Leçon 10 : Arbres : représentations et applications

Pré-requis : Induction, structures C et OCaml. Généralités sur les graphes

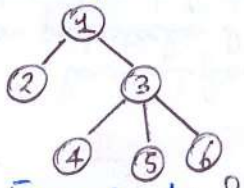
I - Généralités sur les arbres

Un arbre est une structure de données récursives stockant hiérarchiquement des données. Dans un arbre, les données sont appelés des noeuds.

Définition 1 : Un arbre est un couple (u, l) où u est un noeud (élément) et l une liste (potentiellement vide) d'arbre.

- u est appelé la racine de l'arbre (u, l)
- Si l est vide, u est appelé une feuille.
- Les arbres de l sont appelés sous arbres de (u, l)
- Les racines des arbres de l sont appelés enfant de u (et u est leur parent).
- La hauteur de (u, l) est $h(u) = h((u, l)) = 1 + \max_{A \in l} h(A)$ (donc 1 si l est vide).

Exemple 2 : $(1, [(2, [])\ , (3, [(4, []), (5, []), (6, [])])])$



Remarque 3 : Il n'existe pas d'arbres vides.

Exemple 4 : L'organisation des fichiers en repertoire peut être representée sous forme d'arbre.

Remarque 5 : On peut à chaque noeud associer une étiquette. On parle alors d'arbre étiqueté.

Théorème 6 : Identifions les arbres à des graphes non orientés où un sommet (la racine) est choisi, en considérant les liens de parenté comme des arêtes. Les arbres sont alors les graphes connexes acycliques.

Preuve : Exercice.

Conséquence 7 : On représente graphiquement les arbres comme des graphes (cf. exemple précédent).

Définition 8 : Un arbre binaire est définit inductivement par :

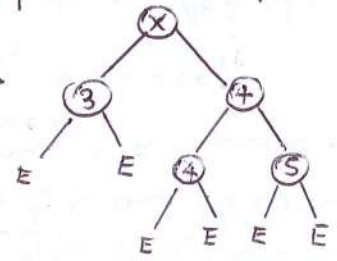
- E est un arbre vide.
- Si A_1 et A_2 sont des arbres binaires et u une donnée, $B(u, A_1, A_2)$ est un arbre binaire, A_1 étant le sous-arbre gauche, et A_2 le droit.

Intuition 9 : Les arbres binaires sont des arbres dont les listes sont de tailles 2, mais avec des arbres vides.

Exemple 10 : Représentation d'une expression arithmétique

$3 \times (4+5)$

↳ $N(\times, N(3, E, E), N(+, N(4, E, E), N(5, E, E)))$ →



Remarque 11 : Pour évaluer l'expression on peut alors faire un parcours postfixe.

II - Representation informatique

II - 1) Représentation comme structure inductive

En représentant un arbre en utilisant sa structure inductive on obtient :

- Pour les arbres générause :

```
type 'a arbre = N of 'a arb_liste and
and type 'a arb_liste = V | Cons of 'a arb * ('a arb_liste) ;;
```

- Pour les arbres binaires :

```
type 'a bin = E | B of 'a * 'a bin * 'a bin ;;
```

Développement 1 : Correspondance entre les arbres binaires et généraux de taille n et utilisation en C.

II - 2) Representation comme un graphe

D'après le théorème 6, on peut representer les arbres comme des graphes.

On peut donc les représenter par :
- listes (ou dictionnaires) d'adjacence
- une matrice d'adjacence
- la liste de toutes les arêtes.

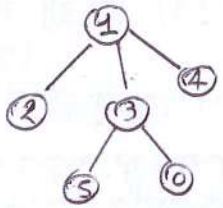
Exemple 12 : En C, par liste d'adjacence cela est pratique, mais on a alors très peu de garanties.

Remarque 13 : Ces représentations sont peu utilisées.

II-3) Représentation par un tableau

Si les identifiants des noeuds sont des entiers de 0 à n-1, on peut stocker l'arbre dans un tableau de taille n, la case i contenant l'identifiant du père du noeud i, -1 si il est la racine.

Exemple 14 :

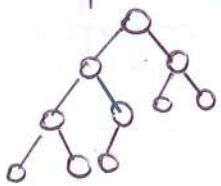


3	-1	1	1	1	3
0	1	2	3	4	5

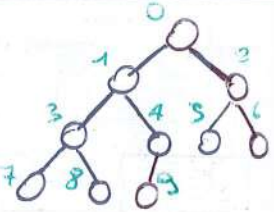
Remarque 15 : On ne stocke ici que la structure. Pour stocker des données, on prendrait un tableau de couples.

II-4) Représentation d'un arbre binaire presque complet

Définition 16 : Un arbre binaire presque complet est un arbre binaire dont tous les étages sont remplies sauf éventuellement le dernier, qui est alors rempli à gauche

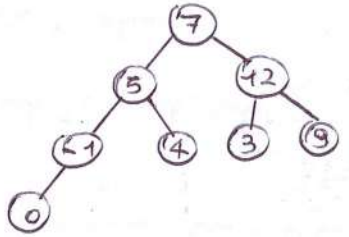


On peut alors représenter un arbre binaire semi complet par un tableau. On numérote alors les sommets par un parcours en largeur préfixe (cf ci contre). Les fils du noeud i sont alors 2i+1 et 2i+2, et son père $\left\lfloor \frac{i-1}{2} \right\rfloor$.



Exemple 17 :

7	5	12	-1	4	3	9	0
0	1	2	3	4	5	6	7



III - Applications

III-1) Dictionnaires

Représentation : Structure inductive.

Définition 18 : Un arbre binaire $B(x, A_1, A_2)$ est un arbre binaire de recherche (ABR) si A_1 et A_2 en sont, et si pour un attribut a, on a $u.a \leqslant x.a \leqslant \min_{u \in A_2} u.a$. (avec E qui est un ABR et $\max_{u \in E} u.a = -\infty$ et $\min_{u \in E} u.a = +\infty$).

Propriété 19 : Chercher et insérer dans un ABR est en O(h).

Propriété 20 : En imposant des contraintes supplémentaires (exemple arbres rouge-noir), on peut forcer $h = O(\log n)$.

Définition 21 : Un dictionnaire (ou tableau associatif) est un tableau dont la valeur des clés ne se limite pas à $[\![0, n-1]\!]$.

On peut alors implémenter un dictionnaire par un ABR, les clés étant les étiquettes des noeuds, à condition qu'on ait un ordre total sur les clés.

Théorème 22 : Une implémentation efficace des dictionnaires par ABR permet des opérations de base en $O(\log(n))$.

III-2) Classes d'équivalence

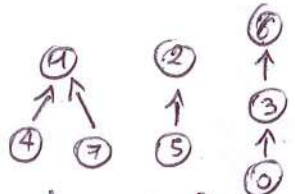
Représentation : tableaux

Définition 23 : Une structure union & trouver est une structure implémentant les classes d'équivalences et permettant de trouver le représentant d'une classe et d'unir deux classes.

On implémente cette structure sous forme d'un ensemble d'arbres (forêt), où chaque arbre représente une classe et où la racine est le représentant de la classe.

Exemple 24 : La relation d'égalité modulo 3 sur $[\![0, 7]\!]$ peut être représenté par

3	-1	-1	6	1	2	-1	1
0	1	2	3	4	5	6	7



Propriété 25 : Si on unifie en faisant pointer la racine de l'arbre le moins profond vers celle de l'arbre le plus profond, on obtient des opérations union et trouver en $O(\log n)$.

III-3) Arbres couvrant de poids minimal

Representation : liste des arêtes

Définition 26 : Dans un graphe pondéré connexe, un arbre couvrant de poids minimal est un sous-ensemble maximal d'arêtes sans cycles, de somme des poids minimal.

Algorithme 27 :

```
Kruskal (L) :  // L est la liste des arêtes
    L ← L trié par poids
    res ← []
    Pour a parcourant L
        Si a ne rajoute pas de cycle dans res
            Ajouter a à res
    Retourner res
```

Propriété 28 : En implémentant la détection de cycle par une structure union & trouver, Kruskal renvoie un arbre couvrant de poids minimal en $O(|L| \times \log |L|)$.

III-4) Files de priorité

Représentation : celles des arbres semi-complet.

Définition 29 : Une file de priorité est une séquence de données dont on peut extraire la donnée d'attribut minimum, et rajouter des données.

On peut implémenter les files de priorité par des tas min.

Définition 30 : Un tas min est un arbre semi-complet où chaque nœud a un attribut plus petit que ses fils.

* Pour ajouter un élément, on le met au bout du tableau, et on l'échange avec son père tant qu'il est plus petit
* Pour extraire le min, on met le dernier élément du tableau au début, et on l'inverse avec le plus petit de ses fils tant qu'il est plus grand.

Développement 2 : Correction de l'insertion dans un tas min et discussion sur l'implémentation.

Propriété 31 : Cette implémentation permet des opérations en $O(\log n)$

Application 32 : Trier par tas en $O(n \log n)$