

# ARBRES : REPRESENTATIONS ET APPLICATIONS

## I) Graphes arborescents

### 1) Rappels sur la connectivité dans les graphes

Déf: Graphes orientés  $\vec{G} = (V, \vec{E})$  avec  $\vec{E} \subset V \times V$   
Graphes non-orientés  $G = (V, E)$  avec  $E \subset \binom{V}{2}$  les paires de  $V$ .  
↳ on ne distingue pas les extrémités. On peut l'orienter.

Déf: Chemin:  $[(v_0, v_1); (v_1, v_2); \dots; (v_{k-1}, v_k)]$  tel que  $\forall i, (v_i, v_{i+1}) \in \vec{E}$   
↳ source:  $v_0$ , but:  $v_k$ , longueur:  $k$

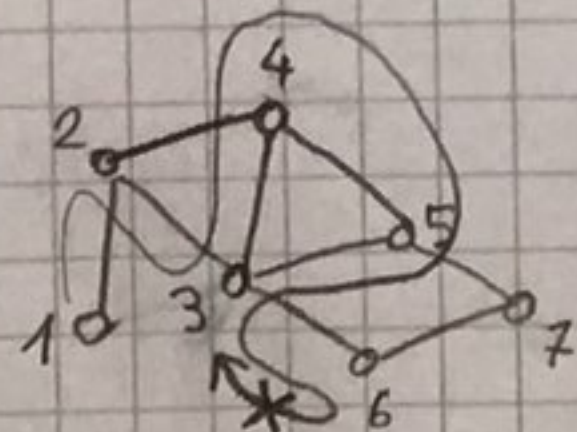
Déf:  $G = (V, E)$  est connexe lorsque pour toute paire de sommets il y a un chemin dont un sommet est la source et l'autre le but.

Déf:  $H = (V_H, E_H)$  est un sous-graphe de  $(V, E)$  lorsque  $V_H \subset V, E_H \subset E$ .

### 2) Cycles, arbres

Déf: un chemin est simple lorsque les arêtes sont prises au plus une fois

Déf: un chemin simple est un cycle lorsque son but est aussi sa source



$[12; 23; 34; 45; 53; 36]$  est simple

$[12; 23; 34; 45; 53; 36; 63]$  non

$[23; 35; 57; 76; 63; 34; 42]$  est un cycle

$[12; 21]$  n'est pas un cycle (car non simple)

Déf: Un graphe est acyclique s'il n'a pas de cycle. Si non-orienté, c'est une forêt.

Une forêt connexe est un arbre. Une feuille est un sommet de degré 1.

Propriété: Dans une forêt, il existe au plus un chemin simple entre chaque paire de sommets. (exactement un, pour les arbres)

Propriété: Si  $|E| \geq 1$ , une forêt a au moins deux feuilles.

Une forêt a au plus  $|V| - 1$  arêtes, un arbre exactement.

### 3) Arbres couvrants, enracinement, combinatoire

Déf: Un arbre couvrant d'un graphe connexe est un sous-graphe de ce graphe qui a les mêmes sommets et est un arbre.



Arbre couvrant du réseau SNCF

Déf: Une évaluation d'arêtes est une fonction  $f: E \rightarrow \mathbb{R}$  qui représente un coût ou un profit.

DEVT 1: Calcul d'un arbre couvrant minimum par Prim

Nos arbres étaient non-orientés, on peut les orienter en choisissant une racine:

Déf: Un enracinement est une orientation d'un arbre tel que chaque sommet sauf un (appelé racine) est au but d'exactly un arc.

Propriété: Une fois la racine choisie, c'est uniquement déterminé.

Théorème: Le graphe complet  $K_n$  a  $n^{n-2}$  arbres couvrants distincts.

Démonstration: Par double comptage des constructions d'arbres orientés.

Exercice: Prouver qu'on trouve un diamètre d'un arbre en 2 parcours en largeur



## II) Arbres hiérarchiquement enracinés

### 1) Definition inductive

Def: Un arbre défini inductivement est un nœud stockant une donnée associée à une collection de sous-arbres

```
type tree =  
  Node of (int) * (tree list)
```

```
struct tree {
    int valeur, nb_fils;
    struct tree* tableau_fils
};
```

→ La collection peut être ordonnée ou non.

(tableau, liste, ensemble, ...). On les appelle ses fils, le nœud est le père de ses fils.

Lien avec les graphes: En associant un sommet à chaque nœud, et un arc de chaque nœud vers ses fils, on obtient un arbre enraciné.

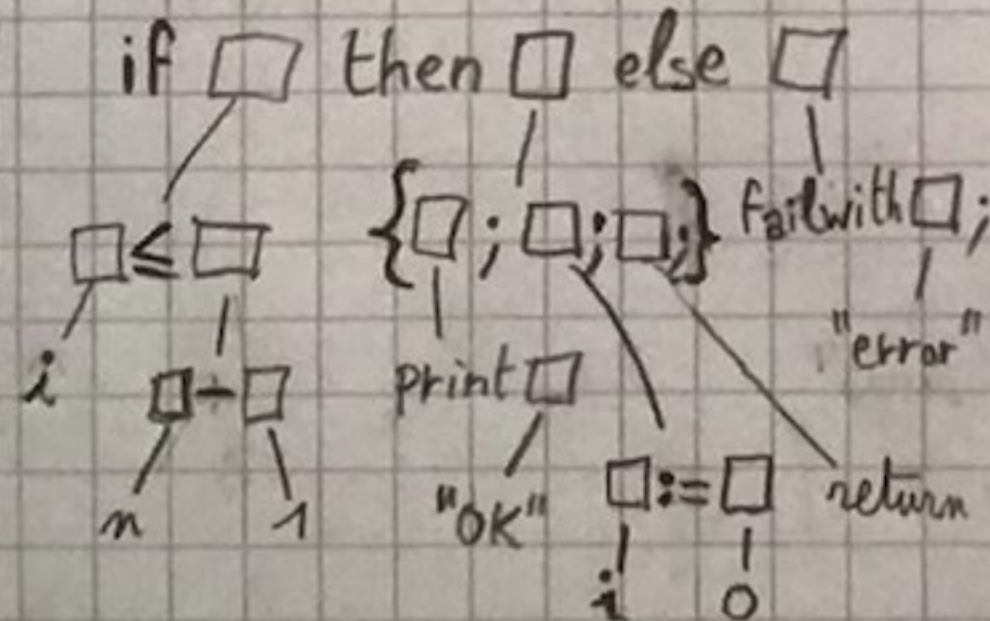
Déf: La profondeur d'un nœud est sa distance à la racine.

La hauteur d'un arbre est le max des profondeurs de ses nœuds.

Remarque (admis): Pour la plupart des modèles,  $E(\text{hauteur}) = \Theta(\sqrt{n \ln n})$

2) Exemple: arbre de syntaxe

Les noeuds peuvent avoir  
une sémantique. On détaille  
sur l'exemple d'un AST :



Ces arbres représentent la logique

d'un code, mais le code est une grande chaîne de caractères! Pareil en XML.

Pour comprendre la correspondance, on met des parenthèses autour de chaque mot.

Déf: Un mot de Dyck, ou expression bien parenthésée, est une chaîne de

(...) caractères '(' et ')' dont chaque préfixe a plus de '(' que de ')', et avec autant dans la chaîne entière.

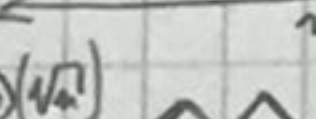
Propriétés: - Les mots de Dyck de taille  $2n$  correspondent aux structures des arbres à  $n$  nœuds, via la construction montrée, qui est bijective.

- il y a  $C_n = \frac{1}{n+1} \binom{2n}{n}$  mots de Dyck à  $2n+2$  lettres (Catalan)

Remarque: la profondeur au cours du mot est

une marche aléatoire sur  $\mathbb{N}$ , ce qui correspond bien

à la remarque précédente on  $E(\text{hauteur}) = O(\sqrt{n})$



### 3) Structure de tas, arbre implicite

Déf: Une valuation des nœuds vérifie la propriété de tas lorsque la valeur sur un père est plus grande que sur chacun de ses fils (donc sur sa descendance)

Exemple: instant d'évaluation pour l'AST d'une expression

On présente l'interface de file à priorité.

```
type bTree = Empty
```

On commence par l'implémentation de la lib Xaml:

| BNode of int \* bTree \* bTree

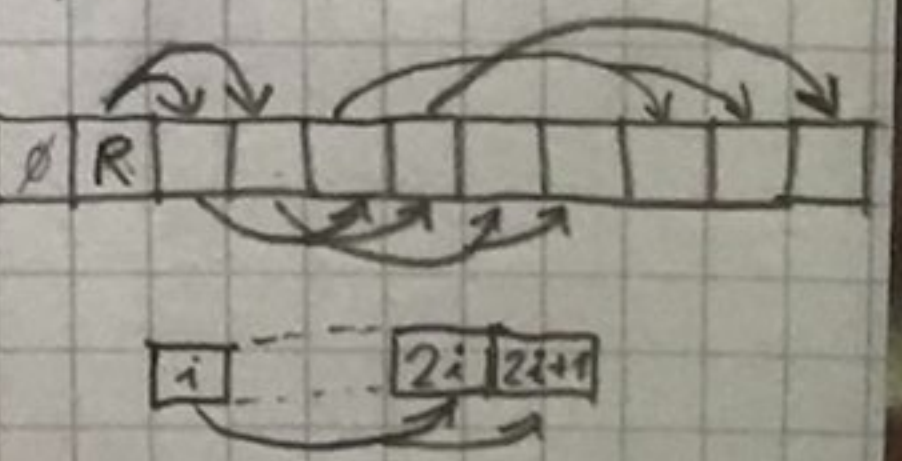
Puis on présente le tas binaire complet en C:

C'est l'occasion de montrer que la

(ici cachée dans les indices dans un tableau)

Remarque: Ces deux structures ont des collections de fils de taille au plus 2 :

ce sont des arbres binaires. Pour Prim par ex., on a d'autres tas.





### III) Arbres binaires

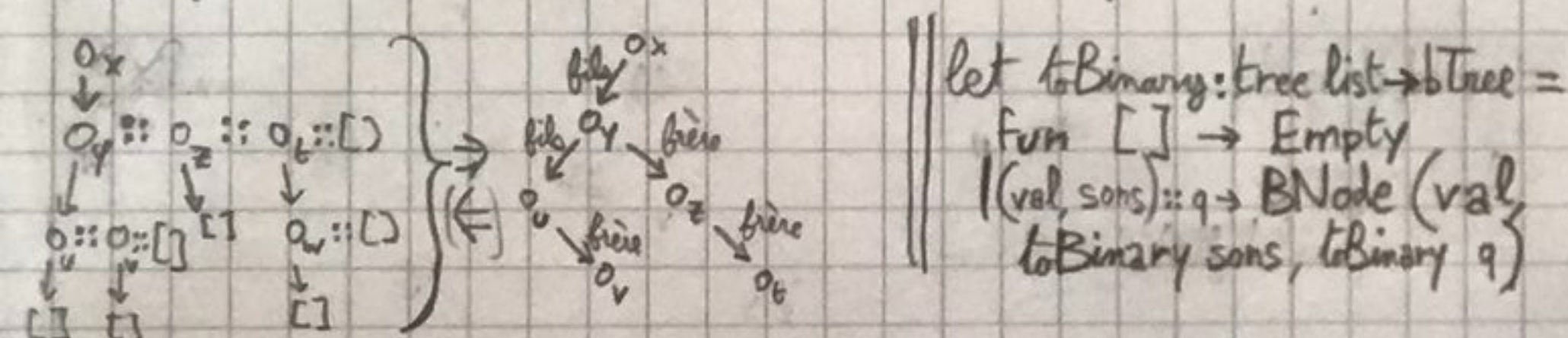
#### 1) Définitions, équivalence fils gauche-frère droit

Def: Dans un arbre binaire la collection des fils sera exactement deux fils (un "gauche" et un "droit") qui sont soit des feuilles (sans fils, "vides") soit des arbres binaires.

Propriété: - Le nombre  $B_n$  d'arbres binaires à  $n$  nœuds vérifie  $B_{n+1} = \sum_{k=0}^n B_k B_{n-k}$

- Les nombres de Catalan  $C_n = \frac{1}{n+1} \binom{2n}{n}$  vérifient ça, soit  $B_n = C_n$

Remarque: C'est les mêmes que pour les arbres enracinés généraux vu plus tôt!



Théorème: les arbres binaires (resp. sans fils droit) sont en bijection avec les listes d'arbres (resp. les arbres) planaires enracinés.

#### 2) Arbres binaires de recherche

Def: Une valuation des nœuds vérifie la propriété de recherche lorsque la valeur d'un nœud est plus grande que celle des nœuds dans son sous-arbre gauche, plus petite que celle des nœuds dans son sous-arbre droit.

Ça permet de chercher facilement un nœud en descendant dans l'arbre.

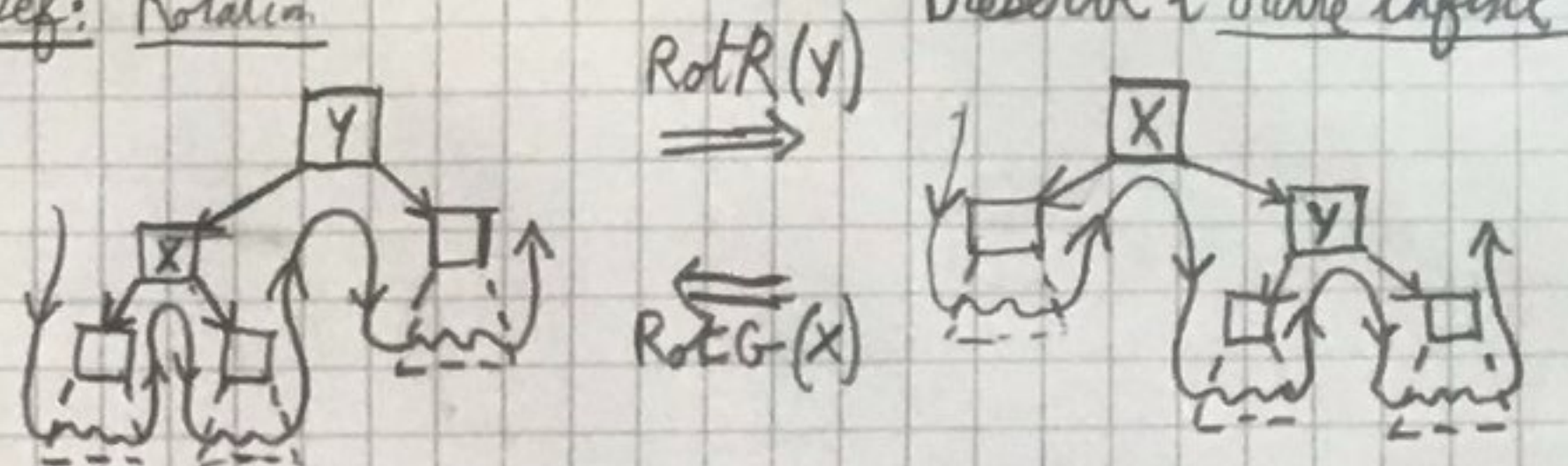
Remarque: Pour une valuation donnée, plein de structures possibles.

On donne le pseudo-code de la recherche, insertion et suppressions naïves.

Remarque: Ces opérations sont en  $O(\text{profondeur})$ , qui peut être  $O(\text{taille})$ .

### 3) Equilibre, équilibrage

Def: Rotation

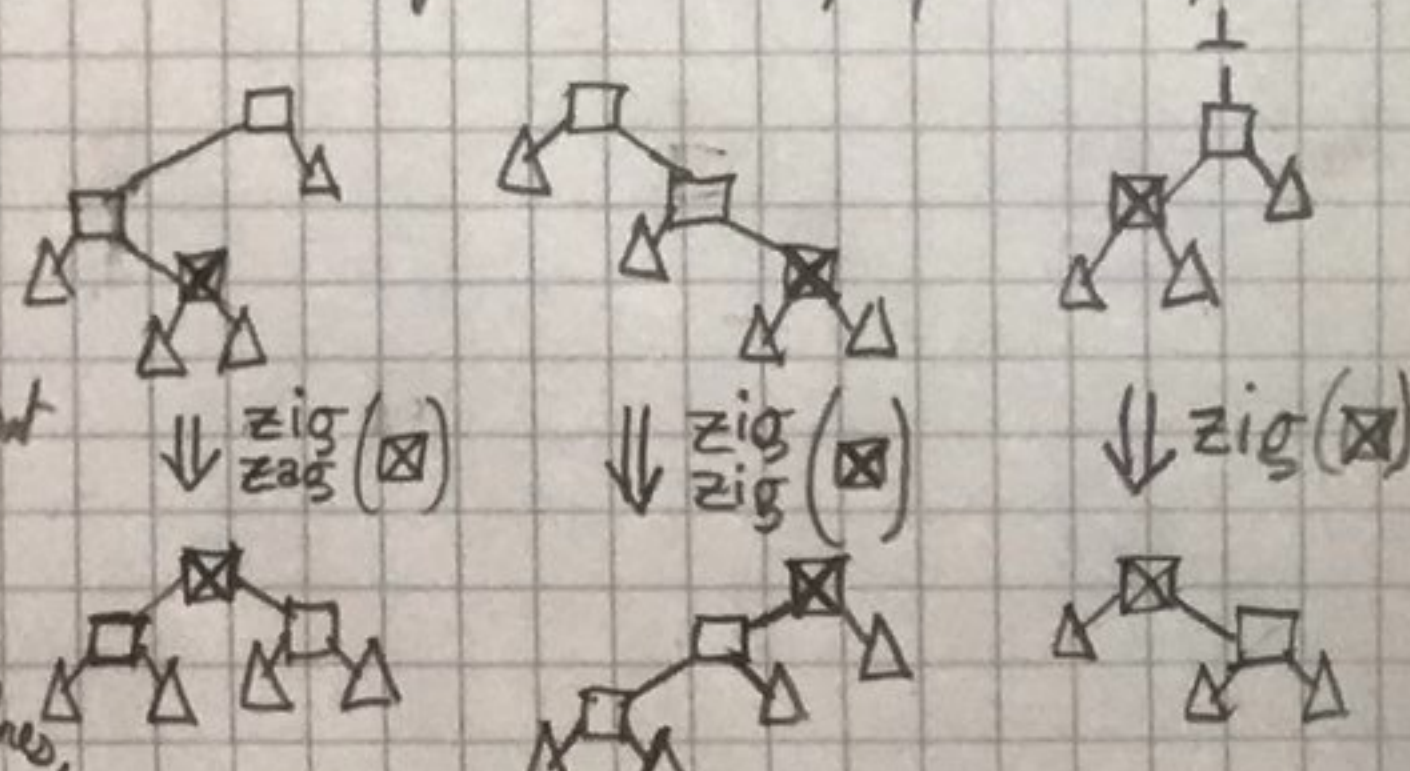


Les rotations permettent, après des opérations naïves de garantir une profondeur en  $O(\log(\text{taille}))$

$\hookrightarrow$  modèles d'arbres AVL et rouge-noir. Complicé à implémenter.

Def: Arbre splay:

à chaque accès, faire remonter l'élément à la racine, en rapprochant ses sous-arbres.



Propriété: Avec le potentiel  $\Phi(T) = \sum_{v \in T} \log_2 [\text{Card}(\text{sousArbre}(v, T))]$  la complexité amortie est  $O(\log(\text{taille}))$ .

Remarque: Pour les ABR on aura pas une distribution uniforme sur les arbres.

Pour les arbres généraux, on a une profondeur en  $O(\sqrt{n})$ . Comment accélérer?

**DEV T 2:** Link-cut tree