ un graphe orienté. Si $X = (a,b) \in A$, on dit que :

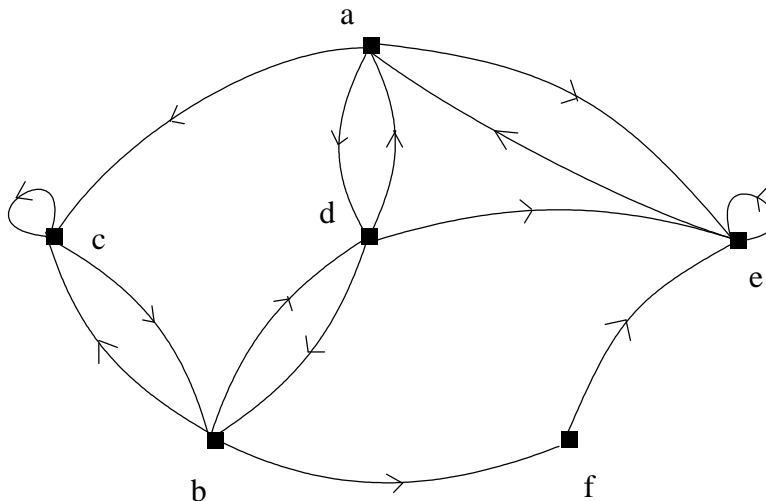
- a est adjacent à b
- a est un prédécesseur de b .
- b est un successeur de a .
- a est l'origine de l'arc X .
- b est l'extrémité de l'arc X .

De plus, si $a = b$, on dit que X est une boucle.

Exercice 1

Soient $S = \{a,b,c,d,e,f\}$ et $A = \{(a;c),(a;d),(a;e),(b;c),(b;d),(b;f),(c;b),(c;c),(d;a),(d;b),(d;e),(e;a),(e;e),(f;e)\}$.
Représenter le graphe orienté (S, A) .

Correction



Définition

On appelle chemin d'un graphe orienté G une suite (finie) d'arcs de G telle que l'extrémité d'un arc est toujours confondue avec l'origine du suivant.

L'origine du premier arc de la suite est appelé origine du chemin.

L'extrémité du dernier arc de la suite est appelé extrémité du chemin

Un chemin est dit simple si tous les arcs qui le composent sont différents.

Un chemin est dit élémentaire si tous les sommets qui le composent sont différents.

On appelle circuit tout chemin dont l'origine et l'extrémité sont confondues.

Exemple 2

En reprenant le premier exemple, on a :

- $\{(1,3);(3,1);(1,2)\}$ est un chemin simple non élémentaire d'origine 1 et d'extrémité 2.
- $\{(1,2);(2,3);(3,4)\}$ est un chemin simple et élémentaire.
- $\{(2;3),(3;4),(4;3),(3;1),(1;2)\}$ est un circuit simple et non élémentaire.

Exercice 2

En reprenant le graphe de l'exercice 1.

- Donner un chemin simple et élémentaire de a à f.
- Donner un chemin simple mais non élémentaire de a à f.
- Donner un circuit.

Correction

- $\{(a;d),(d;b),(b;f)\}$
- $\{(a;c),(c;c),(c;b),(b;f)\}$
- $\{(a;c),(c;b),(b;d),(d;a)\}$

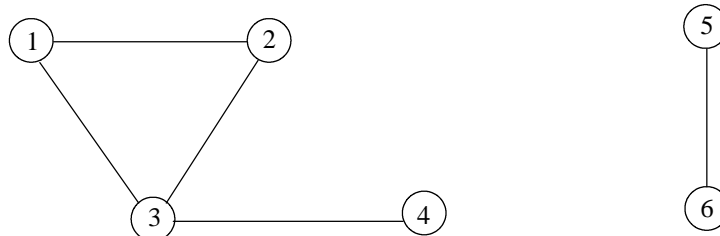
Définition

Un graphe (simple) non orienté G est un couple (S, A) , où :

S est un ensemble dont les éléments sont appelés les sommets.

A est un ensemble de paires (non ordonnées) d'éléments de S , ces paires sont appelées arêtes

Exemple 3



Applications

Les graphes non orientés peuvent servir à modéliser, entre autres :

- Un réseau ferré ou de métro.
- Un réseau routier à grande échelle : chaque ville est un sommet, chaque route entre deux villes est un arc (en général, elle n'est pas en sens unique).
- Un réseau informatique.

Définition

Soit $G = (S, A)$ un graphe non orienté.

Si $X = \{a, b\} \in A$, on dit que a et b sont voisins.

On appelle chaîne de G une suite (finie) d'arêtes de G telle que 2 arêtes consécutives dans la suite ont un sommet commun.

Un circuit est une chaîne dont l'origine et l'extrémité sont confondues.

On dit que G est connexe si et seulement si, pour toute paire de sommets $\{x, y\}$ de S , il existe une chaîne entre les sommets x et y .

Quand on parle de connexité pour un graphe orienté, on considère non pas ce graphe mais le graphe non-orienté correspondant.

II. Arbres

Définition 1 (source : wikipedia - Lexique de la théorie des graphes)

Un arbre est un graphe non orienté, acyclique et connexe.

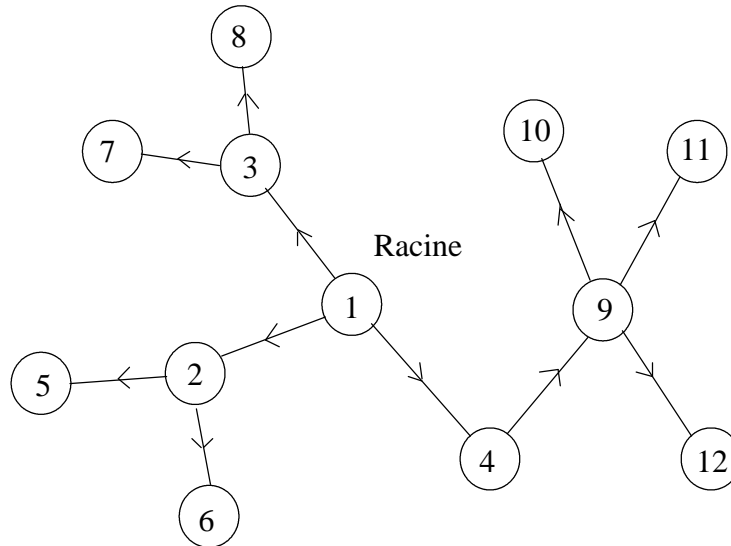
Définition 2 (source : wikipedia - Lexique de la théorie des graphes)

Une arborescence est un graphe orienté acyclique connexe où chaque sommet est de degré entrant au plus 1. Dans ce cas un seul sommet est de degré entrant 0 ; il est appelé racine de l'arborescence.

Définition 3 (celle que nous allons utiliser)

Un arbre est un ensemble de noeuds (appelés aussi parfois sommets) reliés par des arcs tel que chaque noeud possède exactement un arc pointant vers lui hormi un noeud spécial appelé racine qui n'en possède aucun. La racine est donc un noeud particulier puisqu'il n'a pas de prédécesseur. Les feuilles sont les noeuds sans successeur.

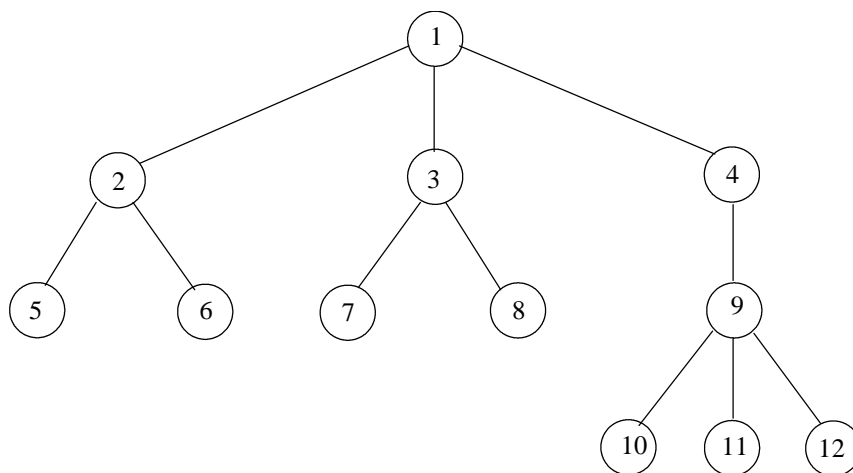
Exemple 4



Remarque

Pour avoir une meilleure vision de l'arbre et des algorithmes que nous allons utiliser, on représente un arbre avec la racine en haut et, pour chaque noeud, tous ses successeurs au même niveau. De plus, l'orientation des arcs étant définies, nous les représenterons de la même façon que les arêtes.

Cela donne pour l'exemple précédent :



Exemple 5

L'arborescence des fichiers et des dossiers dans Windows est un arbre.

Remarque

Chaque nœud possède une étiquette, qui est en quelque sorte la «valeur» du nœud. L'étiquette peut être très simple : un nombre entier, un nom, etc. Mais aussi plus complexe : un objet, un ou plusieurs pointeurs, etc.

Remarque

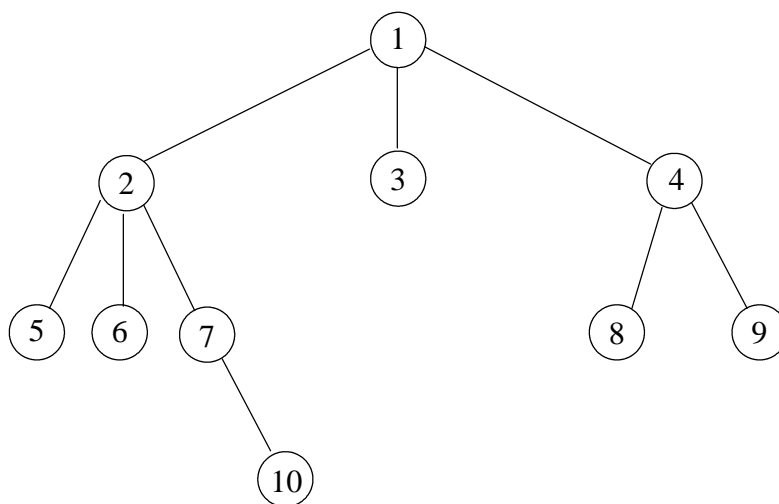
Un arbre peut être aussi défini par récurrence. On peut dire qu'un arbre est une structure formée :

- d'un élément appelé racine
- et d'un ensemble d'arbres de même type, aussi appelés sous-arbres.

Définitions

- # La profondeur d'un nœud est la longueur du chemin de la racine à ce nœud.
- # Le niveau p est l'ensemble des nœuds de profondeur p .
- # La hauteur d'un arbre est la profondeur maximale des feuilles. C'est-à-dire la longueur du plus grand chemin de la racine à une feuille.
- # Par analogie à l'arbre (au sens naturel du terme), on parlera de :
 - racine : l'élément qui n'a pas de prédécesseur dans l'arbre
 - feuilles : tout élément qui n'a pas de successeur de l'arbreet par analogie à l'arbre généalogique on parlera de :
 - père : prédécesseur
 - fil : successeur
 - frères : nœuds qui possèdent le même prédécesseur.

Exemple 6



Dans cet arbre :

- 1 est la racine.
- 5, 6, 8, 9, 10 sont des feuilles.
- 1, 2, 3, ..., 9, 10 sont des nœuds
- 4, 8, 9 appartiennent à la même branche.

De même on dira que :

- 2 est le père de 6.
- 10 est le fils de 7.
- 2, 3, 4 sont frères.

Remarque

Les arbres sont rarement utilisés en tant que tels. De nombreux types d'arbres avec une structure plus restrictive existent. Ils sont couramment utilisés en algorithmique car ils permettent des recherches plus efficaces. Les principaux exemples sont des arbres binaires.

III. Arbres binaires

1. Généralités

Définition

Un arbre binaire est un arbre dans lequel chaque nœud possède au plus deux fils.

Un arbre binaire entier est un arbre dont tous les nœuds possèdent zéro ou deux fils.

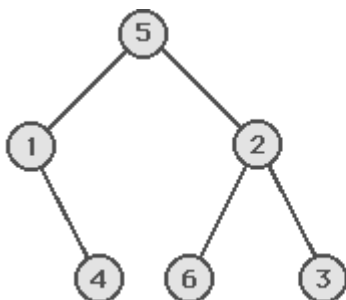
Un arbre binaire parfait est un arbre binaire entier dans lequel toutes les feuilles (nœuds n'ayant aucun fils) sont à la même distance de la racine.

Un arbre équilibré est un arbre dont les sous-arbres ont environ la même hauteur.

Remarque

Dans un arbre binaire, le fils unique d'un nœud ne sera jamais placé à l'aplomb de son père, mais toujours dirigé soit vers la gauche soit vers la droite.

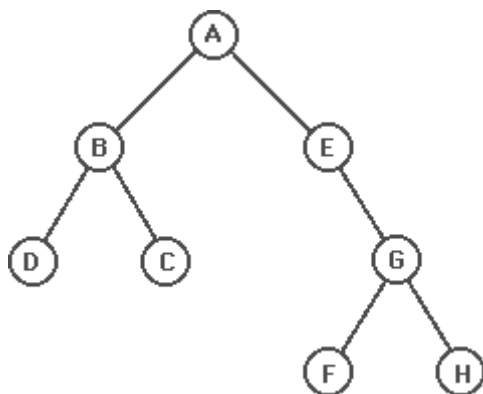
Par exemple :



On parlera alors de fils droit et de fils gauche ou encore de sous arbre droit et de sous arbre gauche (dans l'exemple : 4 et 1 sont dans le même sous arbre gauche issu de 5)

2. Implantation en mémoire

On considère l'arbre binaire suivant :



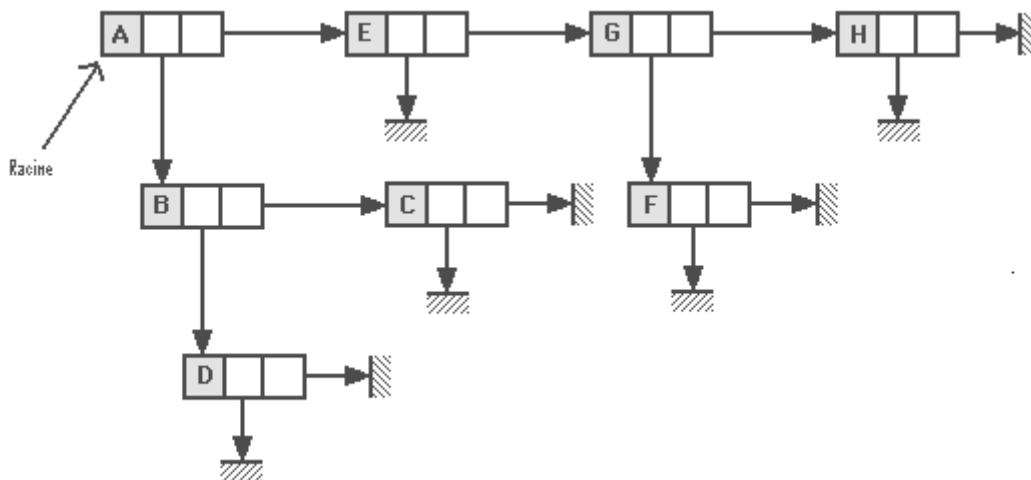
Pour un noeud quelconque de l'arbre binaire, les seules informations nécessaires sont :

- l'information sur son nom.
- les informations concernant ses fils.

On peut alors décider que, pour chaque noeud, on construit un article du type :

Nom du noeud	adresse du Fils gauche	adresse du Fils droit
--------------	------------------------	-----------------------

On obtient un chaînage. Ce qui donne pour notre arbre :



Il pourra être implanté avec des pointeurs (comme les listes doublement chaînées) mais aussi sous forme de tableau à 2 dimensions :

AdrRac	1	2	3	4	5	6	7	8
NœUD	B	A	C	D	E	F	G	H
ADR FILSG	4	1	0	0	0	0	6	0
ADR FILSD	3	5	0	0	7	0	8	0

Exercice 3

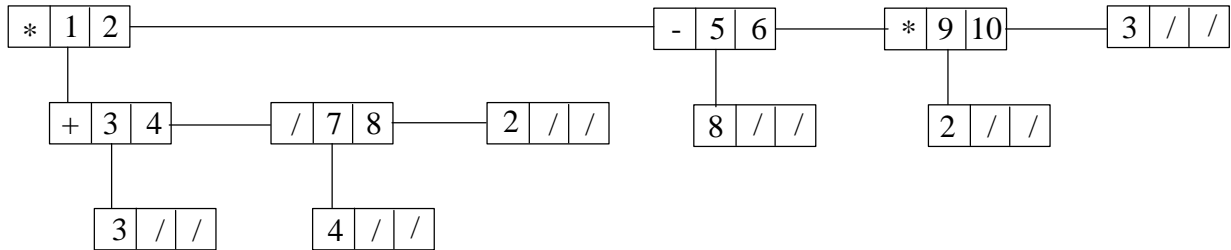
1. Soit l'arbre défini par le tableau suivant et dont la racine est au rang 0 :

Rang	Valeur	gauche	droit
0	*	1	2
1	+	3	4
2	-	5	6
3	3	/	/
4	/	7	8
5	8	/	/
6	*	9	10
7	4	/	/
8	2	/	/
9	2	/	/
10	3	/	/

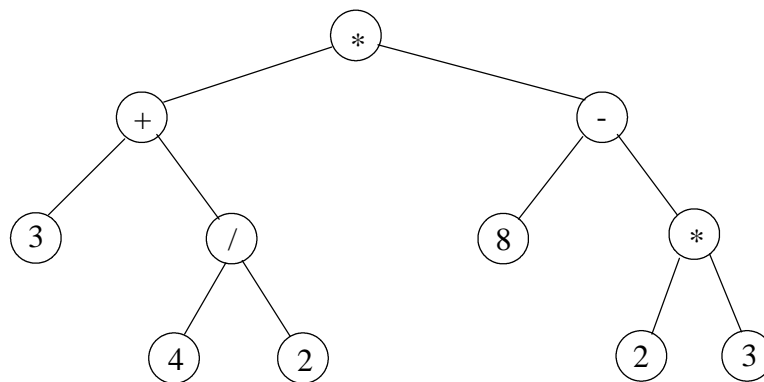
- a. Le représenter sous forme de chaînage.
 - b. Le représenter sous forme d'arbre.
2. Quelle est sa hauteur?

Correction

1. a.



- b.

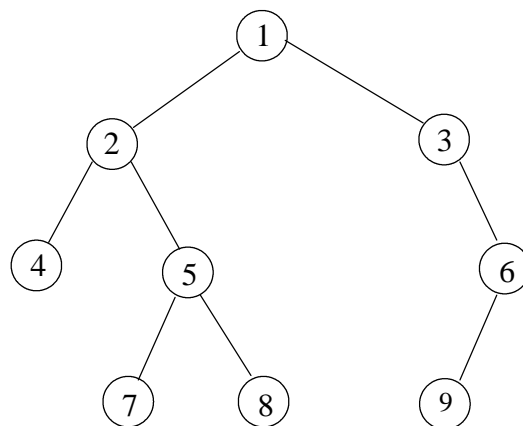


2. La hauteur est de 3.

3. Parcours d'un arbre binaire

Lorsque l'on désire connaître les informations contenues dans un arbre (valeurs ou contenus de ses noeuds), cela nécessite de parcourir (ou visiter) celui-ci. Pour cela, il existe différentes méthodes pour effectuer la visite d'un arbre binaire.

Prenons comme exemple pour les parcours d'arbre, celui-ci :



Rappelons qu'un arbre a est considéré comme une racine (dans l'exemple ci dessus, elle a pour valeur 1) et deux sous arbres. Le premier est identifié à gauche (a) (dans l'exemple, la racine de ce sous arbre a pour valeur 2) et le deuxième est identifié à droite (a) (de valeur racine 3). Ce qui signifie que connaître le noeud racine de l'arbre, c'est avoir l'arbre.

a. Parcours en largeur

Le parcours en largeur correspond à un parcours par niveau de nœuds de l'arbre. L'ordre de parcours d'un niveau donné est habituellement conféré, de manière récursive, par l'ordre de parcours des nœuds parents c'est-à-dire des nœuds du niveau immédiatement supérieur.

Ainsi, pour l'arbre d'exemple, le parcours en largeur sera 1 2 3 4 5 6 7 8 9.

Le parcours en profondeur est un parcours récursif sur un arbre. Il existe trois ordres pour cette méthode de parcours.

b. Parcours en profondeur préfixé ou visite preorder

Dans ce mode de parcours, le nœud courant est traité (prise en compte de l'information) avant le traitement des sous-arbres gauches et droits.

Ainsi, pour l'arbre d'exemple, le parcours en preorder sera 1, 2, 4, 5, 7, 8, 3, 6, 9.

On identifie un arbre à sa racine. Le parcours peut alors s'écrire de façon récursive :

```
parcoursPrefixe(arbre a)
{
    lire(a);
    si (gauche(a) != NIL)
        parcoursPrefixe(gauche(a));
    si (droite(a) != NIL)
        parcoursPrefixe(droite(a));
}
```

c. Parcours en profondeur infixé ou visite inorder

Dans ce mode de parcours, le nœud courant est traité entre le traitement des nœuds gauches et droits.

Ainsi, pour l'arbre d'exemple, le parcours en inorder sera 4, 2, 7, 5, 8, 1, 3, 9, 6.

Cela peut s'écrire de façon récursive :

```
parcoursInfixe(arbre a)
{
    si (gauche(a) != NIL)
        parcoursInfixe(gauche(a));
    lire(a);
    si (droite(a) != NIL)
        parcoursInfixe(droite(a));
}
```

c. Parcours en profondeur suffixé ou visite postorder

Dans ce mode de parcours, le nœud courant est traité après le traitement des nœuds gauches et droits.

Ainsi, pour l'arbre d'exemple, le parcours en postorder sera 4, 7, 8, 5, 2, 9, 6, 3, 1.

Cela peut s'écrire de façon récursive :

```
parcoursPostfixe(arbre a)
{
    si (gauche(a) != NIL)
        parcoursPostfixe(gauche(a));
    si (droite(a) != NIL)
        parcoursPostfixe(droite(a));
    lire(a);
}
```

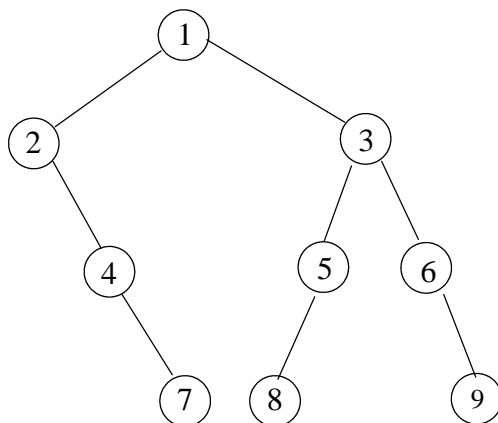
Ce mode de parcours correspond à une notation polonaise inversée.

Remarques

Le sens de parcours de l'arbre binaire est le même dans les trois cas. Seul change, dans la visite, l'instant de prise en considération de l'information au moment du parcours.

Ces algorithmes récursifs de parcours font appel à une pile pour «remonter» au noeuds parents.

Exemple 7



Dans cet arbre binaire,

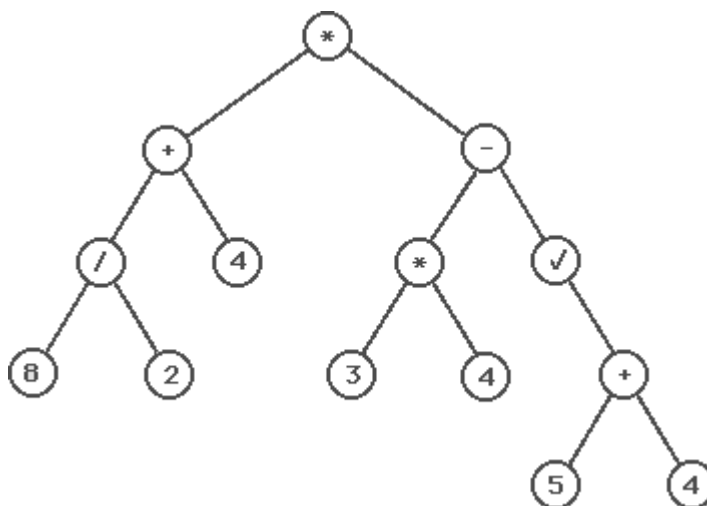
La visite de l'arbre en préfixé donne : 1 2 4 7 3 5 8 6 9

La visite de l'arbre en infixé donne : 2 4 7 1 8 5 3 6 9

La visite de l'arbre en postfixé donne : 7 4 2 8 5 9 6 3 1

Exemple 8

Dans l'arbre suivant, les informations (contenu des noeuds) sont des opérations, fonctions, nombres ou identificateurs.



Observons alors les résultats obtenus après utilisation des différentes méthodes de parcours sur un tel arbre binaire.

Ainsi, après un parcours preorder, on obtient : * + / 8 2 4 - * 3 4 + 5 4

Après un parcours inorder, on obtient : 8 / 2 + 4 * 3 * 4 - √ 5 + 4

Enfin, après un parcours postorder, on obtient : 8 2 / 4 + 3 4 * 5 4 + √ - *

Ajoutons les règles suivantes pour les lectures preorder et inorder :

- lorsque l'on descend dans un sous arbre, on ouvre une parenthèse
- lorsque l'on remonte d'un sous arbre, on ferme une parenthèse

On obtient alors :

Pour preorder, $* (+ (/ (8) (2)) (4)) (- (* (3) (4)) (\sqrt{\quad} (+ (5) (4)))$

Pour inorder, $(((8) / (2)) + (4)) * (((3) * (4)) - (\sqrt{\quad} ((5) + (4))))$ i.e. $(\frac{8}{2} + 4) \times (3 \times 4 - \sqrt{5+4})$.

On s'aperçoit alors que si l'on interprète le résultat obtenu lors de la visite inorder, comme étant l'écriture d'une expression algébrique (écriture dite infixée); le résultat obtenu après une lecture preorder correspond à l'écriture préfixée (sous forme de liste) de la même expression et le résultat de la lecture postorder correspond à l'écriture postfixée de l'expression

Rappelons à titre d'exemple que le langage FORTH utilise l'écriture postfixée sous cette forme et que le langage LISP utilise l'écriture préfixée.

Exercice 4

Donner les résultats des parcours infixé et postfixé de l'exercice 3 en respectant les règles sur le parenthésage dans le cas infixé.

Correction

Infixé : $(((3 + (4 / 2)) * (8 - (2 * 3)))$

Postfixé : $3 4 2 / + 8 2 3 * - *$

Exercice 5

Trouver une fonction qui donne la hauteur d'un arbre binaire.

Correction

```
hauteur(arbre a)
{
  si (a == nil)
    renvoie 0,
  sinon
    renvoie de max(hauteur(gauche(a)) + hauteur(droite(a)) + 1;
}
```

4. Arbres binaires de recherche

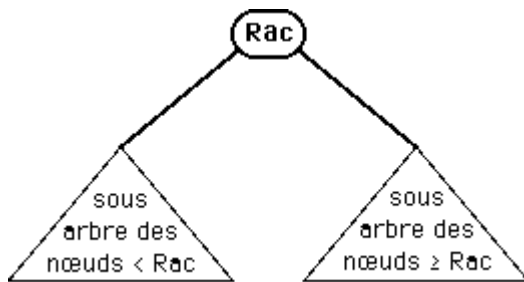
a. Généralités

Définition

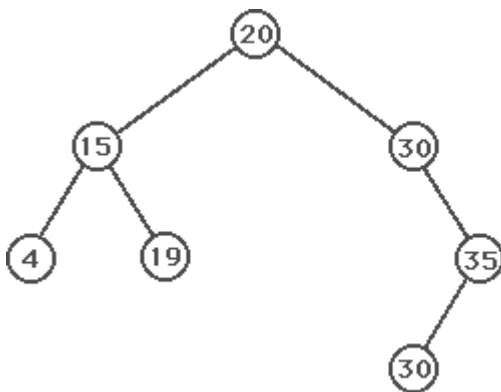
Un arbre binaire de recherche est un arbre binaire possédant récursivement la propriété suivante :

Les noeuds du sous arbre gauche sont tous inférieurs à la racine, elle-même inférieure aux noeuds du sous arbre droit (les éléments égaux pouvant être arbitrairement placés à gauche ou à droite (choix à faire pour un algorithme d'insertion)).

Ce qui peut se visualiser de la façon suivante :



Exemple 9



Remarque

Ainsi que son nom l'indique, cet arbre est particulièrement utile pour résoudre les problèmes de recherche d'un noeud dans un arbre.

En effet, si X est l'élément cherché, il suffit de comparer celui-ci avec la racine pour éliminer tous les candidats se trouvant dans le sous arbre gauche ou dans le sous arbre droit (en fonction du résultat du test avec la racine). C'est évidemment plus intéressant si l'arbre est équilibré.

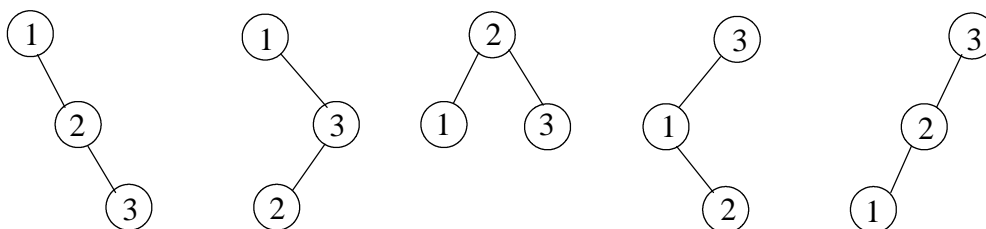
Remarque

En prenant comme relation d'ordre, celui de l'alphabet ou l'ordre lexicographique, on peut considérer des arbres de recherche dont les sommets sont des lettres ou des mots.

Exercice 6

Donner tous les arbres binaires de recherche possédant les trois noeuds de valeurs 1, 2 et 3?

Correction



Exercice 7

Donner un algorithme qui détermine le maximum d'un arbre binaire de recherche abr.

Correction

```
p = abr;  
tant que (droite(p) != NIL)  
    p = droite(p);  
max = valeur(p);
```

b. Création d'un arbre binaire de recherche

Soit une suite $S[1], S[2], \dots, S[N]$ d'éléments. On désire, à partir de ces éléments, créer un arbre binaire de recherche.

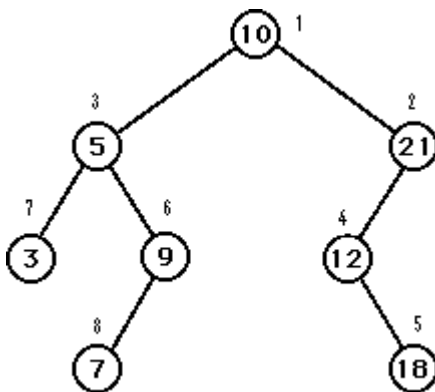
Principe

On réalise alors des insertions successives de chacun des éléments dans un arbre binaire de la manière suivante :

- $S[1]$ (premier élément considéré) est la racine de l'arbre
- si $S[i]$ est inférieur à la racine alors on insère $S[i]$ dans le sous arbre gauche.
- si $S[i]$ est supérieur ou égal à la racine alors on insère $S[i]$ dans le sous arbre droit.

Exemple 10

On considère la suite d'éléments suivantes : 10 , 21 , 5 , 12 , 18 , 9 , 3 , 7. On crée, par l'insertion successive des éléments, l'arbre binaire de recherche suivant :

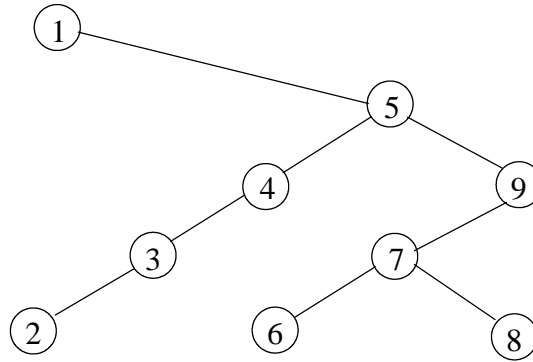


Exercice 8

- Construire, par adjonction successive aux feuilles, un arbre binaire de recherche à partir de la suite 1, 5, 9, 4, 3, 2, 7, 6, 8.
 - Quels sont les parcours préfixé, infixé et postfixé de cet arbre?
- Mêmes questions avec la suite 3, 1, 9, 2, 7, 4, 6, 5, 8.
- Mêmes questions avec la suite 3, 6, 7, 9, 2, 5, 8, 4, 1.
- Que remarquez-vous avec les parcours infixés?

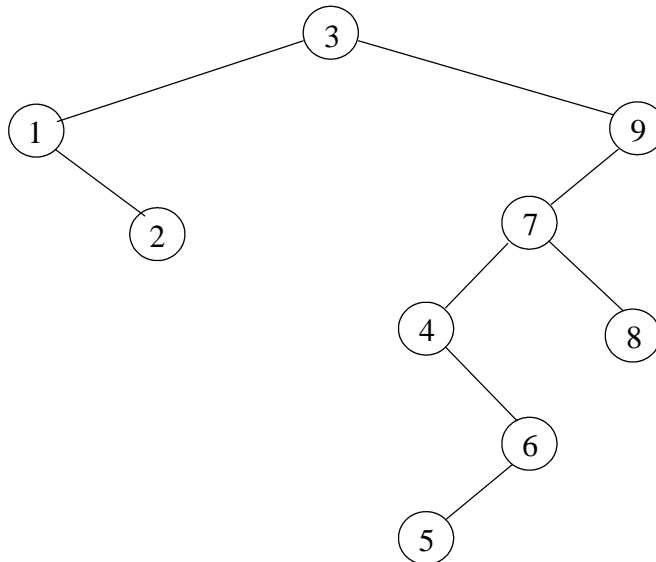
Correction

1. a. 1, 5, 9, 4, 3, 2, 7, 6, 8



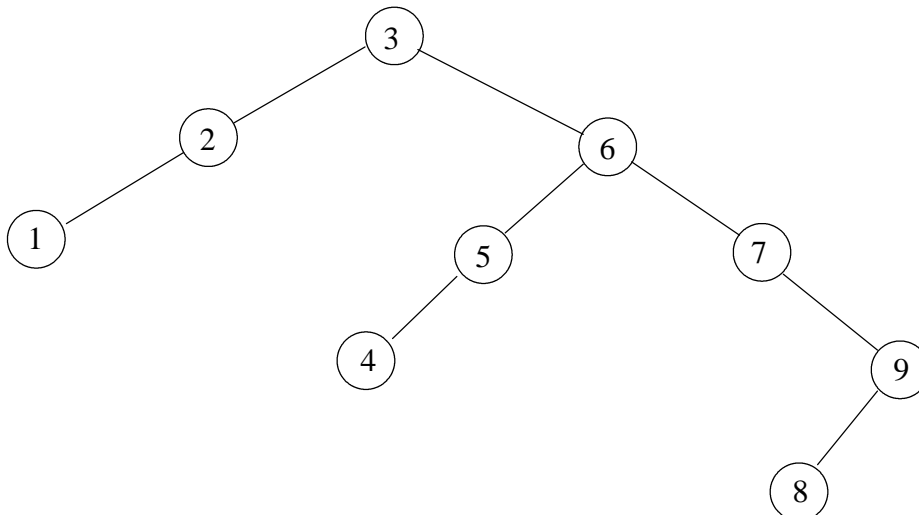
- b. Le parcours préfixé : 1 5 4 3 2 9 7 6 8
Le parcours infixé : 1 2 3 4 5 6 7 8 9
Le parcours postfixé : 2 3 4 6 8 7 9 5 1

2. a. 3, 1, 9, 2, 7, 4, 6, 5, 8



- b. Le parcours préfixé : 3 1 2 9 7 4 6 5 8
Le parcours infixé : 1 2 3 4 5 6 7 8 9
Le parcours postfixé : 2 1 5 6 4 8 7 9 3

3. a. 3, 6, 7, 9, 2, 5, 8, 4, 1



- b. Le parcours préfixé : 3 2 1 6 5 4 7 9 8
 Le parcours infixé : 1 2 3 4 5 6 7 8 9
 Le parcours postfixé : 1 2 4 5 8 9 7 6 3
4. Les parcours sont égaux. Ils correspondent à une liste ordonnée des éléments de l'arbre. Comme ces trois mots sont composés des mêmes lettres, les parcours donnent le même classement.

Algorithme

Nous considérerons à nouveau un arbre de façon récursive : il s'agit d'un noeud a (adresse de l'information) et de deux sous-arbres (le gauche et le droit) pointé par $gauche(a)$ et $droite(a)$. Un arbre abr est donc identifié à sa racine $racine(abr)$.

Un noeud sera une feuille si ses deux sous-arbres sont vides (pointant vers NIL).

L'algorithme d'insertion d'une valeur pointée par p dans un arbre binaire de recherche abr sera le suivant : (dans l'appel initial $pere$ est à NIL)

```

insérerArbreBR(noeud p, arbre abr, noeud pere)
{
  si (abr == NIL)
  {
    abr = p;
    si ((pere != NIL)
        si ((valeur(p) < valeur(pere)))
            gauche(pere) = p;
        sinon
            droite(pere) = p;
    }
  sinon
    si (valeur(p) < valeur(abr))
      insérerArbreBR(p, gauche(abr), abr);
    sinon
      insérerArbreBR(p, droite(abr), abr);
  }
}

```

Remarque

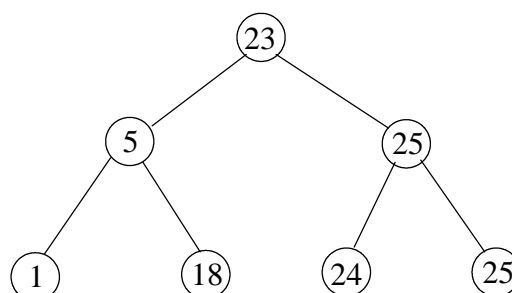
Comme nous l'avons vu dans l'exercice 8, avec les mêmes valeurs, on peut construire différents arbres binaires de recherche.

On pourrait donc s'intéresser à l'équilibrage de ces arbres.

Exemple 11

Soit la liste triée : 1; 5; 18; 23; 24; 25; 25.

Une construction d'un arbre binaire de recherche de hauteur minimale contenant ces valeurs est :



Exercice 9

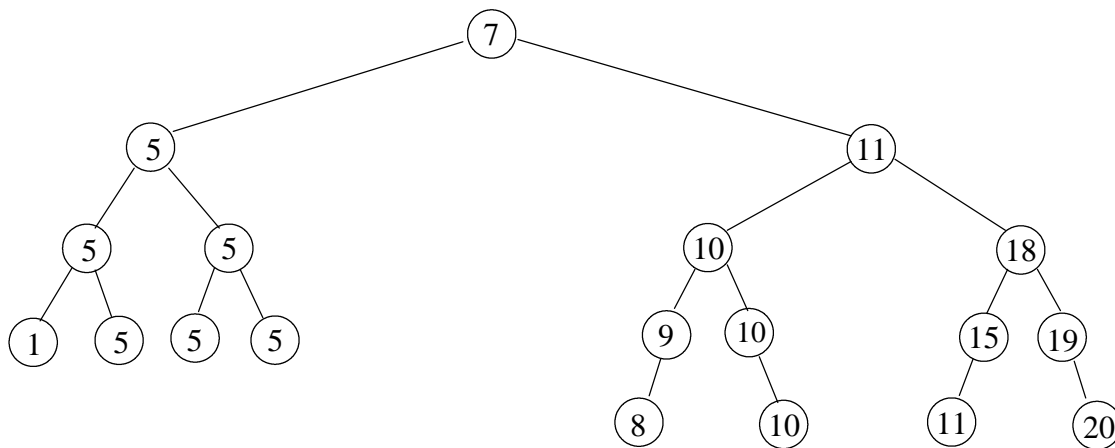
Soit la liste triée 1;5;5;5;5;5;5;7;8;9;10;10;10;11;11;15;18;19;20 correspondant aux notes d'une promotion d'étudiants.

Construire un arbre binaire de recherche de hauteur minimale contenant ces valeurs.

Correction

Puisqu'il y a 19 valeurs, la puissance de 2 strictement supérieur à 19 et qui lui est la plus proche est 32. La hauteur minimale théorique est donc 4 car $19 \leq 31 = 32 - 1 = 2^5 - 1$ qui est le nombre d'éléments d'un arbre binaire complet de hauteur 4.

On obtient par exemple :



c. Suppression d'un noeud

Plusieurs cas sont à considérer, mais à chaque fois, il faut connaître l'adresse du père (initialisé à l'adresse de la racine).

Suppression d'une feuille : On peut de l'enlever de l'arbre vu qu'elle n'a pas de fils.

Suppression d'un nœud avec un enfant : On peut l'enlever de l'arbre en le remplaçant par son fils.

Suppression d'un nœud avec deux enfants : On peut remplacer le sommet de plus grande valeur du sous arbre gauche (ou la plus petite valeur du sous-arbre droit) et de supprimer le noeud associé dans le sous arbre gauche

L'appel initial se fera avec la valeur de pere à NIL.

```
supprimerNoeudArbreBR(noeud p, arbre abr, noeud pere)
{
    si (abr == p)
        si (pere == NIL)
            abr = NIL;
        sinon
            si ((droite(p) == NIL) || (gauche(p) == NIL))
                si ((gauche(pere) == p) && (droite(p) == NIL))
                    gauche(pere) = gauche(p);
                sinon
                    gauche(pere) = droite(p);
```

```

        si ((droite(pere) == p) && (droite(p) == NIL))
            droite(pere) = gauche(p);
        sinon
            droite(pere) = droite(p);
    sinon
        {
            q = gauche(p);
            faire
            {
                q = droite(q);
            }
            tant que (droite(q) != NIL);
            valeur(p) = valeur(q);
            supprimerNoeudArbreBR(q, gauche(p), p);
        }
    sinon
        si (valeur(p) < valeur(abr))
            supprimerNoeudArbreBR(p, gauche(abr), abr);
        sinon
            supprimerNoeudArbreBR(p, droite(abr), abr);
}

```

Exercice supplémentaire 1

Donner une fonction qui donne le nombre de noeuds d'un arbre binaire.

Correction

```

nombreDeNoeuds(arbre a)
{
    si (a == NIL)
        renvoie 0,
    sinon
        renvoie de 1 + nombreDeNoeuds(gauche(a))
            + nombreDeNoeuds(droite(a));
}

```

Exercice supplémentaire 2

Donner une fonction de parcours en largeur d'un arbre binaire (on pourra utiliser une file pour garder en mémoire les noeuds d'un niveau qui ont été visités).

Correction

```

parcoursLargeur(arbre a)
{
    f = NIL;
    ajouter(racine(a), f);
    tant que (f != fileVide)
    {
        noeud = defiler(f);
        lire(noeud);
        si (gauche(noeud) != NIL)
            ajouter(gauche(noeud), f);
        si (droite(noeud) != NIL)
            ajouter(droite(noeud), f);
    }
}

```

Exercice supplémentaire 3

Ecrire un algorithme qui transforme un tableau de n valeurs trié par ordre croissant en un arbre binaire de recherche.

- a. de hauteur maximale
- b. de hauteur minimale.

Exercice supplémentaire 4

Donner un algorithme qui affiche la liste par ordre croissant des valeurs d'un arbre binaire de recherche.

Exercice supplémentaire 5

Ecrire un algorithme qui transforme un arbre en une liste.

- a. simplement chaînée.
- b. doublement chaînée.

Exercice supplémentaire 6

Trouver une fonction qui détermine la largeur d'un arbre binaire.

Exercice supplémentaire 7

Donner une fonction récursive qui détermine le maximum d'un arbre binaire de recherche abr.

Correction

```
Max_abr(arbre abr)
{
    si (droite(abr) != NIL)
        renvoie de Max_abr(droite(abr));
    sinon
        renvoie de valeur(abr);
}
```