

Cours n° 3 : Les Arbres (1)

- Les arbres en théorie des graphes et arbres binaires
 - Introduction
 - En théorie des graphes :
 - Graphes, arbres, arborescences
 - Les arbres informatiques
 - Le TDA ARBRE BINAIRE
 - Mises en œuvre des arbres binaires

Cours n° 3 : Les Arbres (2)

- Parcours d'arbres et récursivité
 - Parcours récursifs et itératifs d'arbres binaires
 - parcours : examiner tous les noeuds pour effectuer un traitement
 - parcours en profondeur à main gauche
 - parcours en largeur (par niveaux, hiérarchique)
 - Parenthèse sur la récursivité

Cours n° 3 : Les Arbres (3)

- Les Arbres Généraux
 - Le TDA ARBRE GÉNÉRAL
 - Mises en œuvre des arbres (généraux)
 - par tableau (étiquette père)
 - par tableau et liste chaînée des fils
 - par faux-pointeurs et liste chaînée des fils
 - par pointeurs (nombre maxi de fils)
 - par arbre binaire (Premier Fils, FrèreDroit)
 - Équivalence avec les arbres binaires

Introduction

- Partie 1 :
 - Arbres en théorie de graphes et arbres binaires
- Partie 2 : parcours d'arbres
 - Parcours d'arbres binaires, récursivité, parcours récursifs et itératifs d'arbres binaires, parenthèse sur la récursivité
- Partie 3 : arbres généraux
 - Le TDA ARBRE GÉNÉRAL
 - Mises en œuvre et équivalence avec les arbres binaires

C3-1 : Les Arbres en théorie des graphes et arbres binaires

1. Introduction
2. **En théorie des graphes :**
Graphes, arbres, arborescences
3. Les arbres informatiques
4. Le TDA ARBRE BINAIRE
5. Mises en œuvre des arbres binaires

En théorie des graphes

- Graphes
 - Définition
 - Exemples
 - Terminologie
- Arbres
- Arborescences

Graphes

- Définition

Un graphe $G = \langle S, A \rangle$ est

- Un ensemble fini S d'objets appelés "sommets"
- Et un ensemble A de relations entre ces sommets appelées "arêtes"

A est un ensemble de paires de sommets

- Exemples

- Ex1 : liaisons aériennes entre villes
- Ex2 : contrôle dans un programme

Graphes : terminologie (1)

- Orientation
 - Graphe **orienté** : A un ensemble fini de couples de sommets appelés "arcs" (ordonné)
 - Graphe **non orienté** : A un ensemble fini de paires de sommets appelés "arêtes"
- (S, s') est un arc de s vers s' noté $s \rightarrow s'$
 - S est l'origine de l'arc (s, s') et s' son extrémité
 - S est un prédécesseur de s' , s' est un successeur de s
- $\{S, s'\}$ est une arête reliant s et s'
 - S et s' sont les extrémités de l'arête
 - S est un successeur de s' , s' est un successeur de s

Graphes : terminologie (2)

- Graphe orienté, non orienté

Étant donné un graphe orienté on appelle graphe non orienté sous-jacent le graphe non orienté obtenu en oubliant l'orientation des arcs

- Informations supplémentaires

- Graphe **étiqueté** :

Les sommets du graphe portent une information appelée "étiquette" du sommet

- Graphe **valué** (ou pondéré) :

À chaque arête (arc) du graphe on associe un nombre appelé le "coût" ou le "poids" de l'arête (arc)

Graphes : terminologie (3)

- **Chemins (resp. Chaîne)**
 - Un chemin reliant un sommet s_0 à un sommet s_n est une suite de sommets où deux éléments successifs sont reliés par une arc (resp. Une arête) : $(s_0, s_1, s_2, \dots, s_n)$, $\forall 0 \leq i \leq n \quad (s_i, s_{i+1}) \in A$
- **Longueur d'un chemin :**
 - Nombre d'arcs empruntés
- **Chemin élémentaire**
 - Chemin dont tous les sommets sont distincts
- **Circuit (resp. Cycle)**
 - Chemin de longueur non nulle reliant un sommet à lui-même

Graphes : terminologie (4)

- Forte connexité (resp. Connexité).
 - Un graphe orienté est fortement connexe si tout couple de sommets distincts est relié par un chemin.
 - Un graphe non orienté est connexe si toute paire de sommets distincts est reliée par une chaîne.
- Composante fortement connexe (resp. Connexe).
 - Un sous-graphe d'un graphe orienté (non orienté) est une composante fortement connexe (resp. Connexe).
 - S'il est fortement connexe (resp. Connexe).
 - Et s'il est maximal .

C3-1 : Les Arbres en théorie des graphes et arbres binaires

1. Introduction
2. **En théorie des graphes :**
Graphes, arbres, arborescences
3. Les arbres informatiques
4. Le TDA ARBRE BINAIRE
5. Mises en œuvre des arbres binaires

Les arbres

- Définition : un arbre est un graphe non orienté connexe sans cycle
- Propriété: soit $G = \langle S, A \rangle$ un graphe non orienté ayant n sommets
Les propositions suivantes sont équivalentes :
 - G est un arbre
 - G est connexe et si on supprime une arête il ne l'est plus
 - G est sans cycle et si on ajoute une arête on crée un cycle
 - G est connexe et comporte $n-1$ arêtes
 - G est sans cycle et comporte $n-1$ arêtes
 - Tout couple de sommet est relié par une chaîne et une seule

Racine d'un arbre

- Racine d'un arbre : un sommet r distingué
- Arbre enraciné : un arbre qui possède une racine
- Une racine n'existe pas toujours

Les arborescences

- Une arborescence est un graphe orienté ayant un sommet particulier appelé racine tel que tous les sommets de l'arborescence sont reliés à la racine par un chemin unique
 - I.E. Une arborescence est un graphe orienté dont le graphe non orienté sous-jacent est un arbre enraciné
- Arborescence ordonnée
 - Une arborescence est ordonnée si l'ensemble des successeurs de la racine est ordonné

C3-1 : Les Arbres en théorie des graphes et arbres binaires

1. Introduction
2. En théorie des graphes :
Graphes, arbres, arborescences
3. **Les arbres informatiques**
4. Le TDA ARBRE BINAIRE
5. Mises en œuvre des arbres binaires

Les arbres informatiques : définitions

- Définition informelle : un arbre est un ensemble de sommets organisé de façon hiérarchique tel que :
 - Il existe un sommet unique appelé racine de l'arbre qui n'a pas de supérieur
 - Tous les autres sommets sont atteints à partir de la racine par une unique voie hiérarchique (une branche, un chemin)
- Définition récursive (et constructive) : un arbre est la donnée d'une racine et d'une liste (éventuellement vide) d'arbres disjoints appelés sous-arbres
$$A = < r > \text{ ou } < r, (A_1, \dots, A_n) >, A_i \text{ arbres disjoints}$$
- Un sommet de l'arbre est la racine d'un sous-arbre

Les arbres informatiques : remarques

- Orientation ?

- Pas forcément, mais généralement on considère une orientation du haut de la hiérarchie (la racine) vers le bas (les feuilles)

- Ordre sur les successeurs ?

- Pas forcément, mais généralement l'ensemble des successeurs est ordonné de la gauche vers la droite

- Rapport math/info

- Généralement : un arbre informatique est une arborescence ordonnée en théorie des graphes
 - Mais en cas de besoin on oublie l'ordre ou l'orientation sans changer de termes
 - Arbre vide ? pas en math, parfois en info

Terminologie : généalogique ou forestière

- père d'un sommet : le prédécesseur d'un sommet
- fils d'un sommet : les successeurs d'un sommet
- frères : des sommets qui ont le même père
- grand-père, ancêtre, descendants ...
- les sommets sont souvent appelés des nœuds :
 - la racine (pas de père)
 - les feuilles ou nœuds externes ou terminaux (pas de fils)
 - les nœuds internes, non terminaux (au moins un fils)
- une branche : généralement un chemin depuis un nœud racine (quelque fois jusqu'à une feuille)

Mesures sur les arbres

- Niveau (profondeur) d'un nœud : la longueur de la branche qui la joint à la racine (nombre d'arcs)
- Hauteur d'un nœud : la longueur de la plus longue branche de ce noeud jusqu'à une feuille
- La hauteur d'un arbre : la longueur de la branche la plus longue (la hauteur de la racine)
- La taille d'un arbre : nombre de ses sommets
- La longueur totale : la somme des longueurs des branches issues de la racine

C3-1 : Les Arbres en théorie des graphes et arbres binaires

1. Introduction
2. En théorie des graphes :
Graphes, arbres, arborescences
3. Les arbres informatiques
4. **Le TDA ARBRE BINAIRES**
5. Mises en œuvre des arbres binaires

Arbres binaires

- Définition informelle

- Dans un arbre binaire tout nœud a au plus deux fils
- Un arbre binaire possède exactement deux sous-arbres (éventuellement vides)

- Définition récursive (et constructive)

- Un arbre binaire est
 - Soit vide
 - Soit composé d'une racine r et de deux arbres binaires ABG et ABD disjoints
 - ABG : arbre binaire gauche
 - ABD : arbre binaire droit

Arbres binaires particuliers

- Arbre binaire dégénéré, filiforme
 - Chaque nœud possède exactement un fils
- Arbre binaire complet (uniforme)
 - Chaque niveau est complètement rempli
 - I.E. Tout sommet est soit une feuille au dernier niveau, soit possède exactement 2 fils
- Arbre binaire parfait (presque complet)
 - Tous les niveaux sont complètement remplis sauf éventuellement le dernier et dans ce cas les feuilles sont le plus à gauche possible
- Arbre binaire équilibré
 - La différence de hauteur entre 2 frères ne peut dépasser 1

Propriétés des arbres binaires

- Un arbre binaire ayant n sommets a une hauteur h qui vérifie : $\lceil \log_2(n + 1) \rceil - 1 \leq h(a) \leq n - 1$
- Un arbre binaire de hauteur h a un nombre de sommets n qui vérifie : $h + 1 \leq n \leq 2^{h+1} - 1$
- Utilisation : mesures de complexité
 - Parcours de chaque nœud de l'arbre en $O(n)$
 - Parcours d'une branche : complexité dans le pire des cas en $O(h) = O(\log n)$
 - Objectif : avoir h minimum (arbre complet ou presque ou équilibré)

Le TDA ARBRE BINAIRE

- Utilise le TDA ELEMENT
- Primitives
 - Création et destruction
 - Construction
 - À côté (programmation fonctionnelle)
 - Sur place (programmation impérative) :
Primitives de modification d' un arbre non vide
 - Accès aux caractéristiques des nœuds
 - Test

C3-1 : Les Arbres en théorie des graphes et arbres binaires

1. Introduction
2. En théorie des graphes :
3. Les arbres informatiques
4. Le TDA ARBRE BINAIRES
5. **Mises en œuvre des arbres binaires**
 - par cellules chaînées,
 - par cellules contiguës,
 - par curseurs (faux pointeurs),
 - arbres binaires à chaînes, arbres parfaits

Représentation par cellules chaînées

- Un arbre binaire est
 - Soit un pointeur sur le nœud racine (arbre non vide)
 - Soit le pointeur NULL (arbre vide)
- Un nœud est une structure à trois champs
 - Une étiquette
 - Le sous-arbre gauche et le sous-arbre droit
- Avantages
 - Définition récursive, simple à programmer, la plus utilisée
- Inconvénients : consomme de la mémoire dynamique

Représentation par cellules contiguës

- Un arbre est une structure à deux champs
 - Un tableau où sont mémorisés les nœuds
 - Un entier qui donne l'indice de la racine dans le tableau
- Un nœud est une structure à 3 champs
 - L'étiquette du nœud
 - Les indices de ses fils gauche et droit (ou 0 si pas de fils)
- Inconvénients
 - Définition non récursive (arbre \neq sous-arbre)
- Utilisée uniquement si on traite un arbre unique

Représentation par curseurs (faux pointeurs)

- Utilisation d'une **variable globale** pour simuler la mémoire où sont mémorisés les nœuds
- Un arbre est l'indice de sa racine dans le tableau (ou 0 s'il est vide)
- Un nœud (une case du tableau) est une structure à 3 champs
 - L'étiquette du nœud, les indices de ses fils gauche et droit (ou 0 si pas de fils)
- 2 stratégies de gestion des cellules disponibles
 - Chaîner entre elles les cellules disponibles, marquer les cellules libres et parcourir le tableau pour trouver la 1re place libre

Représentation plus économiques

- Arbre parfait
 - Représenté par un tableau de n étiquettes
 - $T[i]$ étiquette du père
 - $T[2i]$ étiquette du fils gauche
 - $T[2i+1]$ étiquette du fils droit
- Arbre à chaînes
 - Horowitz p. 203
 - Idée dans la représentation par pointeurs, les feuilles gaspillent de la mémoire (lien NULL) ; remplacer les liens NULL par d'autres nœuds qui facilitent les parcours

Complexité des opérations sur les Arbres binaires

- Construire est en $o(n)$
- Modifier en $o(1)$
- Détruire est
 - En $o(n)$: pointeurs, faux pointeurs
 - En $o(1)$: cellules contiguës
- Test et accès sont en $o(1)$

Cours n° 3 : Les Arbres (2)

- Parcours récursifs et itératifs d'arbres binaires
 - Parcours en profondeur à main gauche
 - Algorithme récursif général
 - Cas particuliers (parcours préfixe, infixé, postfixe)
 - Algorithme itératif général
 - Application : calcul de la hauteur d'un arbre
 - Simplification dans le cas d'un parcours préfixe
 - Parcours en largeur (par niveaux, hiérarchique)
- Parenthèse sur la récursivité

Parcours en profondeur à main gauche

- chemin qui descend toujours le plus à gauche possible
- dans ce parcours chaque nœud est rencontré 3 fois
 - 1ière fois : à la descente à gauche dans le sous-arbre gauche
 - on applique au nœud le traitement 1
 - 2ième fois : quand le sous-arbre gauche a été parcouru, en remontant par la gauche pour descendre à gauche dans le sous-arbre droit
 - on applique au nœud le traitement 2
 - 3ième et dernière fois : quand les deux sous-arbres ont été parcourus, en remontant à droite
 - on applique au nœud le traitement 3
- si l'arbre est vide on lui applique un traitement appelé terminaison

Parcours récursif en profondeur à main gauche

Parcours (A)

- si A est vide
 - alors terminaison(A)
 - sinon
 - * traitement1 (A)
 - * Parcours(sous-arbreGauche(A))
 - * traitement2 (A)
 - * Parcours(sous-arbreDroit(A))
 - * traitement3 (A)

Cas particuliers

Lorsque le noeud est traité une seule fois

- Parcours préfixe (Racine Gauche Droit)
 - le nœud racine est traité au premier passage avant le parcours des sous-arbres
- Parcours infixe ou symétrique (Gauche Racine Droit)
 - le nœud racine est traité au second passage
 - après le parcours du sous-arbre gauche
 - et avant le parcours du sous-arbre droit
- Parcours postfixe (Gauche Droit Racine)
 - le nœud racine est traité en dernier après le parcours des sous-arbres

Parcours préfixe (R G D)

- le nœud racine est traité au premier passage avant le parcours des sous-arbres
- ParcoursPréfixe(A)
 - si A est vide
 - alors terminaison(A)
 - sinon
 - traitement (A)
 - ParcoursPréfixe(sous-arbreGauche de A)
 - ParcoursPréfixe(sous-arbreDroit de A)

Parcours infixé ou symétrique (G R D)

- le nœud racine est traité au second passage après le parcours du sous-arbre gauche et avant le parcours du sous-arbre droit
- $\text{ParcoursInfixe}(A)$
 - si A est vide
 - alors $\text{terminaison}(A)$
 - sinon
 - $\text{ParcoursInfixe}(\text{sous-arbre Gauche de } A)$
 - traitement (A)
 - $\text{ParcoursInfixe}(\text{sous-arbre Droit de } A)$

Parcours postfixe (G D R)

- le nœud racine est traité au troisième passage après le parcours des sous-arbres gauche et droit
- ParcoursPostfixe(A)
 - si A est vide
 - alors terminaison(A)
 - sinon
 - ParcoursPostfixe(sous-arbreGauche de A)
 - ParcoursPostfixe(sous-arbreDroit de A)
 - traitement (A)

"Dérécursiver"

- Pour des raisons d'efficacité (économie de temps et de mémoire) on peut être amené à "dérécursiver" certains algorithmes
- dérécursiver : transformer un algorithme récursif en algorithme itératif
- méthodes générales pour supprimer :
 - un appel récursif **terminal** : boucle tant que
 - un appel récursif **non terminal** : utilisation d'une pile

Parcours itératif en profondeur à main gauche

- idée : utiliser une pile pour mémoriser le nœud courant et la direction future
- descente (à gauche du nœud, 1ier passage)
 - effectuer le traitement 1 sur la racine
 - empiler le nœud et la direction future (descendre à dr., 2ième passage)
 - descendre à **gauche** sur le **sous-arbre gauche**
- remontée gauche (2ième passage)
 - effectuer le traitement 2 sur la racine
 - empiler le nœud et la direction future (remonter à droite, 3ième passage)
 - descendre à gauche sur le sous-arbre droit
- remontée droite (3ième passage)
 - effectuer le traitement 3 sur la racine
 - dépiler le nœud et le sens de la visite sur ce nœud (n° de passage)

Parcours itératif en profondeur à main gauche

- A = l'arbre à parcourir
- créer une pile vide P , direction = descente gauche
- tant que (P est non vide) ou (direction < \neq montée droite)
 - Si A est vide
 - alors terminaison et direction = montée droite
 - sinon si P est non vide alors selon direction
 - direction = descente gauche
 - trait 1, empiler (A, desc. dr),
 - A = ss-arbGauche (A) (desc. à g. sur le sous-arbre gauche)
 - direction = descente droite
 - trait 2, empiler (A, montée. dr),
 - A = ss-arbDroit (A) , direction = descente gauche (descendre à gauche sur le sous-arbre droit)
 - direction = montée droite
 - trait 3, (A et direction) = dépiler
 - détruire la pile P

Exemple : calcul de la hauteur d'un arbre

- algorithme récursif
 - hauteur(A)
 - si A est vide retourner -1
 - sinon
 - $hg = \text{Hauteur}(\text{sous-arbreGauche}(A))$
 - $hd = \text{Hauteur}(\text{sous-arbreDroit}(A))$
 - si $hg > hd$
 - retourner $hg + 1$
 - sinon retourner $hd + 1$

Calcul itératif de la hauteur d'un arbre binaire

- A arbre à parcourir, $h = -1$, $\max = -1$,
- créer une pile vide P , direction = descente gauche
- répéter jusqu'à Pile vide et direction = montée droite
 - si A est vide alors
 - direction = montée droite
 - si $h > \max$ alors $\max = h$
 - si (Pile non vide) ou (direction \leftrightarrow montée droite) alors selon direction
 - direction = descente gauche
 - $h++$, empiler (A , desc. dr),
 - $A = ss\text{-}arbGauche}(A)$
 - direction = descente droite
 - $h++$, empiler (A , montée. dr),
 - $A = ss\text{-}arbDroit}(A)$, direction = descente gauche
 - direction = montée droite
 - $h--$, dépiler le nœud A et la direction
- détruire P et retourner \max

Codage en C

- idée géniale : utiliser
 - le TDA PILE du cours 2 et le TDA ARBRE BINAIRE du cours 3
- problème
 - le TDA ARBRE BINAIRE utilise un TDA ELEMENT incarné par des entiers (ou des caractères)
 - le TDA PILE utilise lui aussi un TDA ELEMENT mais incarné par des couples (arbres, Direction)
- le souk !
 - un même identificateur (ELEMENT) désigne deux types différents
- solution
 - en C : bricolo
 - en C++ : les template (polymorphisme paramétrique)
 - en Smalltalk, Java : polymorphisme d'héritage

Simplifications de l'algorithme général

- exemple : Parcours préfixe itératif à la Sedgewick
- A arbre à parcourir
- créer une Pile vide P
- si l'arbre A est non vide empiler A
- tant que P est non vide répéter
 - dépiler dans A
 - traiter (A)
 - si le sous-arbre droit est non vide, l'empiler
 - si le sous-arbre gauche est non vide, l'empiler
- détruire (P)

/* style Sedgewick ne nécessite qu'une pile d'arbre, pas besoin de direction car connaît le type de parcours ici parcours préfixe ; on écrira une direction = 1 pour se dispenser de réécrire un autre TDA élément. version optimisée on n'empile pas les noeuds vides*/

```
void AfficherPrefITER1(ARBIN A){  
    PILE P = PileCreer(200) ;  
    ELTSPCL elt;  
    if (! ArbinVide(A)) {  
  
        elt = EltSpclInit(A, 1) ; /* 1 ou n'importe quoi */  
        PileEmpiler(elt, P);  
        }  
        while( ! (PileVide(P) )) {  
            elt = PileDepiler (P) ;  
            A = EltSpclArbre(elt) ;  
            EltSpclDetruire(elt) ;  
            ElementAfficher(ArbinEtiquette(A));  
            /* empiler d'abord le fils droit */  
            if (! ArbinVide(ArbinDroit(A)) ) {  
                elt = EltSpclInit(ArbinDroit(A), 1) ; /* 1 ou n'importe quoi */  
                PileEmpiler(elt, P);  
                }  
                /* puis empiler le fils gauche */  
                if (! ArbinVide(ArbinGauche(A)) ) {  
                    elt = EltSpclInit(ArbinGauche(A), 1) ; /* 1 ou n'importe quoi */  
                    PileEmpiler(elt, P);  
                    }  
                    }  
                    PileDetruire(P) ;  
    }
```

Parcours en largeur, hiérarchique, par niveau

- stratégie pas du tout récursive
- traitement des nœuds par niveau :
 - traiter les frères avant les fils
- dans la représentation graphique :
 - 1. de gauche à droite
 - 2. de haut vers le bas
- utilisation d'une file
 - on traite un nœud
 - on fait entrer dans la file son fils gauche puis son fils droit

Algorithme

- ParcoursEnLargeur (A)
- créer une file vide F
- si A est non vide alors
 - entrer A dans F
 - tant que F non vide
 - A = sortir(F)
 - traiter (A)
 - si ss-arbGauche(A) non vide l'entrer dans F
 - si ss-arbDroit(A) non vide l'entrer dans F
- détruire F

Cours n° 3 : Les Arbres (2)

- Parcours récursifs et itératifs d'arbres binaires
 - Parcours en profondeur à main gauche
 - Algorithme récursif général
 - Cas particuliers (parcours préfixe, infixe, postfixe)
 - Algorithme itératif général
 - Application : calcul de la hauteur d'un arbre
 - Simplification dans le cas d'un parcours préfixe
 - Parcours en largeur (par niveaux, hiérarchique)
- Parenthèse sur la récursivité

Y'avait longtemps...

- factorielle version récursive
- long factorielle (int n) {
 - if (n <= 0) return 1 ;
 - else return n * factorielle (n - 1) ;
 - }
- factorielle version itérative
- long factorielle (int n) {
 - int r = 1 ;
 - while (n > 1) { r = r * n ; n -- ; }
 - return r ;
 - }

Et sa copine ...fibonacci

- fibonacci version récursive
- long fibonacci (int n) {
 - if (n <= 1) return 1 ;
 - else return fibonacci (n - 1) + fibonacci (n - 2) ;
 - }
- fibonacci version itérative
- long fibonacci (int n) {
 - int u, v ; u = v = 1 ;
 - for (i = 3 ; i <= n ; i++) { u = u + v; v = u - v ; }
 - return u ;
 - }

Récursif versus itératif

- sur ces 2 exemples triviaux la solution itérative est meilleure
- certains compilateurs astucieux remplacent
 - l'appel récursif terminal par une boucle tant que
 - un appel non terminal par un algorithme itératif utilisant une pile
- certains algorithmes sont « naturellement » récursifs
 - 1) algorithmes récursifs de type diviser pour résoudre
 - 2 appels récursifs traitant chacun en gros la moitié des données sans calcul redondant
 - 2) algorithmes récursifs de type combiner pour résoudre
 - commencer par résoudre des cas triviaux
 - puis en les combinant résoudre des cas complexes

Bilan

- un algorithme récursif peut toujours être transformé en algorithme itératif (ne le faire que si on voit que l'algorithme "patine" ou besoin impérieux)
- un programme itératif n'est pas toujours plus efficace qu'un programme récursif
- la récursivité est centrale
 - en informatique dite théorique pour distinguer les problèmes (problèmes calculables) qu'une machine peut résoudre en temps fini
 - en IA et en recherche opérationnelle pour résoudre des problèmes complexes
 - en programmation pour traiter de manière simple de nombreux problèmes sur les arbres binaires (recherche, tri etc.)

Cours n° 3 : Les Arbres (3)

- Les Arbres Généraux
 - Le TDA ARBRE GÉNÉRAL
 - Mises en œuvre des arbres (généraux)
 - par tableau (étiquette père)
 - par tableau et liste chaînée des fils
 - par faux-pointeurs et liste chaînée des fils
 - par pointeurs (nombre maxi de fils)
 - par arbre binaire (Premier Fils, FrèreDroit)
 - Équivalence avec les arbres binaires

Définition et représentation

- Définition récursive (et constructive)
un arbre est la donnée
 - d'une racine r
 - et d'une liste (éventuellement vide) d'arbres disjoints appelés sous-arbres
- $A = < r >$ ou $< r, (A_1, \dots, A_n) >$, A_i arbres disjoints
- Un arbre n'est théoriquement jamais vide : il a au moins une racine
(différence avec les arbres binaires)

Le TDA ARBRE GÉNÉRAL

- utilise le TDA ELEMENT
- primitives de construction
 - dépendent de la représentation
- primitive de destruction
- primitives de consultation
 - retournent l'étiquette, le premier fils, le frèreDroit
- primitives de test
 - est une feuille, est fils dernier né

Mise en œuvre des arbres par tableau

- un arbre est un tableau qui contient l'étiquette d'un nœud et l'étiquette de son père

indice	1	2	3	4	5	6	7	8
étiquette	a	b	c	d	e	f	g	h
père	0	1	1	1	1	2	5	5

- représentation économique en mémoire, utilisée si l'arbre n'est jamais modifié

Mise en œuvre des arbres par liste des fils

- un arbre est la donnée
 - de l'étiquette de sa racine
 - de la liste de ses fils représentée
 - par tableau,
 - par liste chaînée
 - par curseurs

Mise en œuvre des arbres par listes des fils (tableau)

- un arbre est un pointeur sur une structure composée de 2 champs :
 - un tableau de nœuds
 - indice de la racine dans le tableau
- un nœud est une structure à deux champs
 - l'étiquette de la racine
 - la liste de ses fils
- la liste des fils est une liste simplement chaînée sans en-tête des indices dans le tableau des fils
 - la liste des fils est un pointeur sur une cellule à deux champs
 - un indice
 - un pointeur sur la cellule suivante

Mise en œuvre des arbres par liste des fils (pointeurs)

- un arbre est un pointeur sur une structure formée
 - de l'étiquette de la racine
 - et de la liste de ses fils
- une liste d'arbres est un pointeur sur une cellule composée
 - d'un arbre
 - et d'un pointeur sur la cellule suivante.

Mise en œuvre des arbres par pointeurs (nombre maxi de fils)

- quand on connaît le nombre maximum de fils (n)
- un arbre est un pointeur sur une structure à $n + 1$ champs
 - l' étiquette de la racine
 - le premier sous-arbre, le deuxième... le nième

Mise en œuvre des arbres généraux par arbre binaire

- représentation par premier fils et frère droit
- un arbre est représenté par une structure à trois champs
 - l'étiquette de la racine
 - le premier fils de la racine
 - le frère droit de la racine
- tous les arbres n-aires peuvent être représentés par des arbres binaires