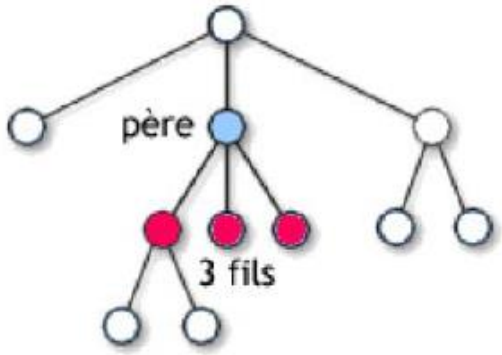
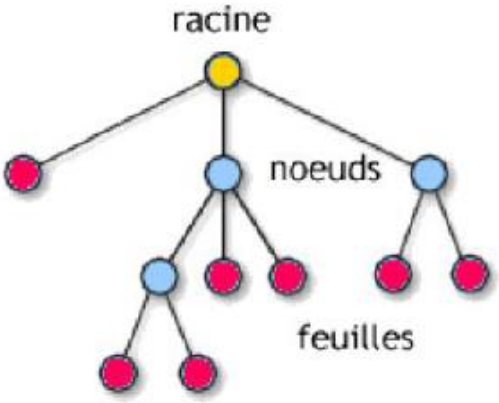
Les arbres

Les structures que nous avons considérées jusqu'à présent sont uniquement des structures linéaires, dans le sens où :
1. chaque élément de la structure possède au plus un prédécesseur direct ;
2. chaque élément de la structure possède au plus un successeur direct.

Nous allons considérer, dans ce chapitre, une généralisation de ces structures linéaires en levant la restriction numéro 2 : chaque élément aura au plus un seul prédécesseur, mais sera autorisé à avoir plusieurs successeurs. On a dès lors affaire à une structure qui se *ramifie*, puisque, lors d'un parcours de cette structure, on a, à chaque élément, plusieurs choix possibles pour « continuer le parcours ». C'est pour cette raison que ce type de structures est appelé un *arbre*.





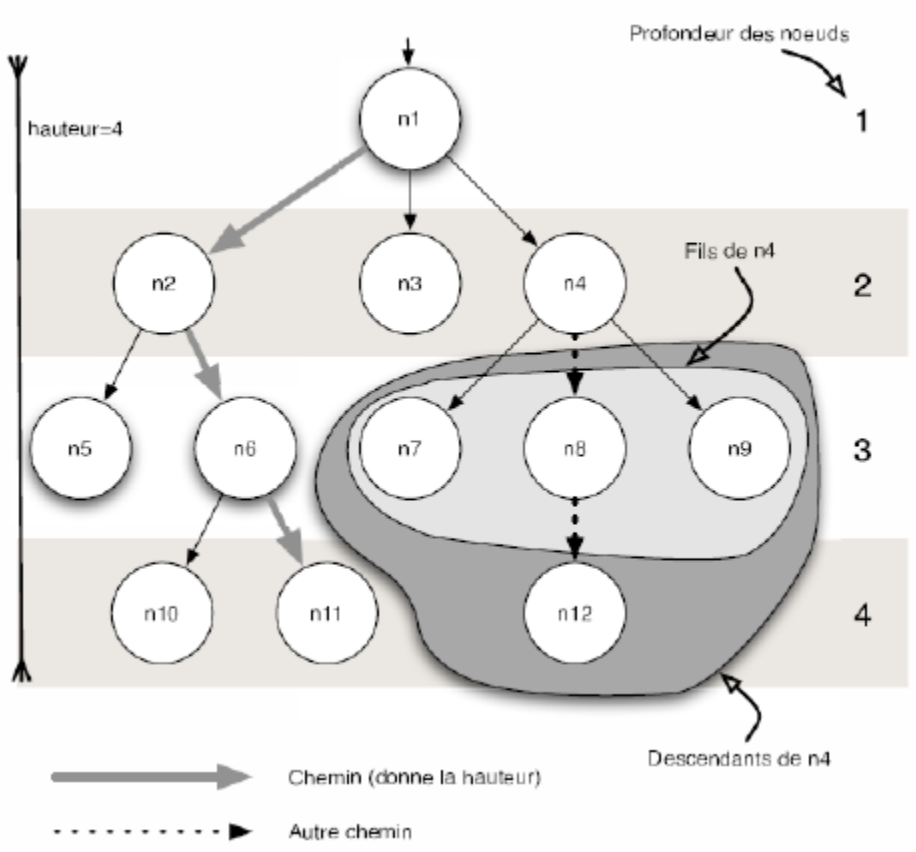
1. Définitions

Il est possible de donner deux types de définitions des arbres : une définition « classique», en termes d'ensembles ; ou bien une définition récursive. Suivant la définition qu'on adopte, on pourra formuler les algorithmes de façon itérative ou récursive.
Définition 1: Un arbre est une structure de données composée d'un ensemble N de nœuds. Chaque nœud n est composé d'une information L(n) et d'un ensemble Succ(n) de nœuds successeurs. Par ailleurs, un et un seul nœud de N est appelé la racine, dénoté r.
Ces éléments respectent les conditions suivantes :

1. r n'a pas de prédécesseur ;
2. Chaque nœud n'a pas plus d'un prédécesseur ;
3. On peut accéder à chaque nœud depuis la racine.

Définition 2
Un arbre est :
1. Soit l'arbre vide ;
2. soit un nœud n appelé racine et un ensemble (potentiellement vide) {A1,A2, . . . ,An} d'arbres non-vides appelés sous arbres.
À chaque nœud est associée une information L(n).

2. Vocabulaire



Un certain vocabulaire conventionnel est associé aux arbres. Il permet en général de simplifier l'exposé des algorithmes sur les arbres.
Fils : Les nœuds de l'ensemble Succ(n) sont appelés les *fils* de n.
Père : Le *père* d'un nœud n est l'unique nœud n' dont n est le fils. Remarquons que la racine n'a pas de père.
Feuille : Une *feuille* est un nœud qui n'a pas de fils.
Nœud interne : Un nœud est un *nœud interne* si et seulement si il n'est pas une feuille.
Accessible : Un nœud n est *accessible* à partir d'un nœud n' s'il existe un chemin allant de n à n' dans l'arbre.

Chemin : Un *chemin* dans un arbre est une séquence $n1n2 \cdots nk$ de nœuds de l'arbre telle que, pour tout $1 \leq i < k$: $ni+1 \in$Succ (ni). On dit que ce chemin va de $n1$ à nk. La *longueur d'un chemin* $n1n2 \cdots nk$ est le nombre k de nœuds qui le composent.
Remarquons que, d'après les définitions d'un arbre, il n'y a qu'un seul chemin allant de la racine à un nœud n donné.
Descendant : Un nœud n est un *descendant* d'un nœud n' si et seulement si n est accessible à partir de n'.
Hauteur : La *hauteur d'un arbre* est la longueur du plus long chemin partant de la racine de l'arbre. La hauteur d'un arbre A est notée hauteur(A).
Profondeur : La *profondeur d'un nœud* n de l'arbre est la longueur du chemin allant de la racine à n.

En pratique, les arbres que l'on utilise, ou qui rendent efficaces les algorithmes proposés, sont des cas particuliers de la définition générale donnée ci-dessus. Dans cette section, nous détaillons trois restrictions supplémentaires qui seront souvent utilisées par la suite.

3. Cas particuliers

3.1. Arbres ordonnés

Dans les définitions données au début de ce chapitre, nous avons toujours considéré qu'un nœud possédait un *ensemble* de fils, ce qui signifie qu'il n'existe pas d'ordre sur ces fils. En pratique, on fixera souvent un ordre, et on parlera alors du *premier fils*, du *second fils*, du *dernier fils*, etc.

3.2. Arbres n-aires

Pour différentes raisons, il sera en général pratique de limiter le nombre de fils qu'un nœud peut posséder. On parle alors d'arbre n-aire :

Définition

Un arbre n-aire est un arbre dont chaque nœud possède au plus n fils.

Le cas le plus utilisé en pratique (et dans la suite de ce cours), est l'arbre binaire, où chaque nœud possède au plus n = 2 fils. Dans la suite, nous supposerons implicitement qu'un arbre binaire est ordonné, et nous appellerons respectivement fils gauche et fils droit le premier et le second fils d'un nœud.

3.3. Terminologie

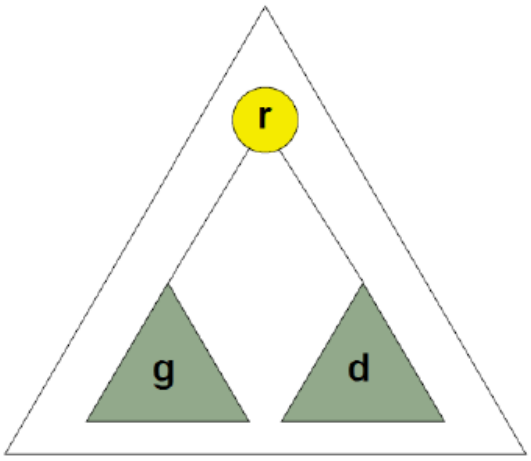
- **L'*arité* d'un nœud**
  - nombre de sous-arbres non vides (0, 1 ou 2)
  - un nœud d'arité nulle est appelé *feuille*
- **Un *chemin* dans un arbre**
  - une séquence de nœuds $(n_0, n_1, \ldots, n_i, \ldots, n_p)$ où n_{i-1} est le père de n_i
  - la longueur du chemin $(n_0, n_1, \ldots, n_i, \ldots, n_p)$ est égale à p
- **Le *niveau* d'un nœud**
  - c'est la longueur de l'**unique** chemin allant de la racine à ce nœud
  - le niveau du nœud racine est égal à 0
- **La *hauteur* d'un arbre**
  - c'est le **maximum** des niveaux de tous les nœuds
  - la hauteur d'un arbre **réduit au nœud racine** vaut 0
  - la hauteur d'un arbre **vide** vaut par **convention** -1

4. Les arbres binaires

4.1. Définition d'un arbre binaire

Un arbre binaire est un arbre ordonné tel que chaque nœud a au plus deux fils et quand il n'y en a qu'un, on distingue le fils droit du fils gauche.

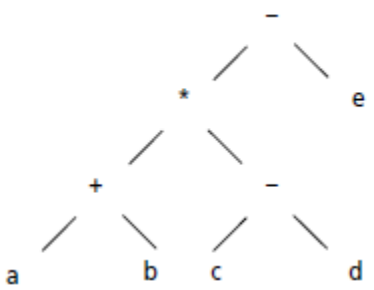
- **Un arbre binaire** a est :
  - soit **vide** : $a = ()$
  - soit composé de :
    - un nœud **racine** r
    - deux **arbres binaires disjoints** :
      - un **sous-arbre gauche** g
      - un **sous-arbre droit** d
    - $a = (r, g, d)$



4.2. Déclaration en C

```
typedef struct Arbre
{
        int Noeud;
        struct Arbre * SAG;
        struct Arbre * SAD;
} Arbre;
```

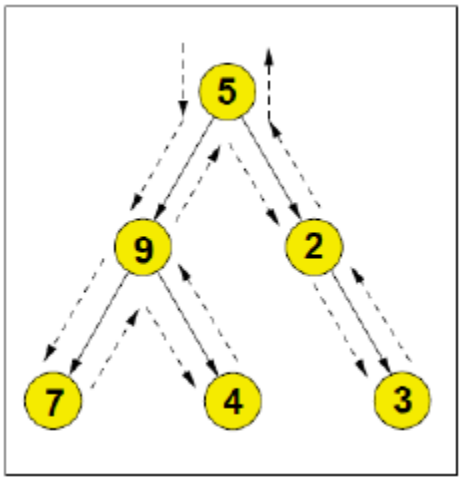
4.3. Exemple d'arbre binaire : *Expression arithmétique*



Arbre binaire de ((a+b)*(c-d))-e.

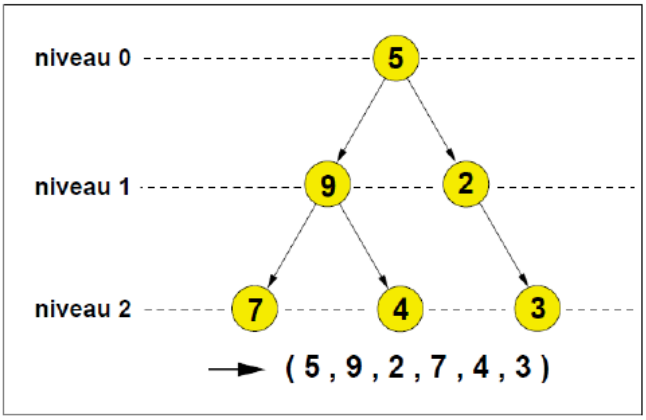
5. Parcours d'un arbre binaire

Un algorithme de parcours d'arbre est un procédé permettant d'accéder à chaque nœud de l'arbre. Un certain traitement est effectué pour chaque nœud (test, écriture, comptage, etc.), mais le parcours est indépendant de cette action et commun à des algorithmes qui peuvent effectuer des traitements très divers.
On distingue deux catégories de parcours d'arbres : les parcours en **profondeur** et les parcours en **largeur**. Dans le parcours en profondeur, on explore **branche par branche** alors que dans le parcours en largeur on explore **niveau par niveau**.



(5, 9, 7, 4, 2, 3)

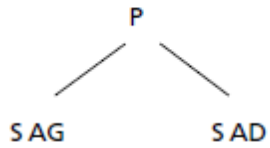
Exemple de parcours en profondeur



Exemple de parcours en largeur

5.1. Les différentes méthodes de parcours en profondeur d'un arbre

Il y a 6 types de parcours possibles (P : père, SAG : sous-arbre gauche, SAD : sous-arbre droit). Nous ne considérons dans la suite de ce chapitre que les parcours gauche-droite. Les parcours droite-gauche s'en déduisent facilement par symétrie.



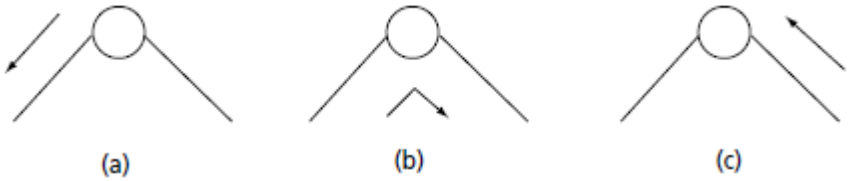
Ces parcours sont appelés parcours en **profondeur** car on explore une branche de l'arbre le plus profond possible avant de revenir en arrière pour essayer un autre chemin.

	gauche - droite	droite - gauche
préfixé	P . SAG . SAD	P . SAD . SAG
infixé	SAG . P . SAD	SAD . P . SAG
postfixé	SAG . SAD . P	SAD . SAG . P

Les 6 types de parcours d'un arbre binaire.

Dans un parcours d'arbre gauche-droite, un nœud est visité trois fois :
• lors de la première rencontre du nœud, avant de parcourir le sous-arbre gauche.
• après parcours du sous-arbre gauche, avant de parcourir le sous-arbre droit.
• après examens des sous-arbres gauche et droit.

L'action à effectuer sur le nœud peut se faire lors de la visite (a), (b) ou (c).



Les 3 visites d'un nœud lors d'un parcours d'arbre binaire.

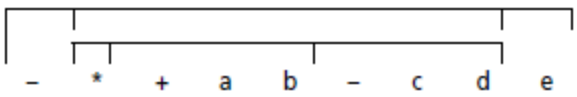
5.1.1. Parcours préfixé

Le premier type de parcours est appelé parcours **préfixé**. Il faut traiter le nœud lors de la **première** visite, puis explorer le sous-arbre gauche (en appliquant la même méthode) avant d'explorer le sous-arbre droit.

La procédure se schématise comme suit :

- traitement de la racine ;
- traitement du sous-arbre gauche ;
- traitement du sous-arbre droit.

Pour chaque noeud, on trouve le nom du noeud concerné, les éléments du SAG, puis les éléments du SAD.



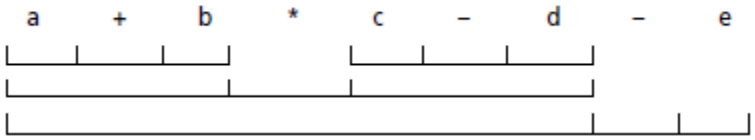
5.1.2. Parcours infixé

Dans un parcours **infixé**, le nœud est traité lors de la **deuxième** visite, après avoir traité le sous-arbre gauche, mais avant de traiter le sous-arbre droit.

Un noeud se trouve entre son SAG et son SAD.

La procédure se schématise comme suit :

- traitement du sous-arbre gauche ;
- traitement de la racine ;
- traitement du sous-arbre droit.

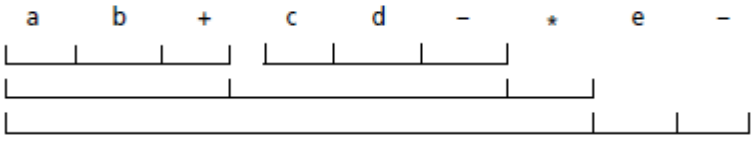


5.1.3. Parcours postfixé

En parcours **postfixé**, le nœud est traité lors de la **troisième** visite, après avoir traité le SAG et le SAD.

La procédure à suivre est donnée ci-dessous :

- traitement du sous-arbre gauche ;
- traitement du sous-arbre droit ;
- traitement de la racine.



Parcours en largeur dans un arbre

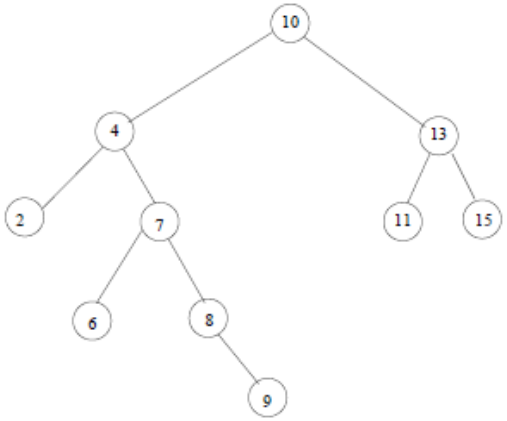
Une autre méthode de parcours des arbres consiste à les visiter étage par étage, comme si on faisait une coupe par niveau. Ainsi, sur l'arbre binaire de l'expression arithmétique, le parcours en largeur est le suivant : - * e + - a b c d. Ce parcours nécessite l'utilisation d'une file d'attente contenant initialement la racine. On extrait l'élément en tête de la file, et on le remplace par ses successeurs à gauche et à droite jusqu'à ce que la file soit vide. Dans les parcours d'arbres, on effectue plutôt des parcours en profondeur, bénéficiant ainsi des mécanismes automatiques de retour en arrière de la récursivité. Le parcours en largeur est effectué lorsque les résultats l'imposent comme pour le dessin d'un arbre binaire.

6. Arbres binaires de recherche

Un arbre binaire de recherche est un arbre binaire qui possède la priorité fondamentale suivante:

- tous les nœuds du sous-arbre de gauche d'un nœud de l'arbre ont une valeur inférieure ou égale à la sienne.
- tous les nœuds du sous-arbre de droite d'un nœud de l'arbre ont une valeur supérieure ou égale à la sienne.

Exemple :



6.1. Recherche dans l'arbre

Un arbre binaire de recherche est fait pour faciliter la recherche d'informations.

La recherche d'un nœud particulier de l'arbre peut être définie simplement de manière récursive :

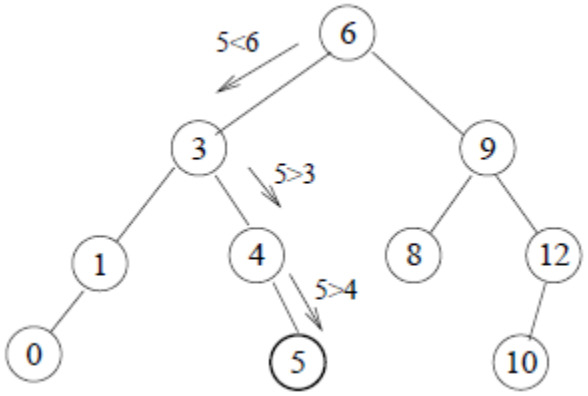
Soit un sous-arbre de racine n_i :

- si la valeur recherchée est celle de la racine n_i, alors la recherche est terminée. On a trouvé le nœud recherché.
- sinon, si n_i est une feuille (pas de fils) alors la recherche est infructueuse et l'algorithme se termine.
- si la valeur recherchée est plus grande que celle de la racine alors on explore le sous-arbre de droite c'est à dire que l'on remplace n_i par son nœud fils de droite et que l'on relance la procédure de recherche à partir de cette nouvelle racine.
- de la même manière, si la valeur recherchée est plus petite que la valeur de n_i, on remplace ni par son nœud fils de gauche avant de relancer la procédure…

6.2. Ajout d'un élément

Pour conserver les propriétés d'un arbre binaire de recherche, l'ajout d'un nouvel élément ne peut pas se faire n'importe comment.
L'algorithme récursif d'ajout d'un élément peut s'exprimer ainsi:

- soit x la valeur de l'élément à insérer.
- soit v la valeur du nœud racine n_i d'un sous-arbre.
  - si n_i n'existe pas, le créer avec la valeur x. fin.
  - sinon
    - si x est plus grand que v,
      - remplacer n_i par son fils droit.
      - recommencer l'algorithme à partir de la nouvelle racine.
    - sinon
      - remplacer ni par son fils gauche.
      - recommencer l'algorithme à partir de la nouvelle racine.



6.3. Implémentation en C

En langage C, un nœud d'un arbre binaire peut être représenté par une structure contenant un champ donnée et deux pointeurs vers les nœuds fils :

```
struct s_arbre
{
int valeur;
struct s_arbre *gauche;
struct s_arbre *droit;
};
typedef struct s_arbre t_arbre;
```