Table des matières

Première partie

Leçons

Leçon 1

Arbres : représentations et applications

Auteur·e·s: Emile Martinez
Références :

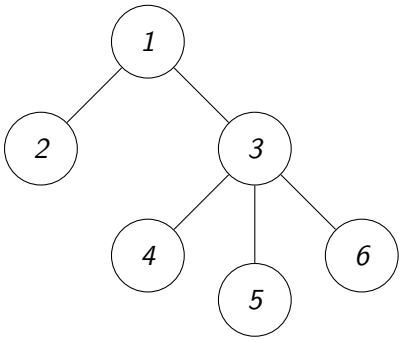
1 Généralités sur les arbres

Un arbre est une structure de données récursives stochant hierarchiquement les données. Dans un arbres, les données sont appelées des noeuds.

Définition 1.1 (arbre) *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 appelées enfant de u (et ont u pour parent)*
- *la hauteur $h(u) = h((u, l)) = 1 + \max_{A \in l} h(A)$ (et donc 1 si l est vide)*

Exemple 1.1 $(1, [(2, []), (3, [(4, [])(5, []), (6, [])])])$ $\longrightarrow$



Remarque 1.1 *Il n'existe ici pas d'abre vide.*

Exemple 1.2 *L'organisation des fichiers en répertoire peut être représenté sous forme d'arbre.*

Remarque 1.2 *On peut à chaque noeud associer une étiquette. On parle alors d'arbre étiqueté.*

Théorème 1.1 *Identifions les arbres à des graphes non orientés où un sommet (la racine) est choisi, en considérant les liens de parentés comme des arêtes. Les arbres sont alors les graphes connexes acycliques*

Démonstration. Exercice □

Corollaire 1.1 *On représente les arbres comme des graphes (cf exemple 1.1)*

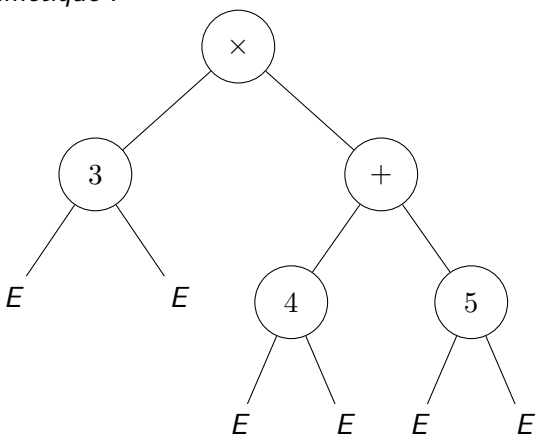
Définition 1.2 *Un arbre binaire est définit inductivement par :*

- *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.*

Idée 1.1 *Les arbres binaires sont des arbres dont les listes sont de taille 2, mais où l'on rajoute des arbres vides.*

Exemple 1.3 *Représentation d'une expression arithmétique :*

$3 \times (4 + 5)$
$\to N(\times, N(3, E, E), A)$
avec $A = N(+, N(4, E, E), N(5, E, E))$ $\to$



2 Représentation informatique

I Représentation comme structure inductive

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

- **Pour les arbres généraux**

```
type 'a arbre = N of 'a arb_liste and
type 'a bin 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 les arbres généraux de taille n et utilisation en C

II Représentation comme un graphe

Les arbres étant des graphes connexes acycliques (cf. théorème 1.1), on peut les représentés comme tels :

— par des listes (ou dictionnaires) d'adjacence

— Par une matrice d'adjacence

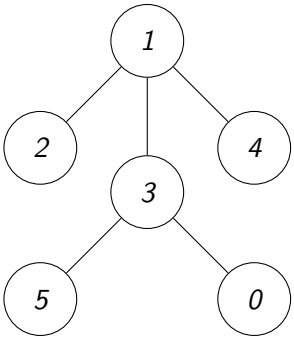
— Comme la liste de toutes les arêtes

Remarque 1.3 *N'exploitant pas la structure d'arbres, ces structures sont peu utilisées.*

III 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 racine.

Exemple 1.4



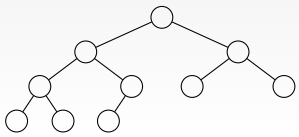
3	-1	1	1	1	3
0	1	2	3	4	5

Remarque 1.4 *On ne stocke ici que la structure. Pour stocker des données, on produit un tableau de couples.*

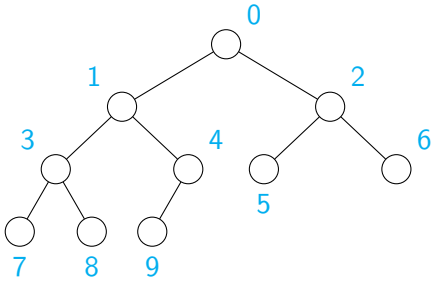
IV Représentation d'un arbre binaire presque complet

Définition 1.3 (Arbre binaire presque complet)

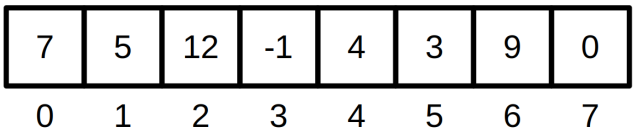
Un arbre binaire presque complet est un arbre binaire dont tous les étages sont remplies sauf éventuellement le dernier qui est alors rempli à gauche

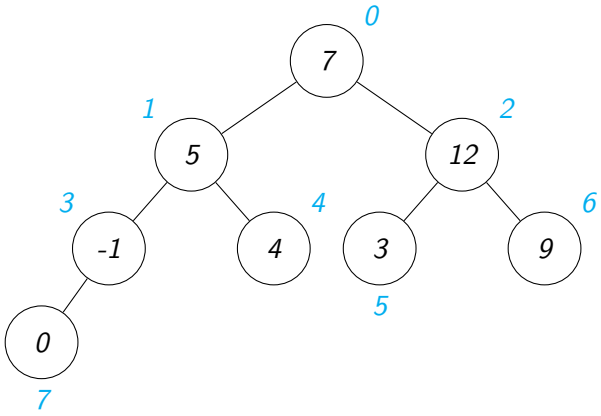


Principe 1.1 *On peut alors représenter un tas par un tableau. On numérote alors les sommets ci contre, donnant l'indice dans le tableau. Les fils du noeuds d'indice i se retrouve aux cases $2 \times i + 1$, $2 \times i + 2$, et son père $\left\lfloor \frac{i-1}{2} \right\rfloor$*



Exemple 1.5





3 Application

Commentaire 1.1 *A chaque fois on va écrire en début quelle représentation on utilise pour notre application. Néanmoins, en classe, on pourrait le laisser proposer par les élèves (une fois l'application présentée). On justifie de ce fait notre organisation, ou les applications arrivent toutes ensembles à la fin. (mais on écrit quand même la représentation au début pour que ce soit plus facile de naviguer à la partie que l'on souhaite).*

I Dictionnaires

Représentation : Structure inductive

Définition 1.4 *Un arbre binaire de recherche $N(x, G, D)$ est un arbre binaire de recherche (ABR) si G et D sont des ABR et si, selon un attribut a, $\max_{u \in G} u.a \leq x.a \leq \min_{v \in D} v.a$ (avec E qui est un ABR et $\max_{u \in E} u.a = -\infty$ et $\min_{u \in E} u.a = +\infty$)*

Proposition 1.1 *Chercher et insérer dans un ABR est en $O(h)$*

Proposition 1.2 *En imposant des contraintes supplémentaires (exemple arbres rouge-noir), on peut forcer $h = O(\log n)$*

Définition 1.5 *Un dictionnaire (ou tableau associatif) est un tableau où les indices (clés) ne se limite pas à $[\![0, n]\!]$*

Idée 1.2 *On peut alors implémenter un dictionnaire par un ABR, les clés étant les étiquettes des nœuds, à condition d'avoir un ordre total sur les clés*

Théorème 1.2 *Une implémentation efficace des dictionnaires par ABR permet des opérations de base en $O(\log n)$*

Commentaire 1.2 *Si on ne veut pas faire le développement 2, et qu'on préfère celui sur les arbres k-dimensionnel, on peut aussi parler aussi du pb des k-plus proches voisins et parler des ABR comme des arbres k-dim pour résoudre le pb*

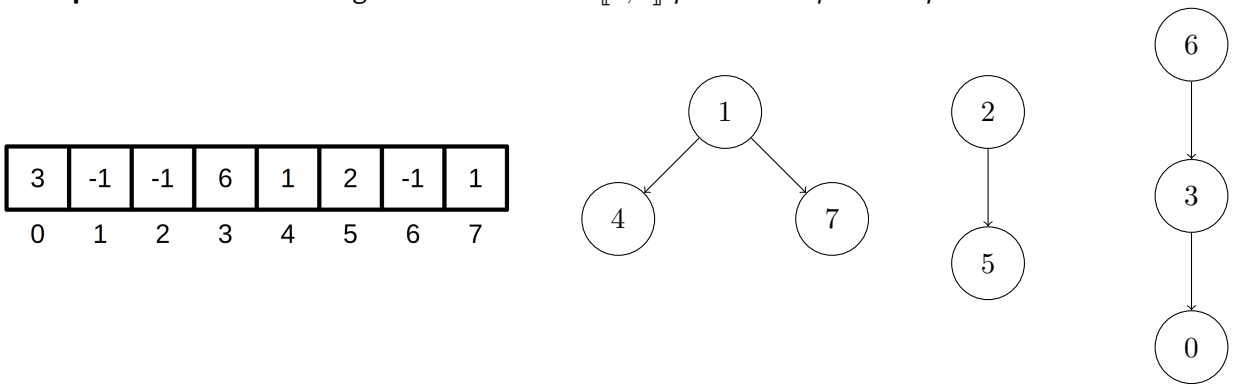
II Classes d'équivalence

Représentation : Tableaux

Définition 1.6 *Une structure union & trouver est une structure implémentant les classes d'équivalence, permettant de trouver les représentants d'une classe et d'unir deux classes.*

Idée 1.3 *On implémente cette structure sous formes d'un ensemble d'arbres (forêt), où chaque arbre représente une classe et où la racine est le représentant de la classe*

Exemple 1.6 *La relation d'égalité modulo 3 sur $[\![0,7]\!]$ peut être représenté par :*



Proposition 1.3 *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 unir et trouver en $O(\log(n))$*

III Arbres couvrant de poids minimal

Représentation : liste des arêtes

Définition 1.7 *Dans un graphe pondéré connexe, un arbre couvrant de poids minimal, est un sous-ensemble maximal d'arêtes sans cycles de somme de poids minimal.*

Algorithme 1.1 : $Kruskal(L)$

Entrées : L liste des arêtes
Sorties : Liste des arêtes d'un ACM
$L \leftarrow L$ trié
$res \leftarrow []$
pour a parcourant L **faire**
  si a ne rajoute pas de cycle dans res **alors**
    Ajouter a à res
retourner res

Proposition 1.4 *En implémentant la détection de cycle par une structure union & trouver, Kruskal renvoie un arbre couvrant de poids minimal en* $O(|L| \times \log |L|)$

IV Files de priorité

Représentation : celles des arbres semi-complet

Définition 1.8 *Une file de priorité est une séquence de données dont on peut extraire la donnée d'attribut minimum, et rajouter des données.*

Idée 1.4 *On peut implémenter les files de priorité par des tas min*

Définition 1.9 *Un tas min est un arbre semi-complet où chaque noeud a un attribut plus petit que ses fils*

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

Principe 1.2

- ⋆ *Pour ajouter un élément, on le met au bout du tablea 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.*

Proposition 1.5 *Cette implémentation permet des opérations en* $O(\log n)$

Application 1.1 *Trier par tas en* $O(n \log n)$