Arbres, Représentation et applications [TOR]

Ce plan prend parti de ne pas de suivre l'ordre chronologique qu'aurait un cours, mais de mettre en avant le lien entre les arbres, leurs applications et les autres parties du programme.

I. Arbres binaires

A. Définitions

Définition 1 **Les arbres binaires** sont définis inductivement comme:

- Soit l'arbre vide, qu'on notera V
- Soit un nœud $N(e, g, d)$ où e est l'étiquette du nœud, g (resp. d) est le sous-arbre *gauche* (resp. *droit*). On dit que g et d sont les **enfants** de $N(e, g, d)$, et que $N(e, g, d)$ est le **parent** de g et d.

Définition 2 Un nœud qui n'a pas de parent est appelé **racine**.

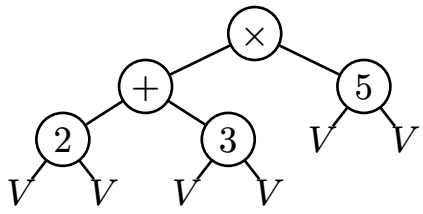
Représentation 3 **Arbres binaires en OCaml, C et Python**

```
type 'a arbre =
| Vide
| Noeud of 'a
         * 'a
arbre
         * 'a arbre
```

```
struct arbre {
  int val;
  struct arbre*
droit;
  struct arbre*
gauche;
}
```

```
class Arbre:
  def __init__(self, val, gauche, droit):
    self.val = val
    self.droit = droit
    self.gauche = gauche
```

Exemple 4 **Les expression arithmétiques** peuvent être représentées sous la forme d'un arbre binaire dans lequel les opérandes sont les sous-arbres de l'opération à effectuer.



L'étiquette de la racine de cet arbre est $\times$.

Définition 5 **Une feuille** est un arbre dont les deux sous-arbres sont vides. Dans l'exemple ci-dessus, les feuilles sont 2, 3 et 5.

Définition 6 **Un nœud interne** est un nœud qui n'est pas une feuille.

Définition 7 À chaque nœud d'un arbre on associe une **profondeur**.La profondeur de la racine est 0. Dans un nœud $N(e, g, d)$, la profondeur des racines de g et de d est 1 plus la profondeur de $N(e, g, d)$.

Définition 8 **Hauteur** La hauteur d'un arbre est le maximum de la profondeur de ses feuilles.

Définition 9 **Le poids** Si on a une fonction poids P pour les étiquettes d'un arbre, on peut l'étendre à l'arbre par :

- $P(N(e, g, d)) = P(e)$ si $N(e, g, d)$ est une feuille
- $P(N(e, g, d)) = P(e) + P(g) + P(d)$ sinon

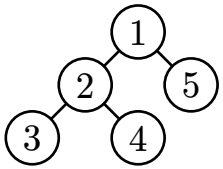
DEV

Application 10 **Algorithme de Huffman.** On peut utiliser des propriétés sur le poids des arbres binaires pour compresser un message en utilisant l'algorithme de Huffman.

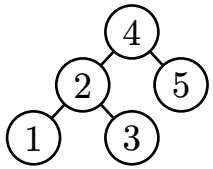
B. Parcours et ordres sur les arbres

Définition 11 **Le parcours en profondeur** suit la définition inductive des arbres: pour chaque nœud, on parcours ses sous-arbres un-à-un jusqu'à atteindre une feuille.

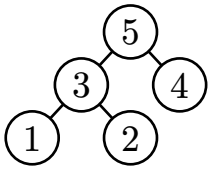
Ordre préfixe



Ordre infixe

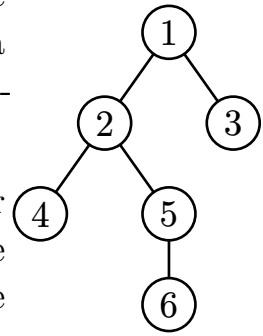


Ordre postfixe



On fera le lien avec la structure de **pile** (premier entré, dernier sorti).

Définition 12 **Parcours en largeur** Ce parcours traite les nœuds de l'arbre par hauteur croissante, de la gauche vers la droite. On fera le lien avec la structure de **file** (premier entré, premier sorti).



Définition 13 **Ordre induit** Numéroter les nœuds par leur ordre de visite pour un parcours permet de les munir d'un ordre. Par exemple, l'ordre postfixe permet d'ordonnancer des tâches.

II. Formes spécifiques d'arbres binaires

A. Arbre binaire complet

Définition 14 Un arbre binaire est **parfait** si chacun de ses nœuds internes a pour sous-arbres soit deux nœuds internes, soit deux feuilles. Dit autrement, tous ses étages sont entièrement remplis.

Définition 15 Un arbre est **complet** si un parcours en largeur ne passe pas par un nœud interne après être passé par une feuille. Dit autrement, tous les étages sont complets sauf éventuellement le dernier; et dans celui-ci les feuilles sont le plus à gauche possible.

Propriété 16 Pour tout arbre binaire de hauteur h ayant n nœuds,

$$h + 1 \leq n \leq 2^{h+1} - 1$$

Représentation 17 **Les arbres binaires complets peuvent être représentés en tableaux** , on retrouve alors le sous-arbre gauche (resp. droit) d'une racine i à l'indice $i \times 2$ (resp. $i \times 2 + 1$).

Exemple 18 L'expression arithmétique $(2 + 3) \times 5$ est représentée comme arbre binaire par le tableau $[\times, +, 5, 2, 3]$.

B. Tas

Définition 19 **Un tas min** (resp. max) est un arbre binaire tel que pour tout nœud $N(e, g, d)$ et tout $e' \in g \cup d$, on aie $e' \geq e$ (resp. $e \leq e'$).

Autrement dit, l'élément de plus petite (resp. plus grande) étiquette de chaque sous-arbre est à la racine du sous-arbre.

Application 20 **Le tri par tas** est un tri utilisant une structure de tas, et ayant une complexité en $O(n \log n)$, ce qui est optimal pour un tri par comparaison. On commence par construire un tas des éléments à trier, puis on retire successivement la racine. DEV

Application 21 **File de priorité** Le type abstrait de file de priorité a plusieurs implémentations utilisant des tas. Par exemple les files gauches ou les tas de Fibonacci.

C. Arbre binaire de recherche

Définition 22 Un **arbre binaire de recherche** (ou ABR) est un arbre binaire tel que pour chaque nœud $N(e, g, d)$, si $e' \in g$ alors $e' < e$ et si $e' \in d$ alors $e < e'$.

Proposition 23 **ABRs et complexité.** Les opérations de recherche et d'insertion d'un élément dans un ABR peuvent être faites en O(hauteur de l'arbre).

Idée 24 **ABRs équilibrés** Si on équilibre la longueur des différentes branches d'un ABR, cela améliore la complexité.

Définition 25 **Les arbres bicolores** sont des ABRs où on associe à chaque nœud l'une de deux couleurs (par exemple « rouge » et « noir ») et on préserve les invariants de structure suivants:

- Le parent d'un nœud rouge n'est jamais un nœud rouge
- Tout chemin de la racine à une feuille a le même nombre de nœuds noirs.

Propriété 26 **La hauteur d'un arbre bicolore** ainsi que la complexité d'y rechercher et insérer un élément sont en O(log(Nombre d'éléments)).

III. Arbres n-aires et arbres d'arité arbitraire

A. Généralisation des arbres binaires

Remarque 27 **Arbres n-aires** On peut généraliser la notion d'arbres binaires à des arbres n-aires, où chaque nœud a jusqu'à n enfants.

Remarque 28 Les listes sont des arbres unaires.

Application 29 **Bases de données** Certains Systèmes de Gestion de Bases de Données utilisent des **B-arbres**, qui peuvent être vus comme une **généralisation des arbres binaires de recherche** à une arité supérieure.

Définition 30 On définit les **arbres d'arité arbitraire** inductivement comme étant un nœud $N(e, s)$ où e est l'étiquette de l'arbre et s est une liste de sous-arbres. Dans la suite, on écrira juste **arbre** pour **arbre d'arité arbitraire**.

Définition 31 Un ensemble de plusieurs arbres est une **forêt**.

Représentation 32 **Arbres en OCaml, C et Python**

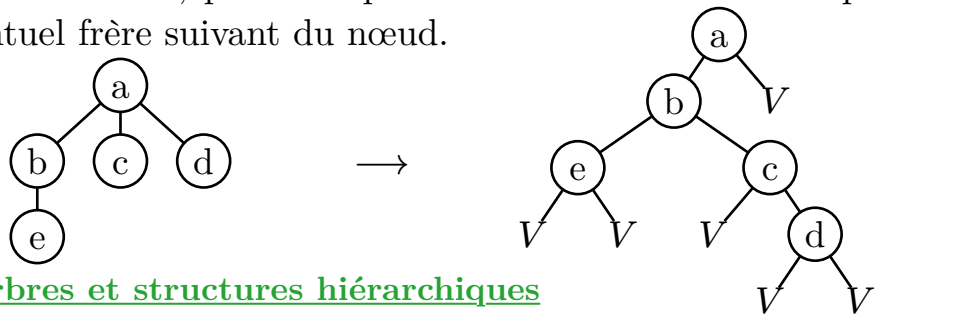
```
type 'a arbre =
| Vide
| Noeud of 'a
        * 'a arbre
list
```

```
struct arbre {
  int valeur;
  struct arbre* sous_arbres;
  int    nb_sous_arbres;
}
```

```
class Arbre:
  def __init__(self, valeur):
    self.valeur = valeur
    self.sous_arbres = []
```

Remarque 33 **Les arbres d'arité arbitraire** sont une généralisation des arbres binaires. Les définitions données pour les arbres binaires (feuille, profondeur, hauteur, etc…) peuvent être transposées aux arbres aisément.

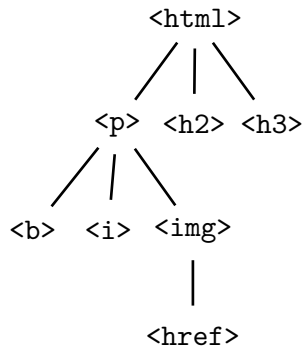
Représentation 34 **Un arbre peut être représenté comme un arbre binaire** dans lequel le sous-arbre gauche correspond au premier enfant du nœud, pendant que le sous-arbre droit correspond à l'éventuel frère suivant du nœud.



B. Arbres et structures hiérarchiques

Remarque 35 Les arbres sont souvent utilisés pour modéliser des structures hiérarchiques.

Exemple 36 **Page HTML** Le format HTML représente un document sous forme d'arbre n-aire.



Application 37 **Un arbre de syntaxe** est une représentation sous forme d'arbre d'un terme (programme, formule booléenne, etc…).

Application 38 La hiérarchie d'un **système de fichiers** peut être représentée comme un arbre où les répertoires sont des nœuds internes tandis que les fichiers sont des feuilles.

Application 39 **Arbres d'inférence** Quand on définit une **relation** de manière **inductive**, cela donne lieu à des **arbres de dérivation**. En particulier, quand la relation est une déduction on parle d'**arbre d'inférence** ou d'**arbre de preuve**.

C. Arbres et algorithmique

DEV

Application 40 **Unir-trouver** La structure « unir-trouver », qui permet de représenter des classes d'équivalence, est implémentée efficacement via des arbres.

Remarque 41 Unir-trouver est utilisé dans la recherche d'arbre couvrant minimal par l'algorithme de Kruskal.

Représentation 42 **Implicite** Un arbre peut être représenté implicitement par des appels de fonctions le parcourant.
C'est le cas pour les méthodes par Séparation-Évaluation (*Branch and bound*) et retour sur trace (*bactracking*).
C'est également le cas pour l'algorithme *min-max*, qui parcoure implicitement l'arbre des configurations d'un jeu.

Remarque 43 Inversement, les appels de fonctions peuvent être vu comme des abres. En particulier en cas d'activations multiples (méthode *diviser pour régner*, par exemple).

Remarque 44 ***Hash-consing*** Quand on représente un arbre, on peut économiser de l'espace mémoire en ne représentant qu'une seule fois chaque sous-arbre, avec une méthode appelée *hash consing*.

D. Arbres et graphes

Remarque 45 Les arbres sont des graphes connexes non-orientés acycliques enracinés.

Remarque 46 **Arbre couvrant minimal** Inversement, on peut vouloir extraire un arbre couvrant minimal d'un graphe. C'est utile, par exemple, pour l'algorithme de 2-approximation du problème du voyageur de commerce dans le cas euclidien.

Arbres, Représentation et applications [TOR]

I. Arbres binaires

A. Définitions

1 Def Les arbres binaires

2 Def

3 Représentation Arbres binaires en OCaml, C et Python

4 Ex Les expression arithmétiques

5 Def Une feuille

6 Def Un nœud interne

7 Def Profondeur

8 Def Hauteur

9 Def Le poids

10 App Algorithme de Huffman.

B. Parcours et ordres sur les arbres

11 Def Le parcours en profondeur

12 Def Parcours en largeur

13 Def Ordre induit

II. Formes spécifiques d'arbres binaires

A. Arbre binaire complet

14 Def Arbre binaire parfait

15 Def Arbre complet à gauche

16 Prop Inégalités entre hauteur et nombre de nœuds.

17 Représentation Les arbres binaires complets peuvent être représentés en tableaux

18 Ex Expression arithmétique comme tableau

B. Tas

19 Def Un tas min

20 App Le tri par tas

21 App File de priorité

C. Arbre binaire de recherche

22 Def Arbres binaires de recherche (ABR)

23 Prop ABRs et complexité.

24 Idée ABRs équilibrés

25 Def Les arbres bicolores

26 Prop La hauteur d'un arbre bicolore

III. Arbres n-aires et arbres d'arité arbitraire

A. Généralisation des arbres binaires

27 Rem Arbres n-aires

28 Rem

29 App Bases de données

30 Def Arbres d'arité arbitraire

31 Def Forêt

32 Représentation Arbres en OCaml, C et Python

33 Rem Les arbres d'arité arbitraire

34 Représentation Un arbre peut être représenté comme un arbre binaire

B. Arbres et structures hiérarchiques

35 Rem Hiérarchie

36 Ex Page HTML

37 App Un arbre de syntaxe

38 App Système de fichier

39 App Arbres d'inférence

C. Arbres et algorithmique

40 App Unir-trouver

41 Rem

42 Représentation Implicite

43 Rem

44 Rem *Hash-consing*

D. Arbres et graphes

45 Rem

46 Rem Arbre couvrant minimal

Notes

- La difficulté est de rendre le plan « vivant », on ne veut pas juste un catalogue de définitions ni d'applications.

Éventuels autres liens avec le programme n'apparaissant pas dans le plan actuel

- Arbres k-dimensionnels (Prépa 14A)
- En POO la hiérarchie des classes est un arbre (hors héritage multiple)
- Les circuits combinatoires (COMP 1A) peuvent être vus comme des arbres, c'est possiblement un exemple plus concret.

Autres devs possibles

- Tri par tas est un bon dev, et il fait gagner de la place dans le plan
- Huffman
- Arbres bicolores
- ABRs optimaux
- Splay trees
- ID3
- Unir-trouver

Bibliographie

[TOR] T. Balabonski & S. Conchon & J. Filliâtre & K. Nguyen & L. Sartre, *MP2I MPI, Informatique Cours et exercices corrigés.*

Aloïs Rautureau & Santiago Sara Bautista