1.1

Algorithmique et Structures de Données

Jean-Charles Régin

Licence Informatique 2ème année

1.2

Arbres

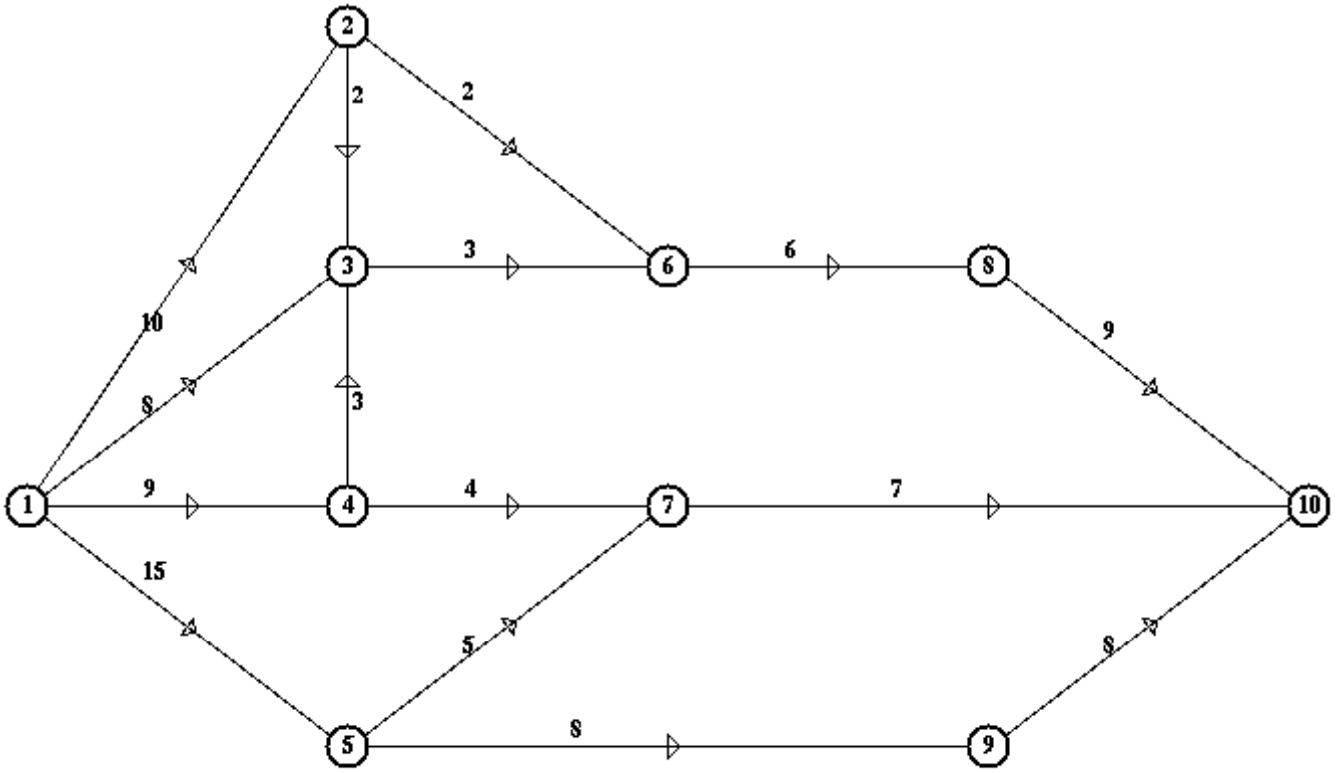
Jean-Charles Régin

Licence Informatique 2ème année

Graphe : définitions

- Un Graphe Orienté G=(X,U) est déterminé par la donnée :
  - d'un ensemble de **sommets** ou **nœuds** X
  - d'un ensemble ordonné U de couples de sommets appelés **arcs**.
- Si u=(i,j) est un arc de G, alors i est l'extrémité initiale de de u et j l'extrémité terminale de u.
- Les arcs ont un sens. L'arc u=(i,j) va de i vers j.
- Ils peuvent être munis d'un coût, d'une capacité etc.

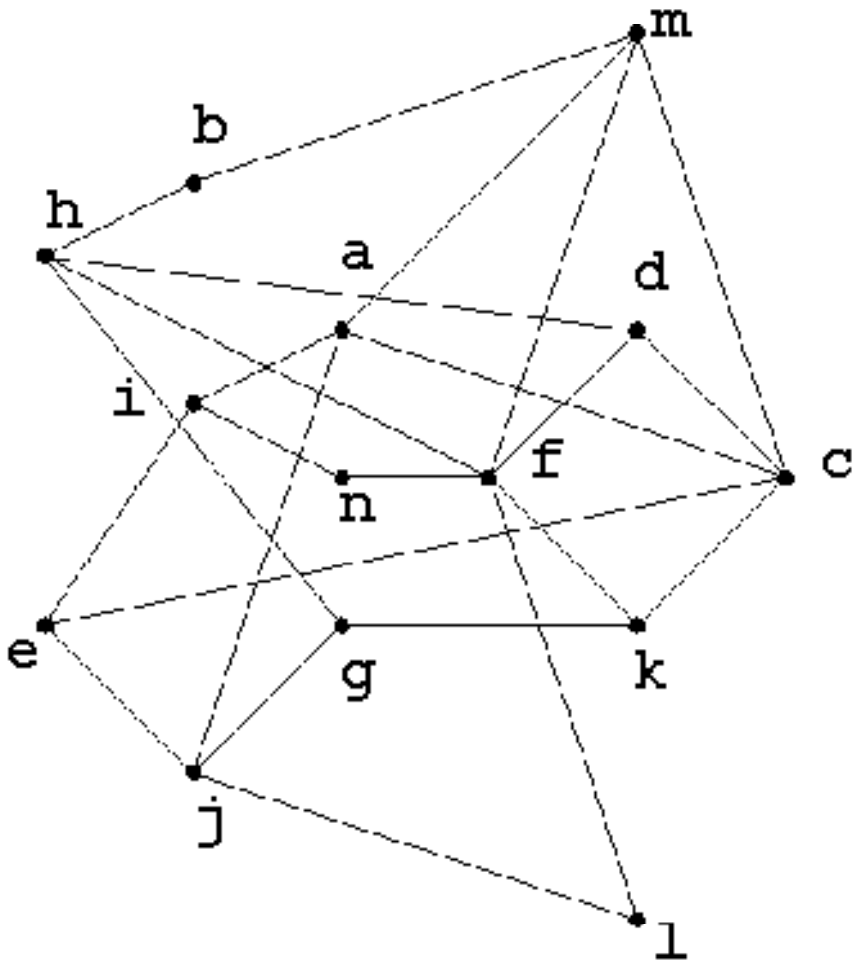
Graphe



Graphe non orienté

- Un graphe non orienté G=(X,E) est déterminé par la donnée :
  - d'un ensemble de sommets ou nœuds X
  - D'un ensemble E de paires de sommets appelées **arêtes**
- Les arêtes ne sont pas orientées

Graphe non orienté



Graphe : définitions

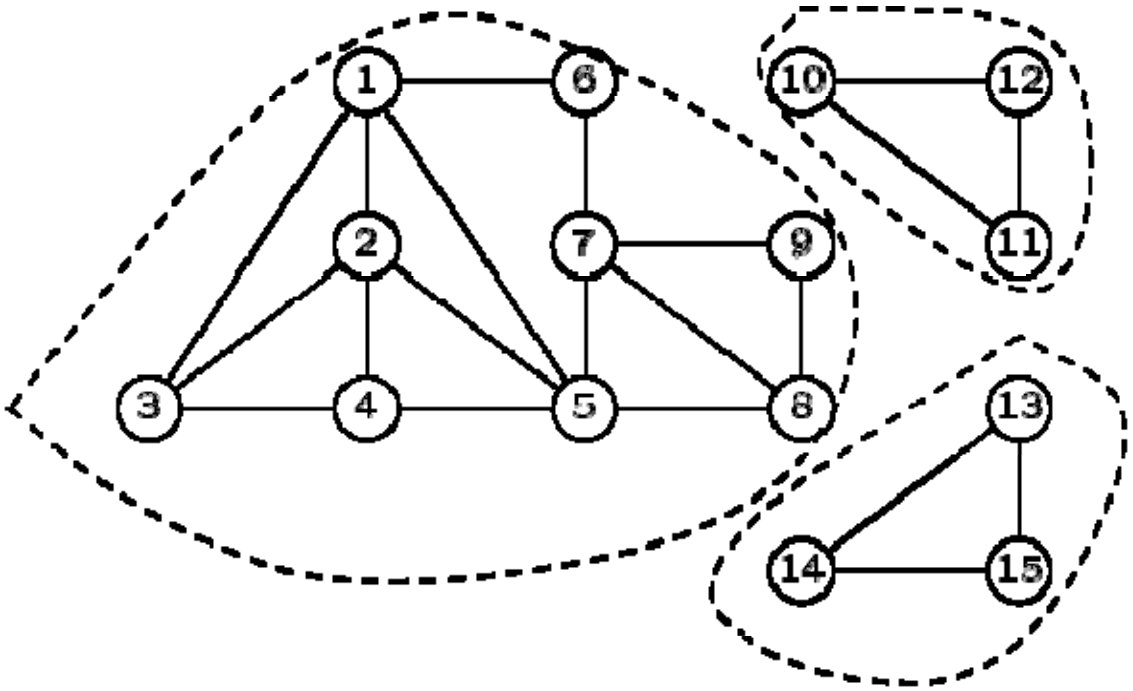
- Chemin de longueur q : séquence de q arcs $\{u_1,u_2,\ldots,u_q\}$ telle que
  - $u_1=(i_0,i_1)$
  - $u_2 = (i_1,i_2)$
  - $u_q=(i_{q-1},i_q)$
- Chemin : tous les arcs orientés dans le même sens
- Circuit : chemin dont les extrémités coïncident

Graphe : définitions

- Chaîne de longueur q : séquence de q arêtes $\{u_1,u_2,\ldots,u_q\}$ telle que
  - $u_1=\{i_0,i_1\}$
  - $u_2 = \{i_1,i_2\}$
  - $u_q=\{i_{q-1},i_q\}$
- Cycle : chaîne dont les extrémités coïncident

Graphe : définitions

- Connexité : un graphe non orienté est connexe s'il existe une chaîne entre tout paire de sommets

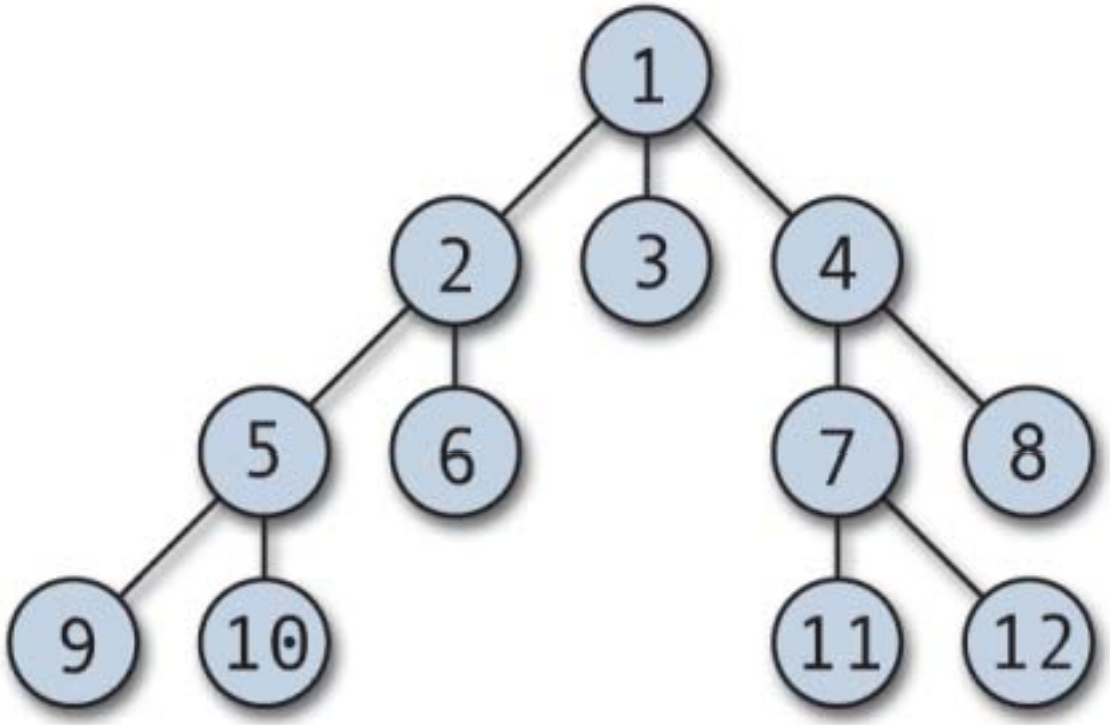


Arbre

- Un arbre (tree en anglais) est un graphe non orienté connexe et sans cycle

- Un graphe non orienté G ayant n sommets est un arbre si et seulement si il vérifie l'une des deux propriétés
  - G est connexe et possède n-1 arêtes
  - G n'a pas de cycle et a n-1 arêtes.

Arbre

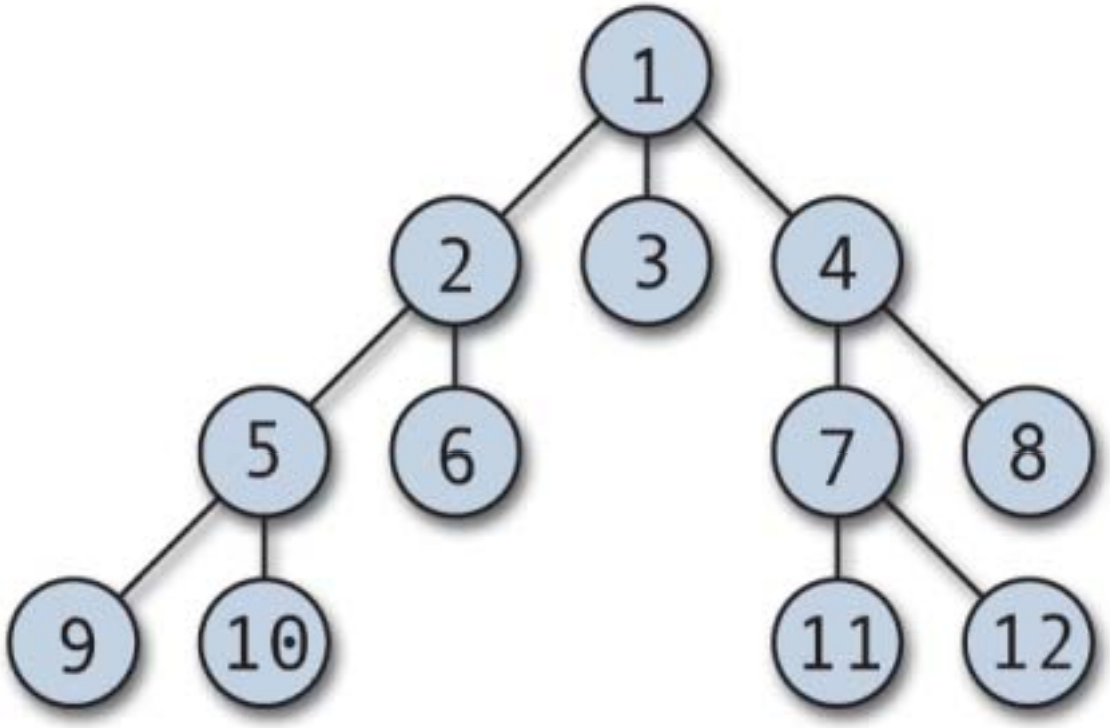


Arbre

- La **racine** r de l'arbre est l'unique nœud ne possédant pas de parent

- Tout sommet x qui n'est pas la racine a
  - un unique parent, noté parent(x) (appelé père parfois)
  - 0 ou plusieurs fils. fils(x) désigne l'ensemble des fils de x
- Si x et y sont des sommets tels que x soit sur le chemin de r à y alors
  - x est un ancêtre de y
  - y est un descendant de x
- Un sommet qui n'a pas de fils est une feuille

Arbre



1 est la racine

9,10,6,3,11,12,8 sont les feuilles

11 est un descendant de 4, mais pas de 2

2 est un ancêtre de 10

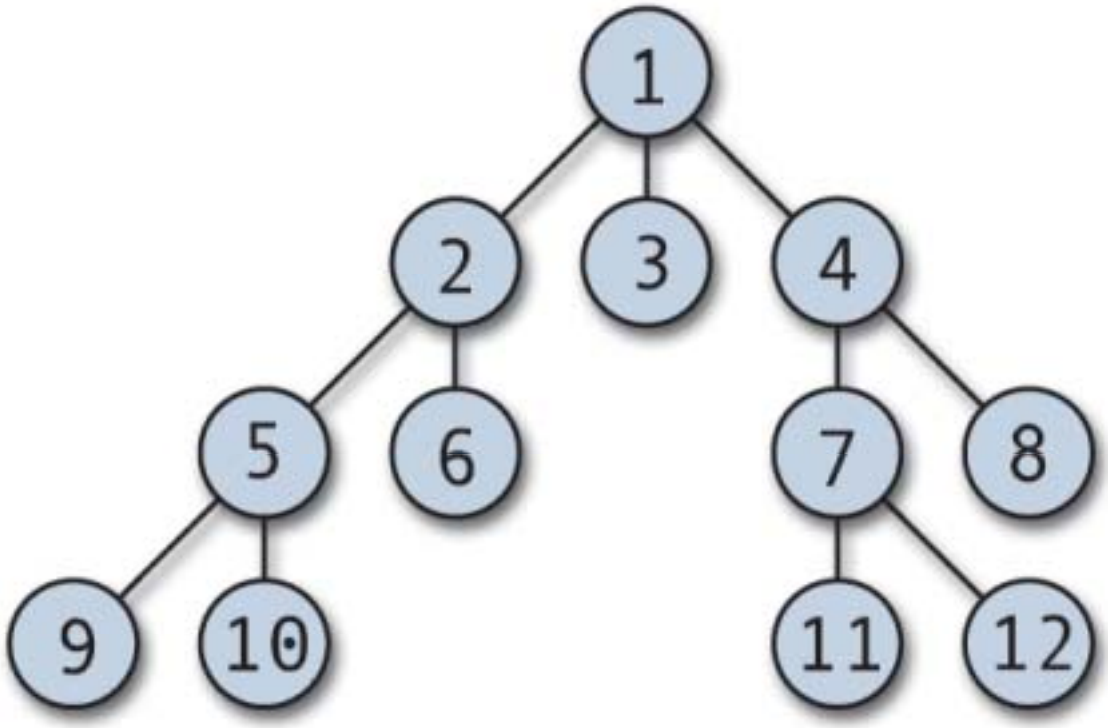
Arbre

- Quand il n'y a pas d'ambiguïté, on regarde les arêtes d'un arbre comme étant orienté de la racine vers les feuilles

- La **profondeur** d'un sommet (depth) est définie récursivement par
  - prof(v) = 0 si v est la racine
  - prof(v) = prof(parent(v)) + 1

- La **hauteur** d'un sommet (height) est la plus grande profondeur d'une feuille du sous-arbre dont il est la racine

Arbre



1 est la racine

2,3,4 sont à la profondeur 1

5,6,7,8 à la profondeur 2

La hauteur de 2 est 2, celle de 9 est 0, celle de 3 est 0, celle de 1 est 4

Arbre : utilisation

- Gérer des bases de données
- Indexation de fichiers.
- Tri par tas

- Ils permettent des recherches rapides et efficaces.

Arbre : parcours

- Parcours (tree traversal) : on traverse l'ensemble des sommets de l'arbre

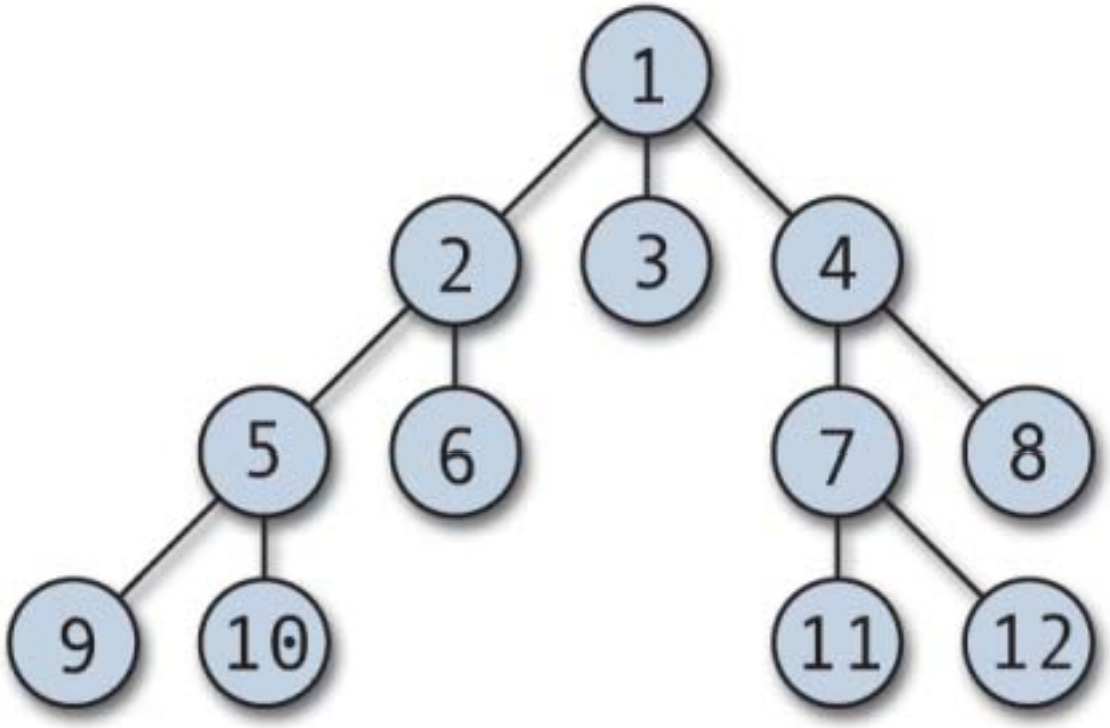
- Parcours en largeur d'abord
- Parcours en profondeur d'abord
  - Préfixé
  - Infixé
  - Postfixé

Arbre : parcours en largeur d'abord

- Largeur d'abord (bfs = breadth-first search)
- On visite la racine, puis on répète le processus suivant jusqu'à avoir visité tous les sommets : visiter un fils non visité du sommet le moins récemment visité qui a au moins un fils non visité

- On visite tous les sommets à la profondeur 1, puis tous ceux à la profondeur 2, puis tous ceux à la profondeur 3 etc…

Arbre



Largeur d'abord: ordre de visite

1, 2 ,3 ,4 ,5, 6 , 7, 8, 9 , 10 ,11 ,12

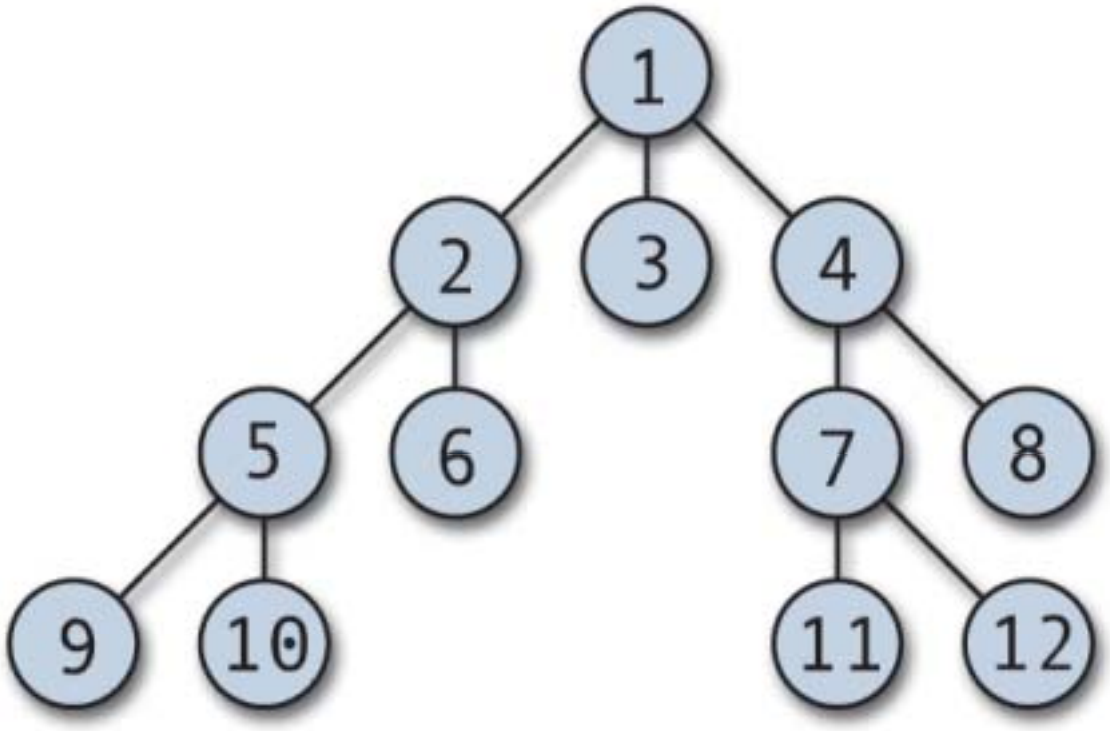
Arbre : largeur d'abord


```
Bfs(T) : array
r ← racine(T)
créer un file F et ajouter r dans F
i ← 0
tant que (F n'est pas vide) {
        x ← premier(F); supprimer x de F
        array[i] ← x
        i++
        pour chaque fils y de x {
                ajouter y dans F
        }
}
```

Arbre : largeur d'abord avec passes


```
Bfs(T) : array
r ← racine(T)
créer deux files F1 et F2 et ajouter r dans F1
i ← 0
faire
        tant que (F1 n'est pas vide)
                x ← premier(F1); supprimer x de F1
                array[i] ← x
                i++
                pour chaque fils y de x
                        ajouter y dans F2
                fin pour
        fin tant que
        F1 ← F2          // fin d'une passe début de la nouvelle : la
        F2 devient vide  // profondeur change
tant que F1 n'est pas vide
```

Arbre



Largeur d'abord: ordre de visite

Passe 1 : 1

Passe 2 : 2,3,4

Passe 3 : 5,6,7,8 et passe 4 : 9,10,11,12

Arbre parcours en profondeur d'abord

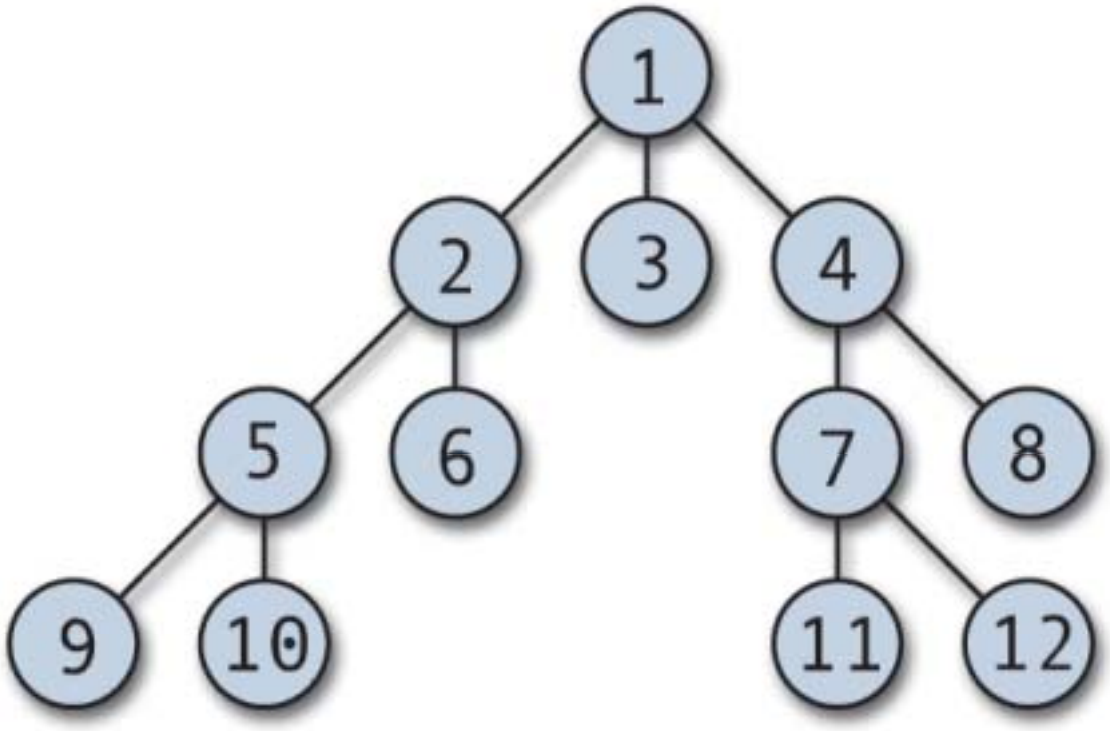
- Profondeur d'abord (dfs = depth-first search)

- Défini de façon récursive
- visit(sommet x)
  previsit(x)
  pour chaque fils y de x
      visit(y)
  postvisit(x)
- Premier appel : visit(racine(T))

Abre : profondeur d'abord

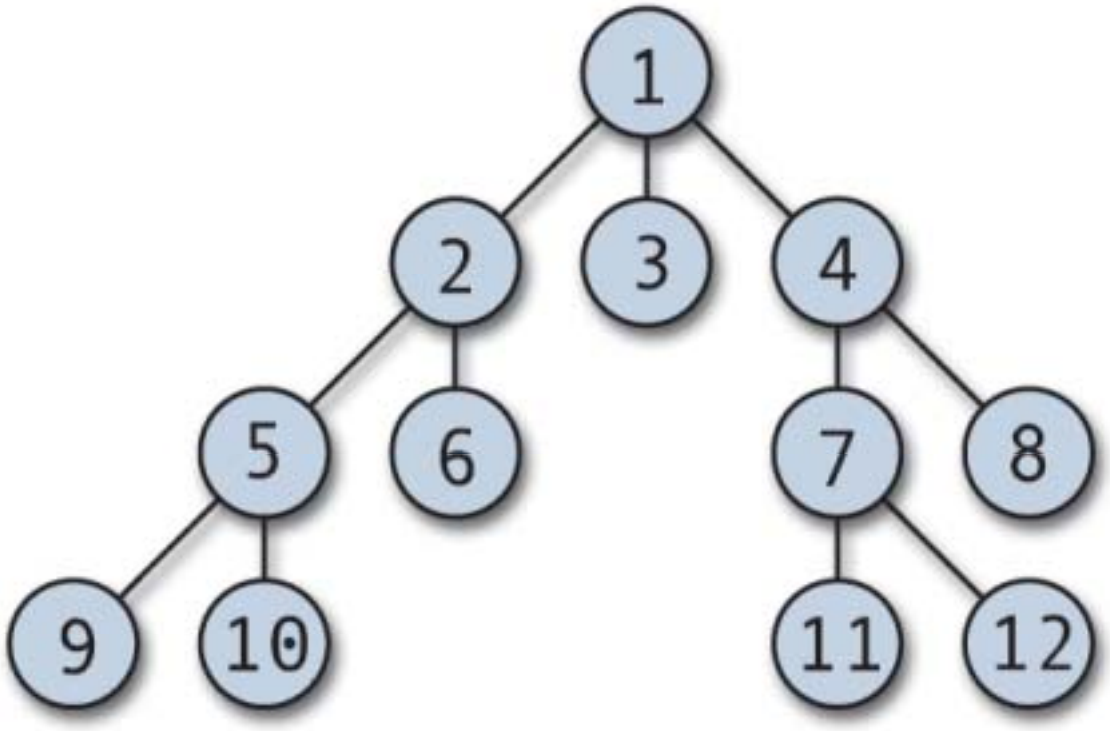
- Ordre préfixé ou postfixé dépend des fonctions previsit et postvisit
- Si previsit(x) : met x dans array et incrémente i alors array contient l'ordre préfixé
- Si c'est postvisit qui le fait alors array contiendra l'ordre postfixé

Arbre



Profondeur d'abord: ordre de visite préfixé
On marque quand on atteint le noeud
1 ,2 ,5, 9, 10, 6, 3, 4, 7, 11, 12, 8

Arbre

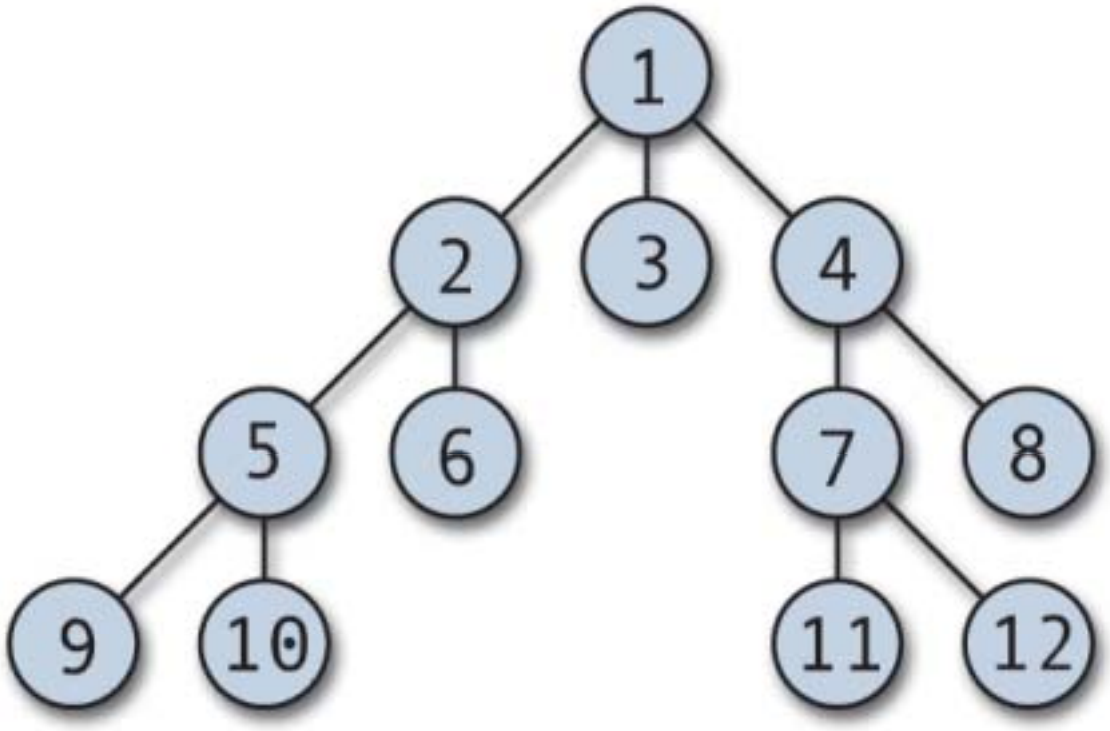


Profondeur d'abord: ordre de visite postfixé
On marque quand on quitte le noeud
9, 10, 5, 6, 2, 3, 11, 12, 7, 8, 4, 1

Arbre : infixé ?

- Valable uniquement pour les arbre binaires (2 fils au plus)
- visit(sommet x)
  previsit(x)
  si fils gauche y existe alors visit(y)
  invisit(x)
  si fils droit y existe alors visit(y)
  postvisit(x)
- Premier appel : visit(racine(T))
- invisit(x) place x dans array

Arbre



Profondeur d'abord: ordre de visite infixé (on ignore 3)
On marque entre les deux fils
9, 5, 10, 2, 6, 1, 11, 7,12, 4, 8

Arbre : profondeur d'abord itérative

- On peut ne pas utiliser un algorithme récursif pour représenter un parcours en profondeur d'abord
- Il faut utiliser une pile
  - Visiter un fils revient à empiler le père
  - Et visiter le fils (qui va être empilé par ces fils et ainsi de suite)
  - Remonter (terminer l'appel récursif) revient à dépiler
  - Quand il n'y a plus de fils, on reprend le sommet de la pile, on dépile et on passe au fils suivant

Arbre : implémentation

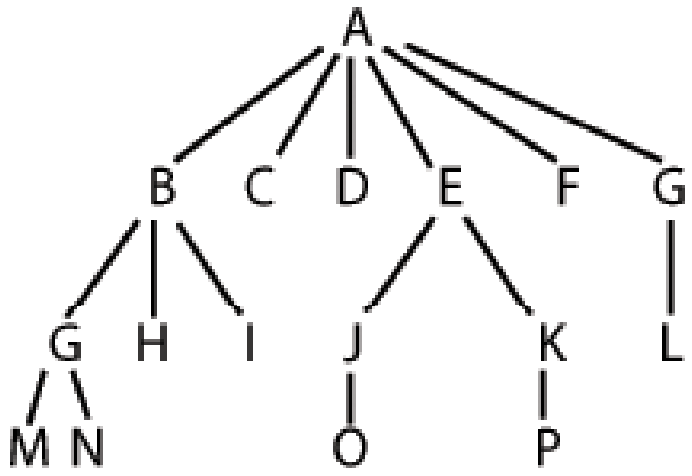
- Représentation des fils
  - Par une liste :
    - Le parent possède un premier fils
    - Chaque sommet possède un pointeur vers son frère suivant (liste chaînée des fils)
  - Par un tableau si le nombre de fils est connu à l'avance (arbre k-aire)
  - Dans le cas binaire, le parent possède le fils gauche et le fils droit

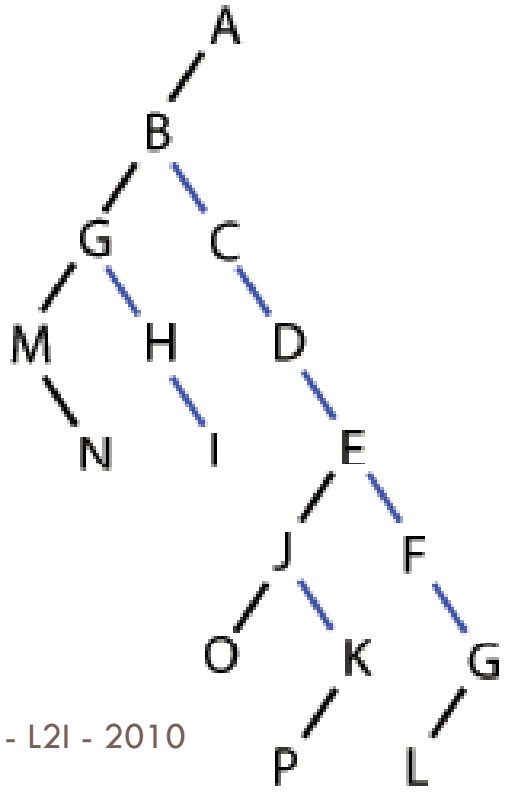
Arbre : implémentation

- Arbre binaire complet : on peut utiliser un tableau pour représenter tout l'arbre
- Voir dessin

Arbre non binaire

- Tout arbre non binaire peut être représenté par un arbre binaire





Classe Arbre ?

- Attention un arbre est un graphe qui vérifie une propriété
- Si vous définissez une classe Arbre, les instances de cette classe doivent toujours vérifier cette propriété. Cela impose des contraintes comme :
  - Suppression d'un sommet entraine la suppression du sous-arbre dont le sommet est racine
  - On ne peut pas ajouter d'arête
  - On fusionne des arbres