Structures de données non linéaires

Arbres

Objectifs

- Étudier des structures non linéaires
  - Arbres
  - Arbres binaires
  - Arbres binaires de recherche

Contenu

Introduction

- Dans les tableaux
  - Un accès direct par indice (rapide)
  - L'insertion et la suppression nécessitent des décalages
- Dans les Listes Linéaires Chaînées
  - Un accès séquentiel lent
  - L'insertion et la suppression se font uniquement par modification de chaînage
- Les arbres représentent un compromis entre les deux
  - Un accès relativement rapide à un élément à partir de sa clé
  - L'insertion et la suppression non coûteuses
- L'usage des arbres est multiple, car ils captent l'idée de hiérarchie
  - Exemple : découpage d'un livre en parties : chapitres, sections, paragraphes, ...

Notion d'Arbre

- Les arbres sont les structures de données les plus importantes en informatique
- Ce sont des structures non linéaires qui permettent d'obtenir des algorithmes plus performants que lorsqu'on utilise des structures de données linéaires telles que les listes et les tableaux
- Ils permettent une organisation naturelle des données

Notion d'Arbre

- Exemples
  - Organisation des fichiers dans les systèmes d'exploitation ;
  - Organisation des informations dans un système de bases de données ;
  - Représentation de la structure syntaxique des programmes sources dans les compilateurs ;
  - Représentation d'une table de matières ;
  - Représentation d'un arbre généalogique ;
  - ...

Terminologie (1)

- Un arbre est un ensemble d'éléments appelés nœuds (ou sommets), liés par une relation (dite de "parenté") induisant une structure hiérarchique parmi ces nœuds.

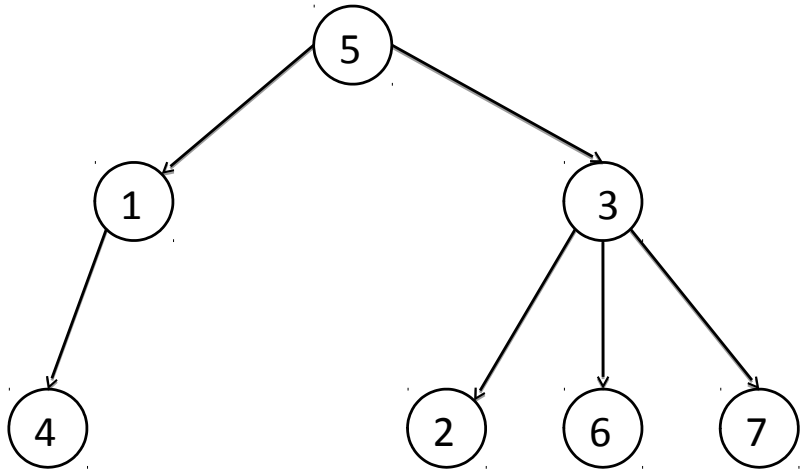
- Un nœud, comme tout élément d'une liste, peut être de n'importe quel type.

Terminologie (2)

- D'une manière plus formelle, une structure d'arbre de type de base T est :
  - soit la structure vide ;
  - soit un nœud de type T, appelé racine , associé à un nombre fini de structures d'arbre disjointes du type de base T appelées sous arbres
- C'est une définition récursive ; la récursivité est une propriété des arbres et des algorithmes qui les manipulent
- Une liste est un cas particulier des arbres (arbre dégénéré), où tout nœud a au plus un sous arbre

Terminologie (3) : illustration & exemple

- Pour illustrer une structure d'arbre, on modélise le plus souvent un nœud par une information inscrite dans un cercle et les liens par des traits.
- Par convention , on dessine les arbres avec la racine en haut et les branches dirigées vers le bas.



Terminologie (4)

- La terminologie utilisée dans les structures d'arbres est empruntée :
  1. aux arbres généalogiques :
     - Père ;
     - Fils ;
     - Frère ;
     - Descendant ;
  2. et à la botanique :
     - Feuille ;
     - Branche ;

Terminologie (5)

- Fils (ou enfants) :
  - Chaque nœud d'un arbre pointe vers un ensemble éventuellement vide d'autres nœuds ; ce sont ses fils (ses enfants).
  - Sur l'exemple précédent, le nœud 5 a deux fils : 1 et 3, le nœud 1 a un fils : 4, et le nœud 3 a trois fils : 2, 6 et 7.
- Père :
  - Tous les nœuds d'un arbre, sauf un, ont un père et un seul. Un nœud p est père du nœud n si et seulement si n est fils de p.
  - Par exemple, le père de 2 est 3, celui de 3 et 5.
- Frères :
  - Deux nœuds ayant le même père.
  - Les nœuds 2, 6 et 7 sont des frères.
- Racine :
  - Le seul nœud sans père.
  - 5 est la racine de l'arbre précédent.

Terminologie (6)

- Feuilles (ou nœuds terminaux, ou nœuds externes) :
  - Ce sont des nœuds sans fils.
  - Par exemple, 4, 2, 6 et 7.
- Nœud interne :
  - Un nœud qui n'est pas terminal.
  - Par exemple, 1, 3 et 5.
- Degré d'un nœud :
  - Le nombre de fils de ce nœud.
  - Sur l'exemple, 5 est de degré deux, 1 est de degré un, 3 est de degré trois et les feuilles (4, 2, 6, 7) sont de degré nul.
- Degré d'un arbre (ou arité) :
  - Plus grand degré des nœuds de l'arbre. Un arbre de degré n est dit n-aire
  - Sur l'exemple, l'arbre est un arbre 3-aire.
- Taille d'un arbre :
  - Le nombre total des nœuds de l'arbre.
  - Sur l'exemple, l'arbre est de taille 7.

Terminologie (7)

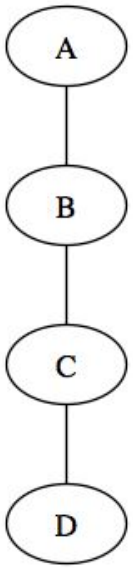
- Chemin :
  - Une suite de nœuds d'un arbre $(n_1, n_2, ..., n_k)$ tel que n_i=père$(n_i + 1)$ pour $1 \leq i \leq k - 1$ est appelée chemin entre le nœud n_1 et le nœud n_k.
  - La longueur d'un chemin est égale au nombre de nœuds qu'il contient moins 1.
  - Sur l'exemple, le chemin qui mène du nœud 5 au nœud 6 est de longueur 2.
- Branche :
  - Un chemin qui commence à la racine et se termine à une feuille.
  - Par exemple, les chemins (5, 1, 4), (5, 3, 2), (5, 3, 6) et (5, 3, 7).
- Descendant :
  - Un nœud A est un descendant d'un nœud B s'il existe un chemin de B vers A.
  - Sur l'exemple, 5 admet les 7 nœuds de l'arbre comme descendants.

Terminologie (8)

- Ancêtre :
  - Un nœud A est un ancêtre d'un nœud B s'il existe un chemin de A vers B.
  - Par exemple, les ancêtres de 2 sont 2, 3 et 5
- Sous arbre :
  - Un sous arbre d'un arbre A est constitué de tous les descendants d'un nœud quelconque de A.
  - Les ensembles de nœuds 3, 2, 6, 7 et 2 forment deux sous arbres de l'exemple précédent.
- Hauteur (ou profondeur, ou niveau) d'un nœud :
  - Longueur du chemin qui relie la racine à ce nœud.
  - La racine est elle même de hauteur 0, ses fils sont de hauteur 1, et les autres nœuds de hauteur supérieure à 1.
- Hauteur d'un arbre :
  - Plus grande profondeur des nœuds de l'arbre supposé non vide, c'est-à-dire $h(A) = \text{Max}\{h(x)\,;\, x \text{ nœud de } A\}$
  - L'arbre de l'exemple est de profondeur 2.
  - Par convention , un arbre vide a une hauteur de -1.

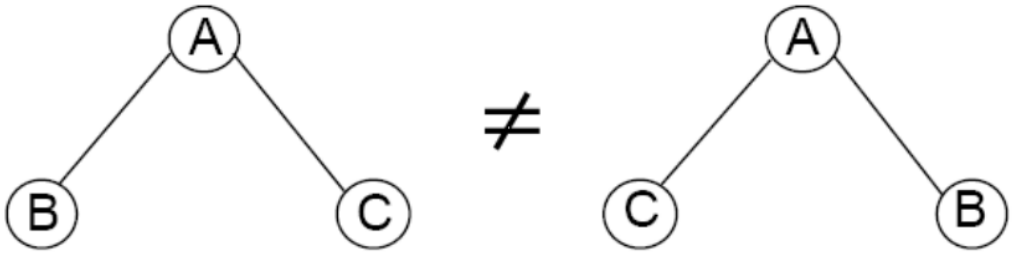
Terminologie (9)

- Arbre dégénéré ou filiforme :
  - Un arbre dont chaque nœud a au plus un fils



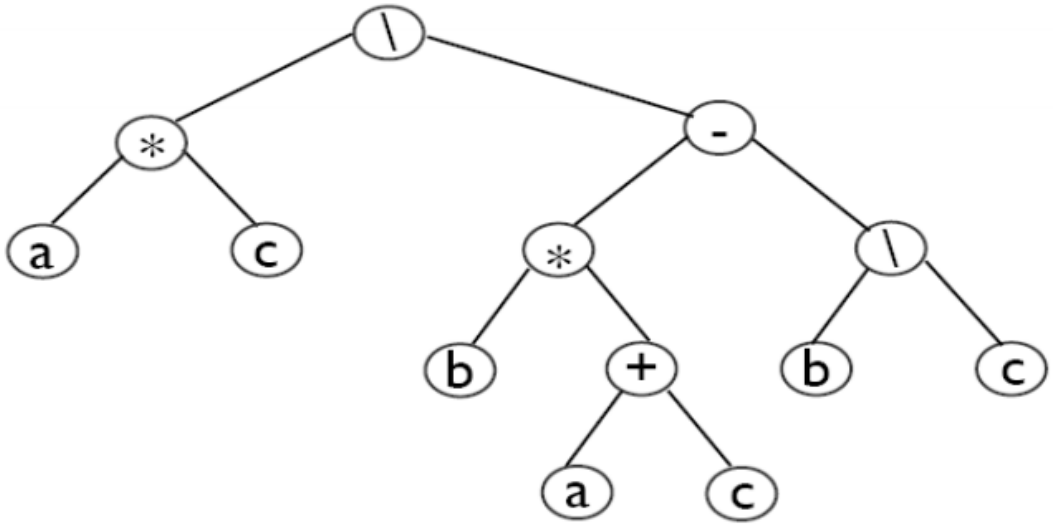
Terminologie (10)

- Arbre ordonné :
  - Un arbre où la position respective des sous arbres reflète une relation d'ordre. En d'autres termes, si un nœud a k fils, il existe un 1^{er} fils, un 2^{eme} fils, ..., et un k^{eme} fils.
  - Les deux arbres de la figure qui suit sont différents si on les regarde comme des arbres ordonnés, mais identiques si on les regarde comme de simples arbres.



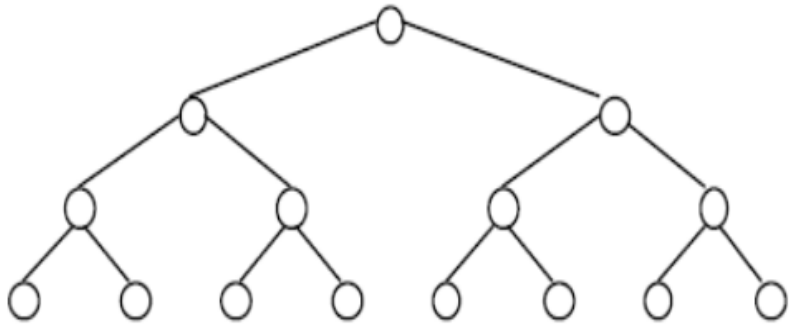
Terminologie (11)

- Arbre binaire :
  - Un arbre où chaque nœud a au plus deux fils.
  - Quand un nœud de cet arbre a un seul fils, on précise s'il s'agit du fils gauche ou du fils droit .
  - La figure qui suit montre un exemple d'arbre binaire dans lequel les nœuds contiennent des caractères.



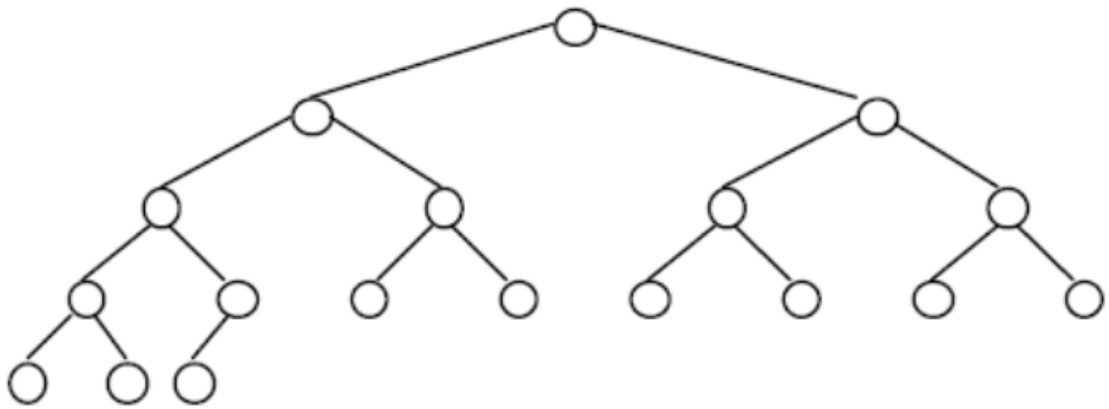
Terminologie (12)

- Arbre binaire complet :
  - Arbre binaire dont chaque niveau est rempli.



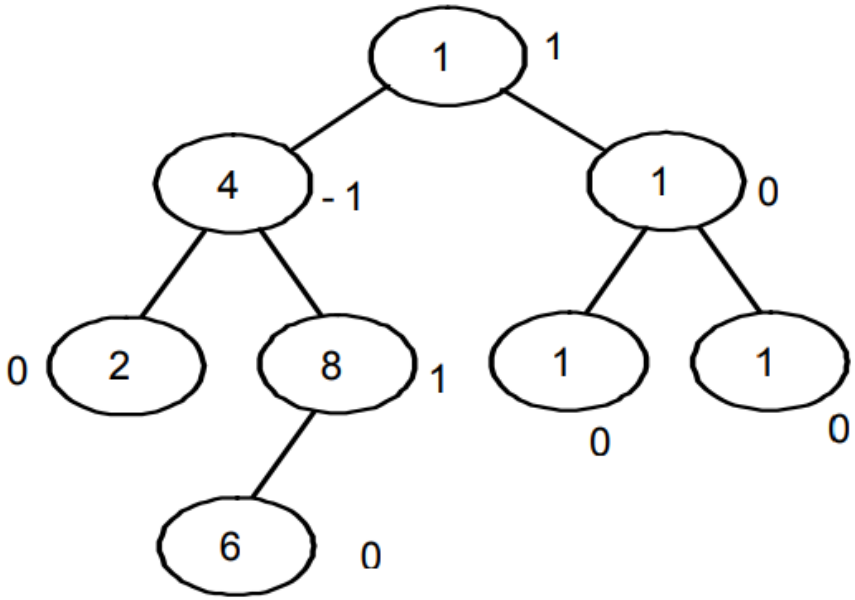
Terminologie (13)

- Arbre binaire parfait (ou presque complet) :
  - Arbre binaire dont chaque niveau est rempli sauf éventuellement le dernier
  - Dans ce cas les nœuds terminaux (feuilles) sont groupés le plus à gauche possible.



Terminologie (14)

- Facteur d'équilibre d'un nœud d'un arbre binaire :
  - Hauteur du sous arbre partant du fils gauche du nœud moins la hauteur du sous arbre partant de son fils droit.
- Arbre binaire équilibré (au sens des hauteurs) :
  - Un arbre binaire tel que pour chaque nœud, la valeur absolue du facteur d'équilibre est inférieure ou égal à un.
  - Sur l'exemple qui suit, on place à côté de chaque nœud son facteur d'équilibre.



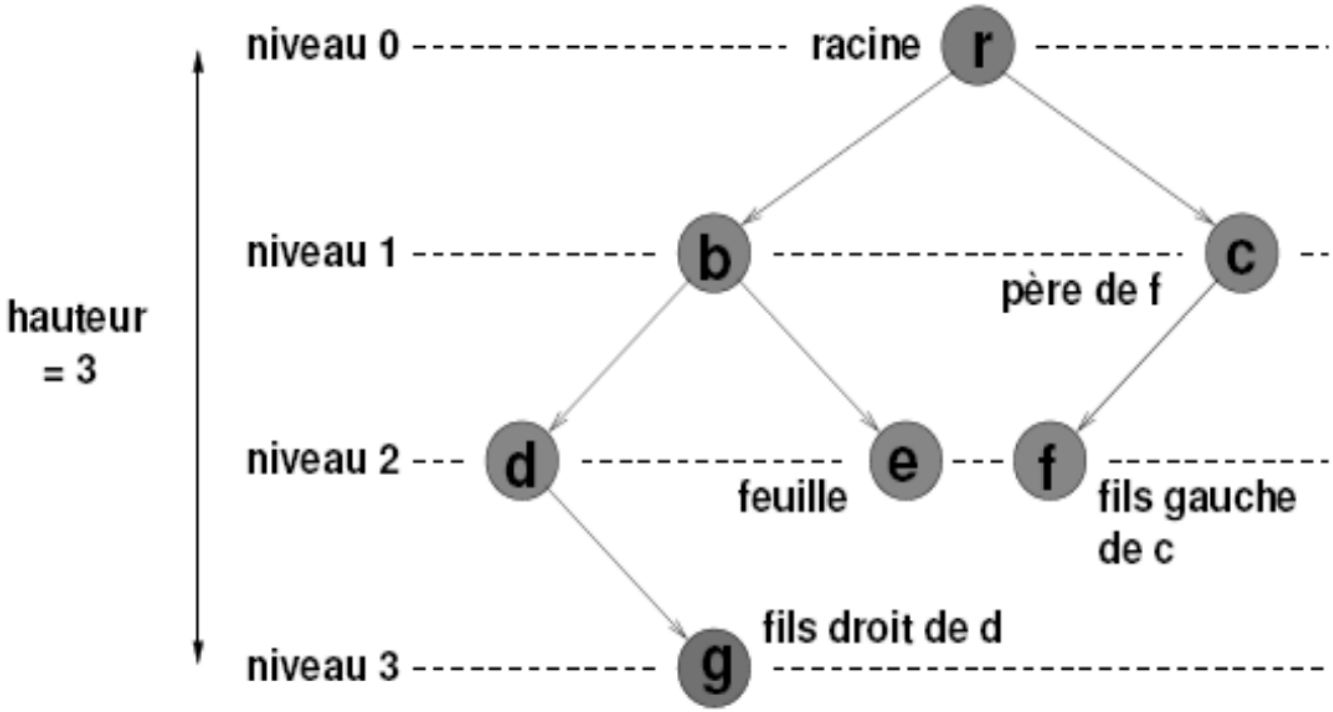
Arbres Binaires

(Binary Trees)

Définition

- Un arbre binaire A est :
  - soit vide (A = ( ) ou A = $\emptyset$),
  - soit de la forme A = <r, A1, A2>, c-à-d composé :
    1. d'un nœud r appelé racine contenant un élément
    2. et de deux arbres binaires disjoints A1 et A2, appelés respectivement sous arbre gauche (ou fils gauche) et sous arbre droit (ou fils droit).

Exemple d'arbre binaire



Représentation en arbre binaire d'un arbre général

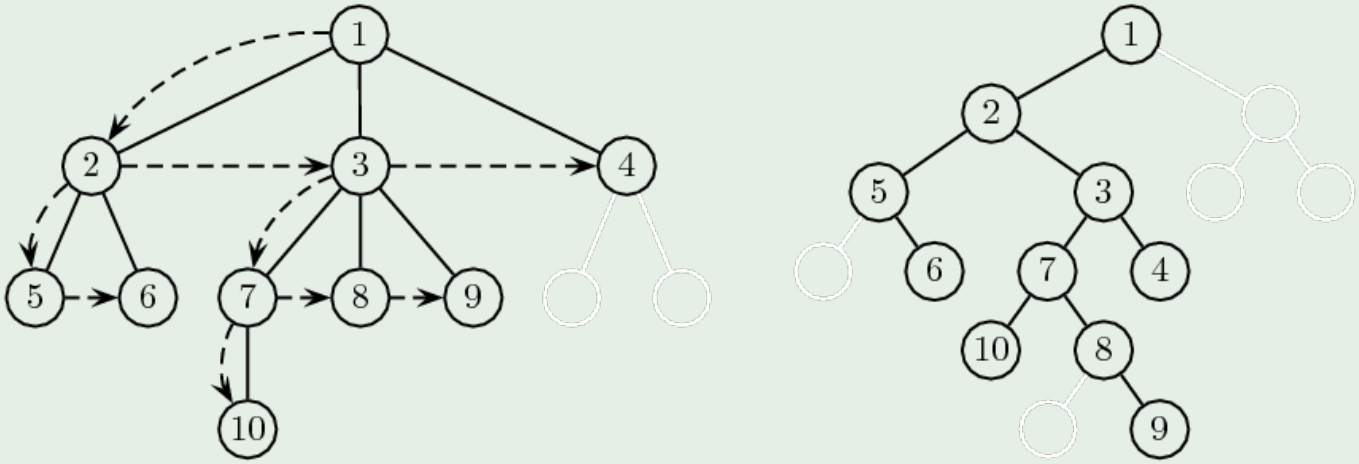
- Un arbre binaire peut être utilisé pour représenter un arbre général où les nœuds peuvent avoir un nombre quelconque de fils.

- Cette représentation, appelée "fils gauche-frère droit " (ou "fils aîné-frère droit") , utilise un arbre binaire dans lequel le fils gauche sera le fils aîné du nœud courant et le fils droit le frère cadet du nœud courant.

- Noter que la racine de l'arbre binaire n'a pas de fils droit.

Représentation en arbre binaire d'un arbre général

Exemple



Parcours d'arbre binaire

- Un parcours d'arbre permet d'accéder à chaque nœud de l'arbre :
  - Un traitement (test, affichage, comptage, etc.), dépendant de l'application considérée, est effectué sur l'information portée par chaque nœud
  - Chaque parcours de l'arbre définit un ordre sur les nœuds
- On distingue :
  - Les parcours de gauche à droite (le fils gauche d'un nœud précède le fils droit) ;
  - Les parcours de droite à gauche (le fils droit d'un nœud précède le fils gauche).
- On ne considèrera que les parcours de gauche à droite
- On distingue aussi deux catégories de parcours d'arbres :
  - Les parcours en profondeur ;
  - Les parcours en largeur

Parcours en profondeur

- Soit un arbre binaire A = <r, A1, A2>
- On définit trois parcours en profondeur de cet arbre :
  - Le parcours préfixe ;
  - Le parcours infixe ou symétrique ;
  - Le parcours postfixe ou suffixe .

Parcours en profondeur : Parcours préfixe

- En abrégé RGD (Racine, Gauche, Droit)
- Consiste à effectuer dans l'ordre :
  - Le traitement de la racine r ;
  - Le parcours préfixe du sous arbre gauche A1 ;
  - Le parcours préfixe du sous arbre droit A2.
- L'ordre correspondant s'appelle l'ordre préfixe

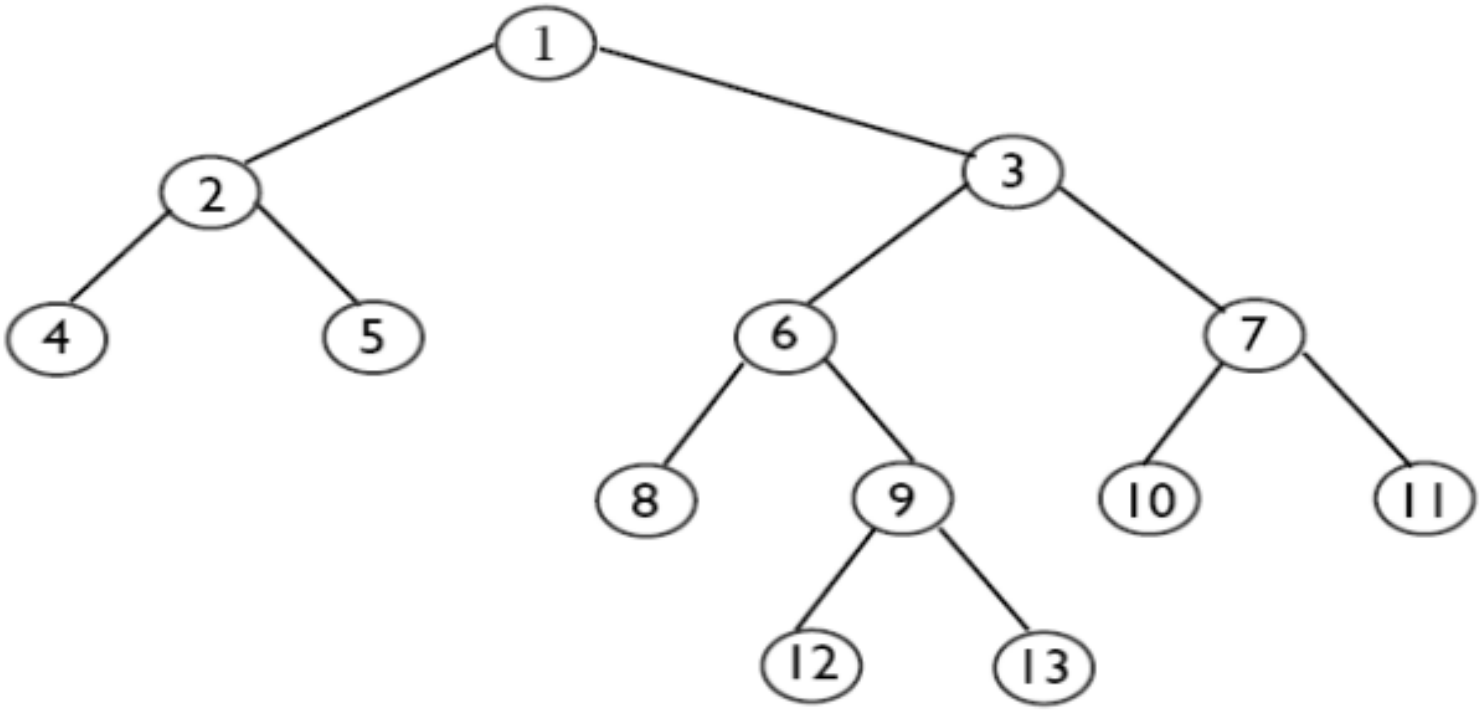
Parcours en profondeur : Parcours infixe ou symétrique

- En abrégé GRD (Gauche, Racine, Droit)
- Consiste à effectuer dans l'ordre :
  - Le parcours infixe du sous arbre gauche A1 ;
  - Le traitement de la racine r ;
  - Le parcours infixe du sous arbre droit A2.
- L'ordre correspondant s'appelle l'ordre infixe

Parcours en profondeur : parcours postfixe ou suffixe

- En abrégé GDR (Gauche, Droit, Racine)
- Consiste à effectuer dans l'ordre :
  - Le parcours postfixe du sous arbre gauche A1 ;
  - Le parcours postfixe du sous arbre droit A2 ;
  - Le traitement de la racine r.
- L'ordre correspondant s'appelle l'ordre suffixe

Exemple de Parcours en profondeur



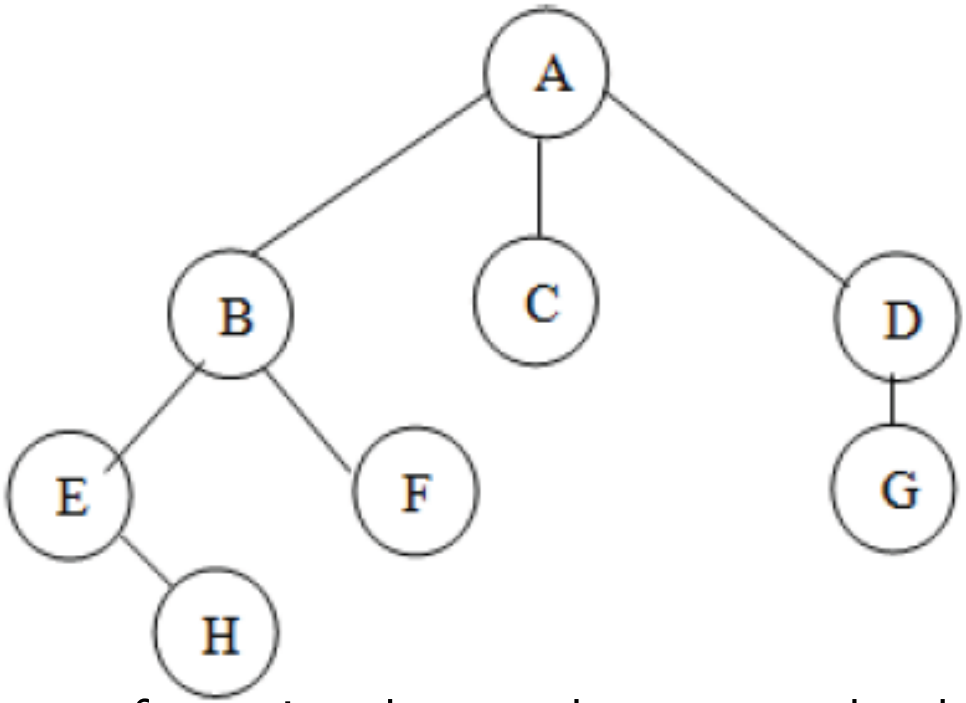
- Le parcours préfixe affiche les nœuds dans l'ordre : 1, 2, 4, 5, 3, 6, 8, 9, 12, 13, 7, 10, 11
- Le parcours infixe affiche les nœuds dans l'ordre : 4, 2, 5, 1, 8, 6, 12, 9, 13, 3, 10, 7, 11
- Le parcours postfixe affiche les nœuds dans l'ordre : 4, 5, 2, 8, 12, 13, 9, 6, 10, 11, 7, 3, 1

Parcours en largeur

- On explore les nœuds :
  - niveau par niveau,
  - de gauche à droite,
  - en commençant par la racine.
- Exemple :
  - Le parcours en largeur de l'arbre de la figure précédente affiche la séquence d'entiers suivante : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13

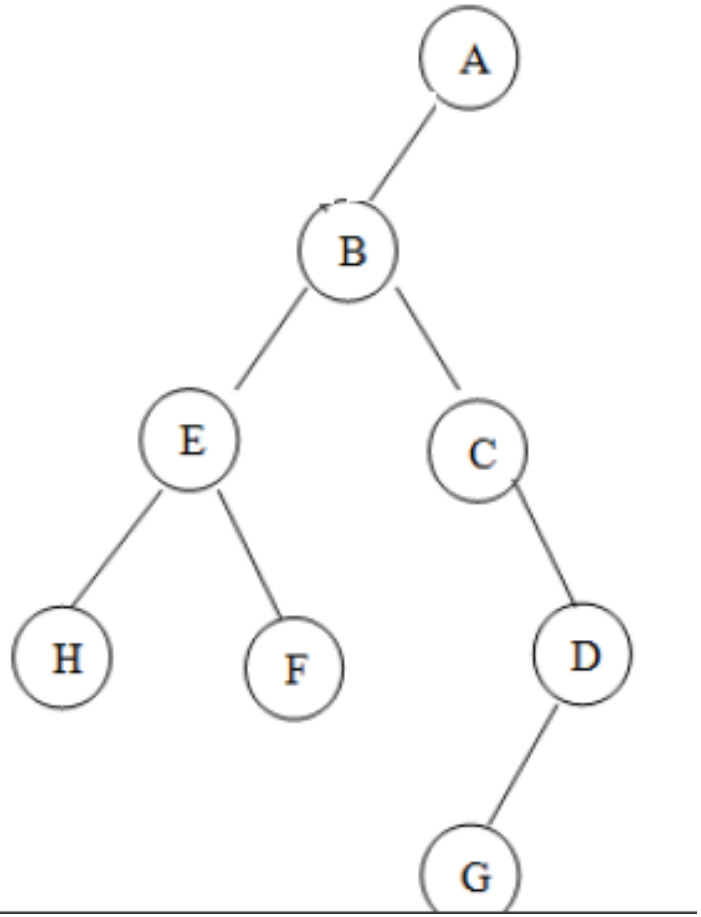
Exercice

On considère l'arbre suivant :



- Donner la transformation de cet arbre en un arbre binaire et le résultat de parcours infixe, prefixe et postfixe.

Exemple de Parcours en profondeur



- Le parcours préfixe affiche les nœuds dans l'ordre : A B E H F C D G
- Le parcours infixe affiche les nœuds dans l'ordre : H E F B C G D A
- Le parcours postfixe affiche les nœuds dans l'ordre : H F E G D C B A

Type Abstrait Arbre_Binaire

Type Arbre_Binaire
Utilise Nœud, Élément, Booléen
Opérations
arbre_vide : $\rightarrow$ Arbre_Binaire
est_vide : Arbre_Binaire $\rightarrow$ Booléen
cons : Pnœud $\times$ Arbre_Binaire $\times$ Arbre_Binaire $\rightarrow$ Arbre_Binaire
racine : Arbre_Binaire $\rightarrow$ Pnœud
gauche : Arbre_Binaire $\rightarrow$ Arbre_Binaire
droite : Arbre_Binaire $\rightarrow$ Arbre_Binaire
contenu : Pnœud $\rightarrow$ Élément
Préconditions
racine(A) est-défini-ssi est_vide(A) = faux
gauche(A) est-défini-ssi est_vide(A) = faux
droite(A) est-défini-ssi est_vide(A) = faux
Axiomes
Soit, r : Nœud, A1, A2 : Arbre_Binaire
racine(<r, A1, A2>) = r
gauche(<r, A1, A2>) = A1
droite(<r, A1, A2>) = A2

Opérations sur un Arbre Binaire (1)

- arbre_vide : $\rightarrow$ Arbre_Binaire
  - opération d'initialisation ; crée un arbre binaire vide.
- est_vide : Arbre_Binaire $\rightarrow$ Booléen
  - teste si un arbre binaire est vide ou non.
- cons : Pnœud $\times$ Arbre_Binaire $\times$ Arbre_Binaire $\rightarrow$ Arbre_Binaire
  - cons(r,G,D) construit un arbre binaire dont le sous arbre gauche est G et le sous arbre droit est D, et r est le nœud racine qui contient une donnée de type Élément.
- racine : Arbre_Binaire $\rightarrow$ Pnœud
  - si A est un arbre binaire non vide alors racine(A) retourne le nœud racine de A, sinon un message d'erreur.

Opérations sur un Arbre Binaire (2)

- gauche : Arbre_Binaire $\rightarrow$ Arbre_Binaire
  - si A est un arbre binaire non vide alors gauche(A) retourne le sous arbre gauche de A, sinon un message d'erreur.
- droite : Arbre_Binaire $\rightarrow$ Arbre_Binaire
  - si A est un arbre binaire non vide alors droite(A) retourne le sous arbre droit de A, sinon un message d'erreur.
- contenu : Pnœud $\rightarrow$ Élément
  - permet d'associer à chaque nœud d'un arbre binaire une information de type Élément.

Opérations Auxiliaires

Extension Type Arbre_Binaire
Utilise Entier, Booléen
Opérations
taille : Arbre_Binaire → Entier
hauteur : Arbre_Binaire → Entier
feuille : Arbre_Binaire → Booléen
Pré-conditions
Axiomes
Soit, r : Noeud, A1, A2 : Arbre_Binaire
taille(arbre_vide) = 0
taille(<r, A1, A2>) = 1 + taille(A1) + taille(A2)
hauteur(arbre_vide) = -1
si hauteur(A1) > hauteur(A2) alors hauteur(<r, A1, A2>) = 1+hauteur(A1)
sinon hauteur(<r, A1, A2>) = 1 + hauteur(A2)
si est_vide(A) = faux et est_vide(gauche(A)) = vrai
et est_vide(droit(A)) = vrai
alors feuille(A) = vrai
sinon feuille(A) = faux

Représentations d'un arbre binaire

- Représentation par tableau (par contiguïté)
- Représentation par pointeurs (par chaînage)

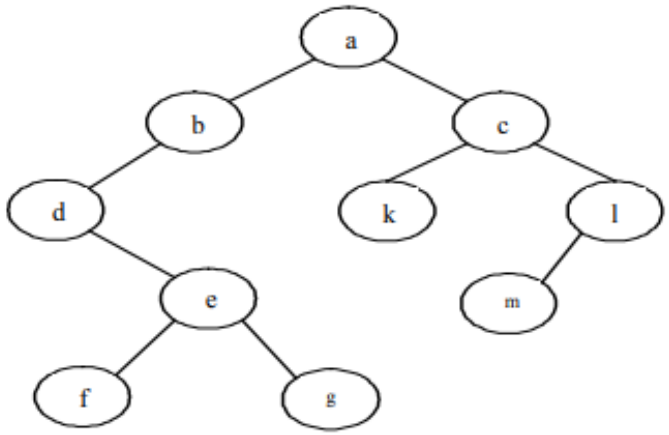
Représentation contiguë d'un arbre binaire

- On caractérise un arbre binaire par :
  - sa taille (nombre de nœuds) ;
  - sa racine (indice de son emplacement dans le tableau de nœuds) ;
  - un tableau de nœuds.
- Chaque nœud contient trois données :
  - une information de type Élément ;
  - deux entiers (indices dans le tableau désignant respectivement l'emplacement des fils gauche et droit du nœud).

Représentation contiguë d'un arbre binaire

```
#define NB_MAX_NOEUDS 15
typedef int Element;
typedef struct noeud {
Element val;
int fg;
int fd;
} Noeud;
typedef Noeud TabN[Nb_MAX_NOEUDS];
typedef struct arbre {
int nb_noeuds;
int racine;
TabN les_noeuds;
} Arbre_Binaire;
```

Représentation contiguë d'un arbre binaire



nb_noeuds	racine
10	2

les_noeuds

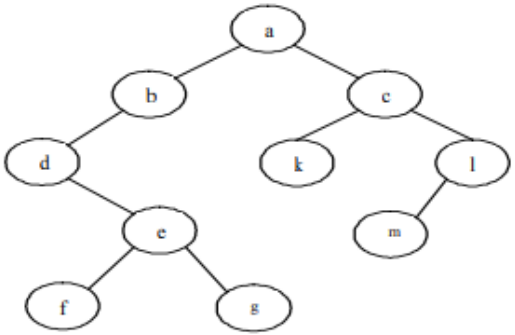
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
val		d	a	g	b	c		f	m	e	l		k		
fg		-1	4	-1	1	12		-1	-1	7	8		-1		
fd		9	5	-1	-1	10		-1	-1	3	-1		-1		

Autre représentation contiguë d'un arbre binaire

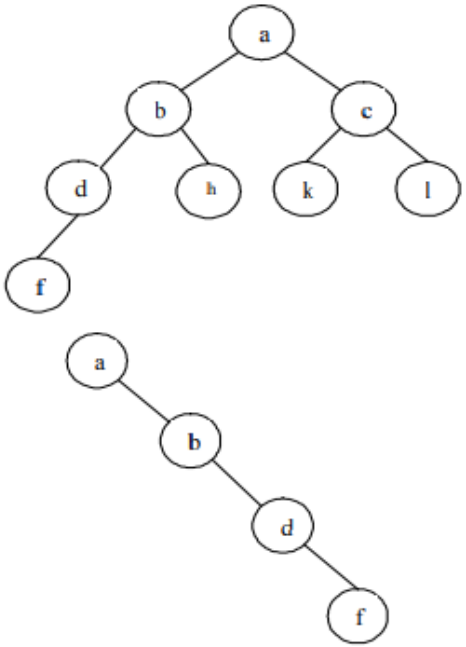
- Repose sur l'ordre hiérarchique (numérotation des nœuds niveau par niveau et de gauche à droite)
- On rappelle que pour stocker un arbre binaire de hauteur h, il faut un tableau de $2^{h+1} - 1$ éléments
- On organise le tableau de la façon suivante :
  - Le nœud racine a pour indice 0 (en langage C) ;
  - Soit le nœud d'indice i dans le tableau, son fils droit a pour indice 2i +1, et son fils gauche a pour indice 2(i+1).
- Représentation idéale pour les arbres binaires parfaits. En effet, elle ne gaspille pas d'espace.

Autre représentation contiguë d'un arbre binaire

- Exemples



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
a	b	c	d		k	l		e					m				f	g



0	1	2	3	4	5	6	7
a	b	c	d	h	k	l	f

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
a		b				d								f

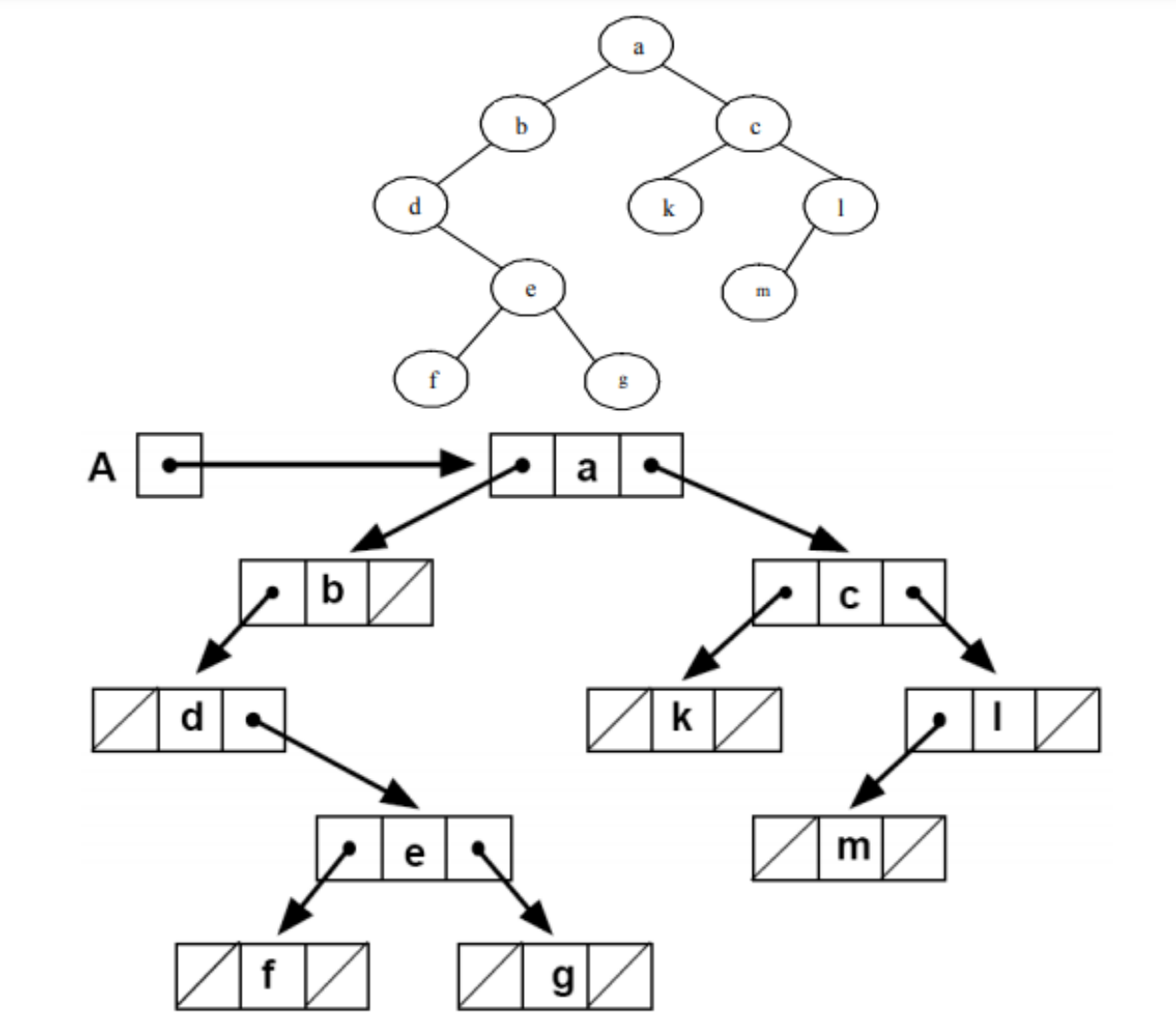
Représentation chaînée d'un arbre binaire

- Chaque nœud a trois champs :
  - val (l'élément stocké dans le nœud) ;
  - fg (pointeur sur fils gauche) ;
  - fd (pointeur sur fils droit).
- Un arbre est désigné par un pointeur sur sa racine
- Un arbre vide est représenté par le pointeur NULL

Représentation chaînée en C d'un arbre binaire

```
typedef int Element;
typedef struct noeud *Pnoeud;
typedef struct noeud {
Element val;
Pnoeud fg;
Pnoeud fd;
} Noeud;
typedef Noeud *Arbre_Binaire;
```

Représentation chaînée d'un arbre binaire



Réalisation chaînée d'un arbre binaire

```
Arbre_Binaire arbre_vide() {
return NULL;
}
Booleen est_vide(Arbre_Binaire A) {
return A == NULL ;
}
Pnoeud nouveau_noeud(Element e) {
// faire une allocation mémoire et placer l'élément e
// en cas d'erreur d'allocation, le pointeur renvoyé est NULL
Pnoeud p = (Pnoeud) malloc(sizeof(Noeud));
if (p != NULL) {
p->val = e;
p->fg = NULL;
p->fd = NULL;
}
return (p);
}
```

Réalisation chaînée d'un arbre binaire

```
Arbre_Binaire cons(Pnoeud r, Arbre_Binaire G, Arbre_Binaire D) {
r->fg = G ;
r->fd = D ;
return r ;}
Pnoeud racine(Arbre_Binaire A) {
// précondition : A est non vide !
if (estvide(A)) {printf("Erreur : Arbre vide !\n");exit(-1);}
return A ;}
Arbre_Binaire gauche(Arbre_Binaire A) {
// précondition : A est non vide !
if (estvide(A)) {printf("Erreur : Arbre vide !\n");exit(-1);}
return A->fg ;}
Arbre_Binaire droite(Arbre_Binaire A) {
// précondition : A est non vide !
if (estvide(A)) {printf("Erreur : Arbre vide !\n");exit(-1);}
return A->fd ;}
Element contenu(Pnoeud n) {
return n->val;
}
```

Exemples d'applications d'arbres binaires

- Recherche dans un ensemble de valeurs :
  - Les arbres binaires de recherche ;
- Tri d'un ensemble de valeurs :
  - Le parcours GRD d'un arbre binaire de recherche ;
- Représentation d'une expression arithmétique :
  - Un parcours GDR pour avoir une notation postfixée ;
- Méthodes de compression :
  - Le codage de Huffman utilisant des arbres binaires ;
  - La compression d'images utilisant des quadtrees (arbres quaternaires, ou chaque nœud non feuille a exactement quatre fils) ;

Arbres Binaires de Recherche

(Binary Search Trees)

Notion d'arbre binaire de recherche

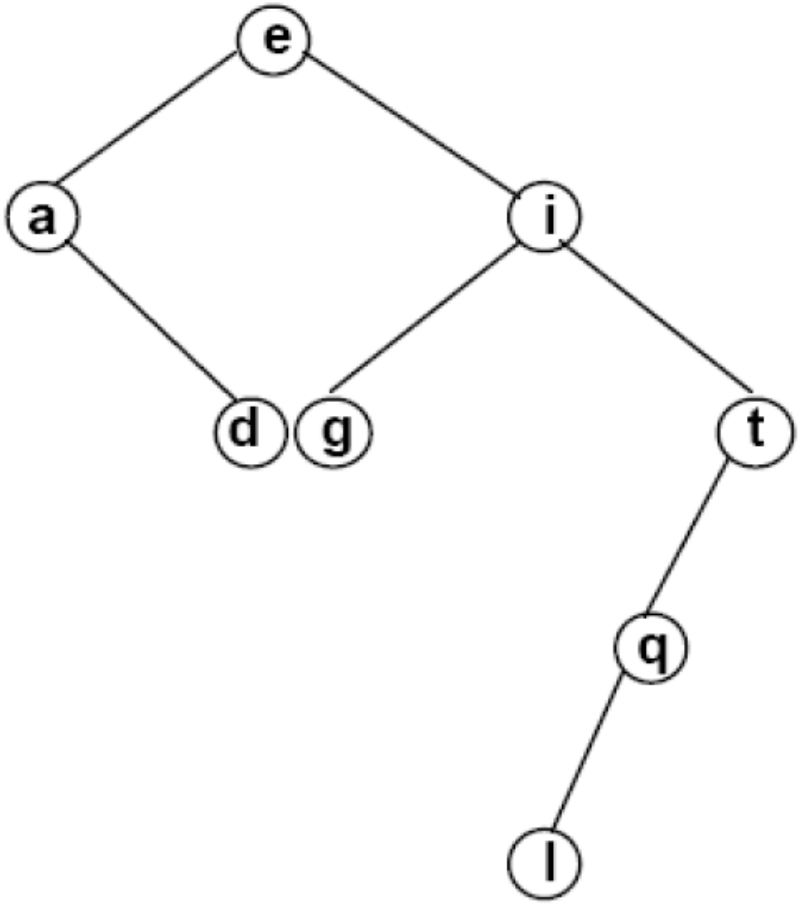
- C'est un arbre binaire particulier :
  - Permet d'obtenir un algorithme de recherche proche dans l'esprit de la recherche dichotomique ;
  - Pour lequel les opérations d'ajout et de suppression d'un élément sont aussi efficaces.
- Cet arbre utilise l'existence d'une relation d'ordre sur les éléments, représentée par une fonction clé, à valeur entière.

Arbre binaire de recherche : Définition

- Un arbre binaire de recherche (binary search tree en anglais), en abrégé ABR, est un arbre binaire tel que pour tout nœud :
  - les clés de tous les nœuds du sous-arbre gauche sont inférieures ou égales à la clé du nœud,
  - les clés de tous les nœuds du sous-arbre droit sont supérieures à la clé du nœud.
- Chaque nœud d'un arbre binaire de recherche désigne un élément qui est caractérisé par une clé (prise dans un ensemble totalement ordonné) et des informations associées à cette clé.
- Dans toute illustration d'un arbre binaire de recherche, seules les clés sont représentées. On supposera aussi que toute clé identifie de manière unique un élément.

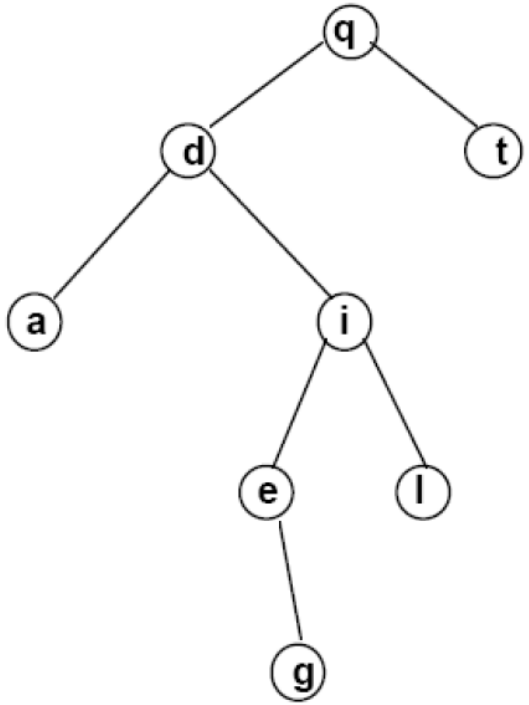
Arbre binaire de recherche

- L'arbre de la figure suivante est un arbre binaire de recherche
- Cet arbre représente l'ensemble : E = {a, d, e, g, i, l, q, t} muni de l'ordre alphabétique



Arbre binaire de recherche

- Plusieurs représentations possibles d'un même ensemble par un arbre binaire de recherche
- En effet, la structure précise de l'arbre binaire de recherche est déterminée :
  - par l'algorithme d'insertion utilisé,
  - et par l'ordre d'arrivée des éléments.
- Exemple :
  - L'arbre binaire de recherche de la figure qui suit représente aussi E = {a, d, e, g, i, l, q, t}



Opérations sur les arbres binaires de recherche

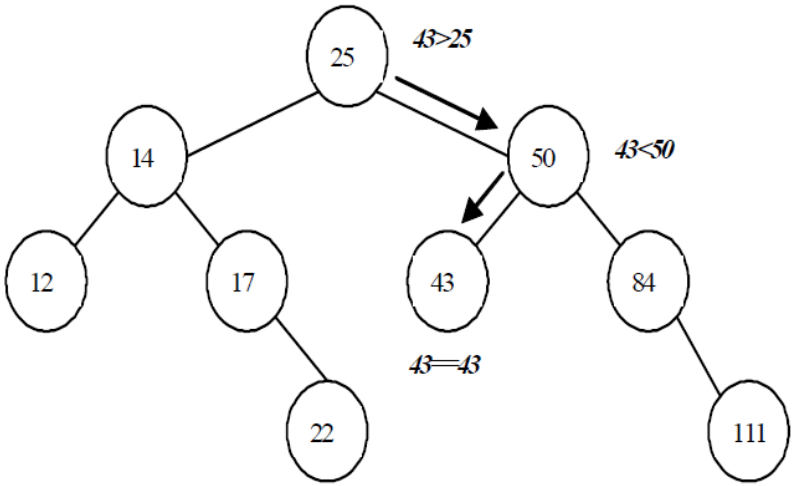
- Le type abstrait arbre binaire de recherche, noté Arbre_Rech, est décrit de la même manière que le type Arbre_Binaire
- On reprend les opérations de base des arbres binaires, excepté le fait que dans des arbres binaires de recherche, on suppose l'existence de l'opération clé sur le type abstrait Élément
- On définit, en tenant compte du critère d'ordre, les opérations spécifiques de ce type d'arbre concernant :
  - la recherche d'un élément dans l'arbre ;
  - l'insertion d'un élément dans l'arbre ;
  - la suppression d'un élément de l'arbre.

Recherche d'un élément

- Principe de l'algorithme :
  - On compare la clé de l'élément cherché à la clé de la racine de l'arbre ;
  - Si la clé est supérieure à la clé de la racine, on effectue une recherche dans le fils droit ;
  - Si la clé est inférieure à la clé de la racine, on effectue une recherche dans le fils gauche ;
  - La recherche s'arrête quand on ne peut plus continuer (échec) ou quand la clé de l'élément cherché est égale à la clé de la racine d'un sous arbre (succès).

Recherche d'un élément : Exemple

- la figure suivante illustre la recherche de l'élément de clé 43 dans un arbre binaire de recherche.
- Les flèches indiquent le chemin de la recherche



Recherche d'un élément : Spécification

Extension Type Arbre_Rech
Utilise Élément, Booléen
Opérations
Rechercher : Élément $\times$ Arbre_Rech $\rightarrow$ Booléen
Axiomes
Soit, x : Élément, r : Nœud, G, D : Arbre_Rech
Rechercher(x, arbre_vide) = faux
si clé(x) = clé(contenu(r))
alors Rechercher(x, <r, G, D>) = vrai
si clé(x) < clé(contenu(r))
alors Rechercher(x, <r, G, D>) = Rechercher(x, G)
si clé(x) > clé(contenu(r))
alors Rechercher(x, <r, G, D>) = Rechercher(x, D)

Recherche d'un élément

Réalisation en C

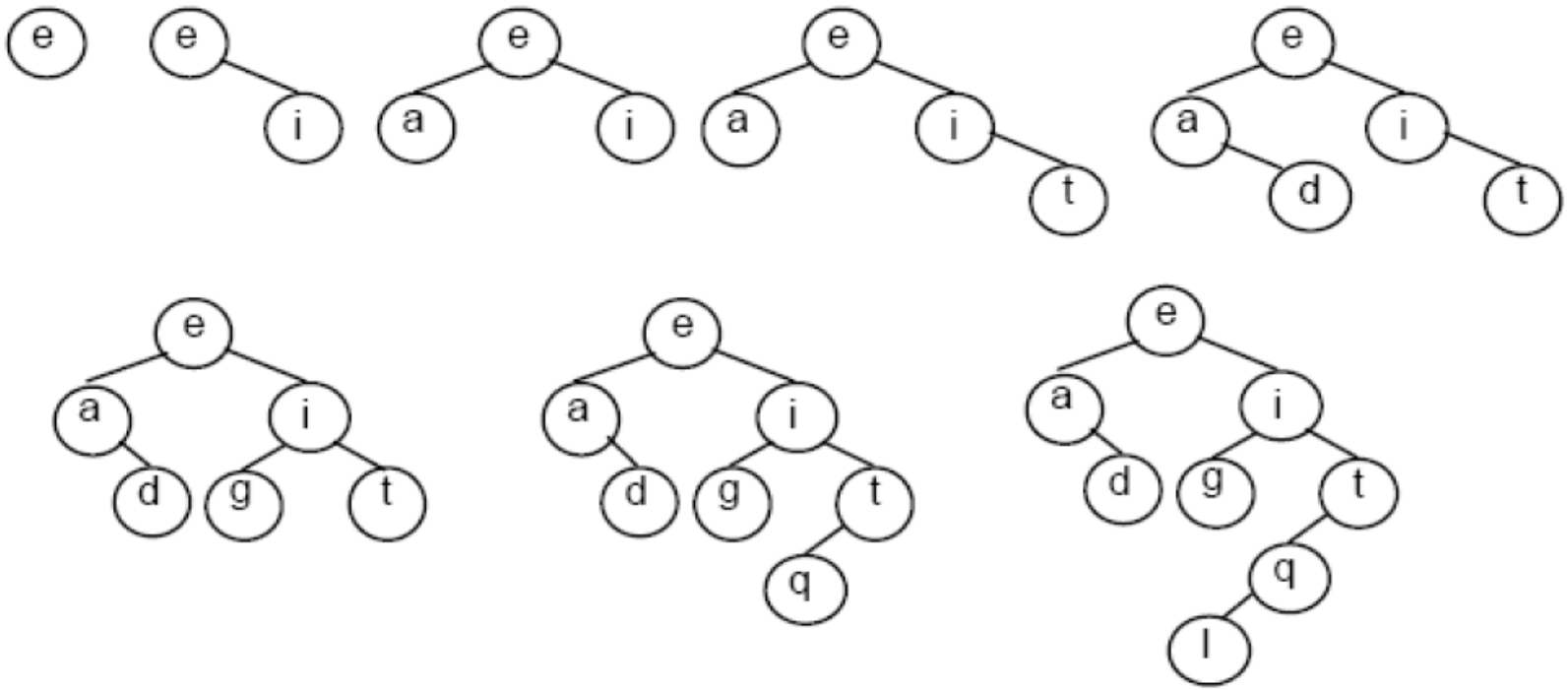
```
Booleen Rechercher (Arbre_Rech A, Element e) {
if ( est_vide(A) == vrai )
return faux; // e n'est pas dans l'arbre
else {
if ( e == A->val )
return vrai; // e est dans l'arbre
else if ( e < A->val )
// on poursuit la recherche dans le SAG du
// nœud courant
return Rechercher(A->fg , e);
else // on poursuit la recherche dans le SAD du
// nœud courant
return Rechercher(A->fd , e);
}
}
```

Ajout d'un élément

- La technique d'ajout spécifiée ici est dite "ajout en feuille ", car tout nouvel élément se voit placé sur une feuille de l'arbre
- Le principe est simple :
  - si l'arbre initial est vide, le résultat est formé d'un arbre binaire de recherche réduit à sa racine, celle-ci contenant le nouvel élément ;
  - sinon, l'ajout se fait (récursivement) dans le fils gauche ou le fils droit, suivant que l'élément à ajouter est de clé inférieure ou supérieure à celle de la racine.
- Remarque :
  - si l'élément à ajouter est déjà dans l'arbre, l'hypothèse d'unicité des éléments pour certaines applications fait qu'on ne réalise pas l'ajout

Ajout d'un élément : Exemple

- Les figures suivantes illustrent l'ajout successif de e, i, a, t, d, g, q et l dans un arbre binaire de recherche, initialement vide



Ajout "en feuille" d'un élément

Réalisation en C

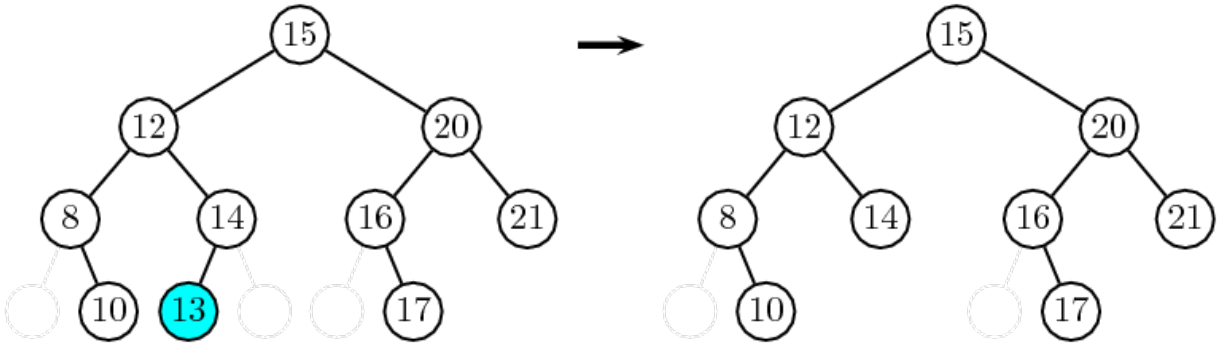
```
Arbre_Rech Ajouter_feuille(Element x, Arbre_Rech A) {
if (est_vide(A)) {
Pnoeud r = nouveau_noeud(x);
if (r == NULL) {
printf("Erreur : Pas assez de mémoire !\n");
exit(-1);
}
return cons(r, arbre_vide(), arbre_vide());
}
elseif (x > contenu(racine(A)))
return cons(A, gauche(A), Ajouter_feuille(x, droite(A)));
else
return cons(A, Ajouter_feuille(x, gauche(A)), droite(A));
}
```

Suppression d'un élément

- La suppression est délicate :
  - Il faut réorganiser l'arbre pour qu'il vérifie la propriété d'un arbre binaire de recherche
- La suppression commence par la recherche du nœud qui porte l'élément à supprimer. Ensuite, il y a trois cas à considérer, selon le nombre de fils du nœud à supprimer :
  - si le nœud est sans fils (une feuille), la suppression est immédiate ;
  - si le nœud a un seul fils, on le remplace par ce fils ;
  - si le nœud a deux fils (cas général), on choisit de remplacer ce nœud, soit par le plus grand élément de son sous arbre gauche (son prédécesseur), soit par le plus petit élément de son sous arbre droit (son successeur).

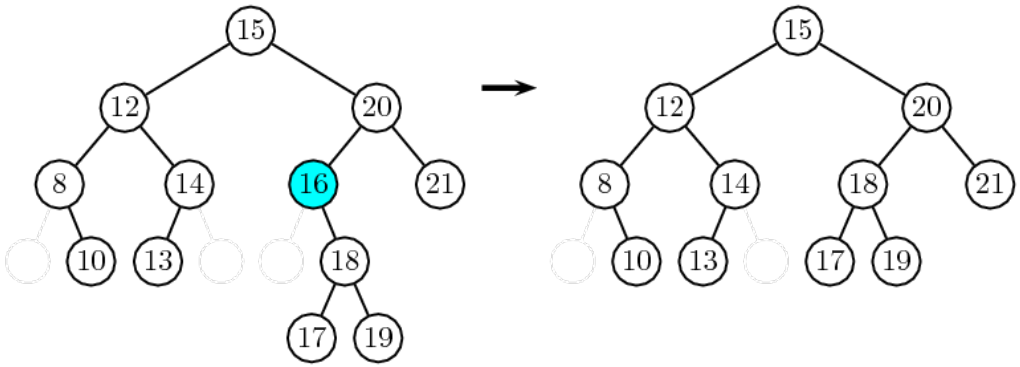
Suppression d'un élément : Exemple 1

- La figure qui suit illustre la suppression de la feuille qui porte la clé 13



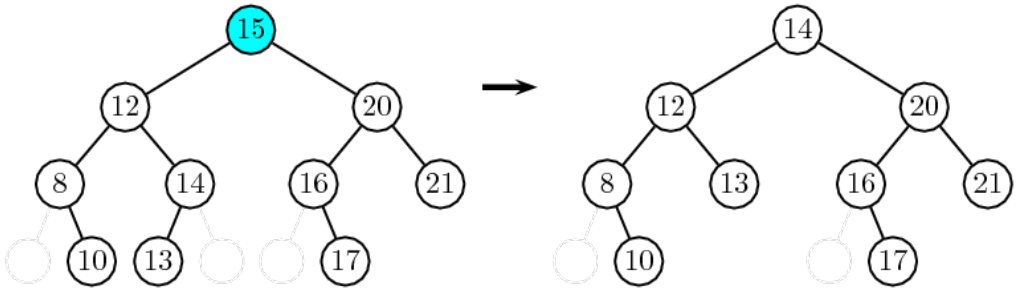
Suppression d'un élément : Exemple 2

- La figure qui suit illustre la suppression du nœud qui porte la clé 16
- Ce nœud n'a qu'un seul fils ; le sous arbre de racine portant la clé 18
- Ce sous arbre devient fils gauche du nœud qui porte la clé 20



Suppression d'un élément : Exemple 3

- La figure qui suit illustre le cas d'un nœud à deux fils.
- La clé 15 à supprimer se trouve à la racine de l'arbre. La racine a deux fils ; on choisit de remplacer sa clé par la clé de son prédécesseur.
- Ainsi, la clé 14 est mise à la racine de l'arbre. On est alors ramené à la suppression du nœud du prédécesseur.
- Comme le prédécesseur est le nœud le plus à droite du sous arbre gauche, il n'a pas de fils droit, donc il a zéro ou un fils, et sa suppression est couverte par les deux premiers cas.



Suppression d'un élément : Cas général

- On choisit ici de remplacer le nœud à supprimer par son prédécesseur (le nœud le plus à droite de son sous arbre gauche)
- On a besoin de deux opérations supplémentaires :
  - une opération Max qui retourne l'élément de clé maximale dans un arbre binaire de recherche ;
  - une opération SupprimerMax qui retourne l'arbre privé de son plus grand élément.

Suppression d'un élément

Réalisation en C

```
Pnoeud Max_arbre(Arbre_Rech A){
      if (droite(A)==NULL)
          return A;
      else
          Max_arbre(A->fd);
      }


Arbre_Rech SupprimerMax(Arbre_Rech A){
          if (droite(A)==NULL)
              return gauche(A);
          else
              return cons(A,gauche(A),SupprimerMax(droite(A)));
          }
```

Suppression d'un élément

Réalisation en C

```
Arbre_Rech Supprimer(Arbre_Rech A, Element e){
  if (A==NULL) return A;
  elseif (e> contenu(racine(A)))
      return cons(A,gauche(A), Supprimer(droite(A),e));
  elseif (e< contenu(racine(A)))
     return cons(A,Supprimer(gauche(A),e),droite(A));
  else {
       if (droite(A)==NULL) return gauche(A);
       elseif(gauche(A)==NULL) return droite(A);
       else
       return cons(Max_arbre(gauche(A)),SupprimerMax(gauche(A)),
                                                    droite(A));
        }
     }
```