# Rapport projet TER - Génération d'arbres

Bintou Fofana Bin Liu

Master Informatique Paris Saclay UVSQ 2016-2017

# Table des matières

# 1 Introdution

**La structure d'arbre est très utilisée dans le domaine de l'informatique.Les mathématiciens voient les arbres comme des cas particuliers de graphes non orientés connexes et acycliques, donc contenant des sommets et des arcs.**

**Certains arbres particuliers nommés arbres binaires sont les plus utilisés en informatique et les plus simples à étudier. En outre il est toujours possible de "binariser" un arbre non binaire.**

**En ce qui concerne notre projet,nous nous intéressons à la génération d'arbres enracinés et non étiquetés .Le projet a pour but de réaliser un programme informatique (en langage C) qui génère pour un nombre N de sommets donnés tous les arbres concernés.Un autre but du projet serait de nous initier ,étudiants que nous sommes à la recherche.**

# 2 Généralités sur les arbres

1. Un **graphe orienté** est défini par le couple $(N, A)$ :
   $N$ : ensemble des noeuds ou sommets du graphe $A$ : ensemble des arêtes du graphe. L'ensemble des arêtes est un sous-ensemble du produit cartésien N×N. Une arête est une ligne joignant un noeud de départ d à un noeud d'arrivée. L'arête da si elle existe est différente de l'arête ad.

2. **Un graphe est non orienté** , si on ne distingue pas le sommet de départ et le sommet d'arrivée.

3. La relation**successeur** est une application de N dans l'ensemble des parties de N, qui à tout n appartenant à N associe l'ensembles des a appartenant à N tels que na appartient à A.

4. La relation **prédécesseur** est une application de N dans l'ensemble des parties de N, qui à tout n appartenant à N associe l'ensembles des a appartenant à N tels que da appartient à A.

5. **Une chaîne** est une séquence d'arêtes $\{a_1, ...a_k\}$ telle que pour tout i de 1 á k-1 les arêtes ai et $a_{i+1}$ ont une extrémité commune.

6. **Un cycle** est une chaîne telle que $a_1$ et $a_k$ ont une extrémité commune.

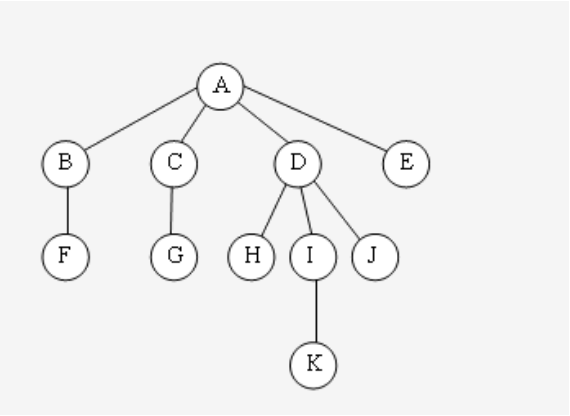
7. **Un chemin** est une séquence d'arêtes $\{a_1, ...a_k\}$ telle que pour tout i de 1 à k-1 les arêtes $a_i$ et $a_{i+1}$ ont une extrémité commune, cette extrémité étant de type arrivée pour $a_i$ et départ pour $a_{i+1}$.

8. **Un circuit** est un chemin tel que $a_1$ et $a_k$ ont une extrémité commune, cette extrémité étant de type arrivée pour $a_k$ et départ pour $a_1$.
   La longueur du chemin, chaîne, circuit ou cycle est le nombre d'arêtes formant ce chemin, chaîne, circuit ou cycle.

9. Un graphe est **connexe** si et seulement si $\forall(n_1, n_2) \ni N \times \quad N, \exists$ une chaîne de $n_1$ á $n_2$.

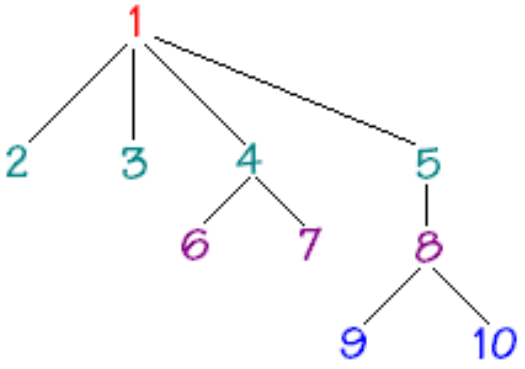
10. **Un arbre** est un graphe non orienté, acyclique et connexe. Sa forme évoque en effet la ramification des branches d'un arbre. Un arbre est aussi un **graphe planaire** c'est-à- dire un graphe qu'on peut dessiner dans le plan sans que ses arêtes ne se touchent, sauf à leurs extrémités.

    Il peut être représenté avec l'origine de l'arête racine en bas ou en haut (en informatique, la racine est souvent figurée en haut). La **racine** d'un arbre est l'unique noeud ne possédant pas de parent.

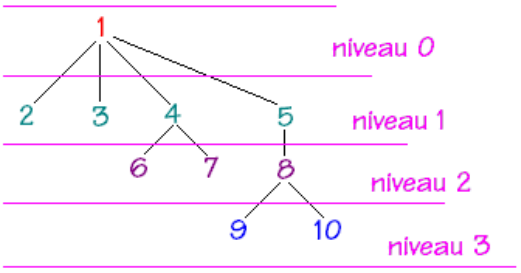



11. Une **feuille** ou noeud terminal est un noeud qui n'a pas de successeur. Tout noeud d'un arbre qui n'est pas une feuille est un **noeud interne**.

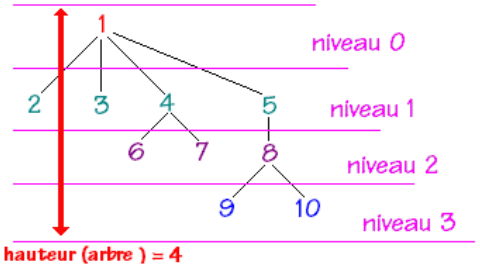
12. Un arbre dont tous les noeuds sont nommés est dit **étiqueté**. L'étiquette (ou nom du sommet) représente la "valeur" du noeud ou bien l'information associé au noeud.
Ci-dessous un arbre étiqueté dans les entiers entre 1 et 10.



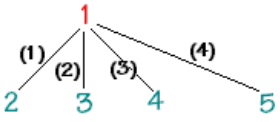

13. Nous conviendrons de définir **la hauteur** (ou profondeur ou niveau ) d'un noeud X comme égale au nombre de noeuds à partir de la racine pour aller jusqu'au noeud X. En reprenant l'arbre précédant et en notant h la fonction hauteur d'un noeud : Pour atteindre le noeud étiqueté 9 , il faut parcourir le lien $1--5$, puis $5--8$, puis enfin $8--9$ soient 4 noeuds donc 9 est de profondeur ou de hauteur égale à 4, soit $h(9) = 4$.Pour atteindre le noeud étiqueté 7 , il faut parcourir le lien $1--4$, et enfin $4--7$, donc 7 est de profondeur ou de hauteur égale à 3, soit $h(7) = 3$.**Par définition la hauteur de la racine est égal à 1.** $h(racine) = 1$ **(pour tout arbre non vide)**
14. Par définition **la hauteur d'un arbre** c'est le nombre de noeuds du chemin le plus long dans l'arbre. La hauteur h d'un arbre correspond donc au nombre de niveau maximum : h (Arbre) = max { h ( X ) / " X, X noeud de Arbre }



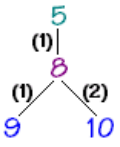

si Arbre = $\emptyset$ alors h( Arbre ) = 0
La hauteur de l'arbre ci-dessous :




15. **Le degré d'un noeud** est égal au nombre de ses descendants (enfants). Soient les deux exemples ci-dessous extraits de l'arbre précédent :



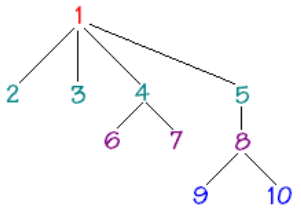

*Le noeud 1 est de degré 4, car il a 4 enfants*




*Le noeud 5 n'ayant qu'un enfant son degré est 1.*
*Le noeud 8 est de degré 2 car il a 2 enfants.*

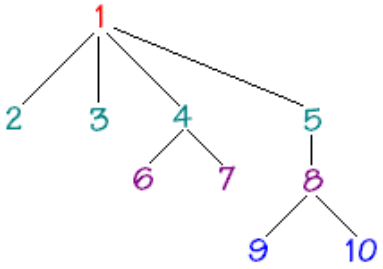
Remarquons que lorsqu'un arbre a tous ses noeuds de degré 1, on le nomme arbre dégénéré et que c'est en fait une liste.

16. Le **degré d'un arbre** est égal au plus grand des degrés de ses noeuds. d(Arbre) = max { d ( X ) / X, X noeud de Arbre } Soit à répertorier dans l'arbre ci-dessous le degré de chacun des noeuds :
17. On appelle **taille d'un arbre** le nombre total de noeuds de cet arbre : $taille(< r, fg, fd >) = 1 + taille(fg) + taille(fd)$




d°(1) = 4 d°(2) = 0
d°(3) = 0 d°(4) = 2
d°(5) = 1 d°(6) = 0
d°(7) = 0 d°(8) = 2
d°(9) = 0 d°(10) = 0

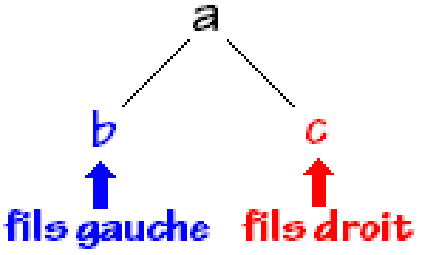
La valeur maximale est 4 , donc cet arbre est de degré 4.




Cet arbre a pour taille 10 (car il a 10 noeuds)

18. Un arbre est **un arbre n-aire** si tous les noeuds de l'arbre ont au plus n successeurs.

19. Un **arbre binaire** est arbre arbre dont les noeuds sont de degré 2 au plus. Les descendants (enfants) d'un noeud sont lus de gauche à droite et sont appelés respectivement fils gauche (descendant gauche) et fils droit (descendant droit) de ce noeud.



# 3 Génération d'arbres binaires

## 3.1 Nombre de Catalan

En mathématiques, et plus particulièrement en combinatoire, les nombres de Catalan forment une suite d'entiers naturels utilisé dans divers problèmes de dénombrement, impliquant souvent des objets définis de façon récursive. Le nombre de Catalan d'indice n, appelé n-iéme nombre de Catalan, est défini par :

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(2n+1)!n!} = \prod_{k=2}^{n}\frac{n+k}{k} \qquad \text{pour } n \geq 0 \qquad (1)$$

Les dix premiers nombres de Catalan (pour n de 0 à 9) sont :

1, 1, 2, 5, 14, 42, 132, 429, 1 430 et 4 862 Les nombres de Catalan interviennent dans de nombreux problèmes de combinatoire tels que :

1. $C_n$ est également le nombre de façons différentes de placer des parenthèses autour de n + 1 facteurs, pour préciser une expression faisant intervenir n fois une loi de composition interne non associative
2. $C_n$ est également le nombre d'arbres binaires entiers à n noeuds.
3. $C_n$ est aussi égal au nombre de façons de découper en triangles un polygone convexe à n+2 côtés en reliant certains de ses sommets par des segments de droite.
4. $C_n$ est le nombre de chemins monotones le long des arêtes d'une grille à n× n carrés, qui restent sous (ou au niveau de) la diagonale.
5. $C_n$ est le nombre de trajectoires de longueur 2n + 1 d'une marche aléatoire simple qui ont la propriété d'aller de la hauteur 0 à la hauteur 1 en restant négatif ou nul lors des 2n premières étapes.
6. $C_n$ est le nombre d'arbres planaires enracinés à n arêtes.
7. $C_n$ est égal au nombre de mots de Dyck de longueur 2n

## 3.2 Mots de DYCK

Nous nous intéressons ici aux mots sur l'alphabet constitué des deux lettres ( et ), c'est à dire les parenthèses ouvrante et fermante, et plus spécifiquement des mots correspondant aux suites de parenthèses intervenant dans une expression mathématique syntaxiquement correcte.

Par exemple, l'expression : $2 + (3 \times (x - 5) + (5 - x) \times y) \times (y - x)$ fournit le mot bien parenthésé : (()())() , alors que les mots )( ou ()( sont mal parenthésés.

Par la suite et pour faciliter la lecture, on posera $\Sigma = \{a, b\}$, $a$ désignant la parenthèse ouvrante et $b$ la parenthèse fermante. Ainsi, le mot bien parenthésé que nous venons de donner en exemple s'écrira : *aababbab*.

On conviendra aisément que l'ensemble $\mathcal{D}$ des expressions bien parenthésées, appelées aussi **mots de Dyck**, peut être défini à l'aide des règles de construction suivantes :

$$\begin{cases} \epsilon \in \mathbb{D} \\ (r,s) \in \mathcal{D}^2 \Longrightarrow arbs \in \mathcal{D} \end{cases}$$

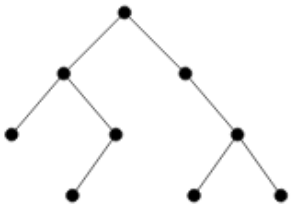
Et dans une expression bien parenthésée, toute parenthèse ouvrante ne peut correspondre qu'à une unique parenthèse fermante.

## 3.3 Bijection entre mots de Dyck et arbres binaires

Nous avons déjà mentionné que $C_n$ est le nombre d'arbres binaires entiers à $n$ noeuds et qu'il est également le nombre de mots de Dyck de longueur $2n$ ; on peut en effet établir une bijection entre un mot bien parenthésé et un arbre binaire en convenant que :

— au mot vide est associé l'arbre vide ;
— à un mot bien parenthésé de la forme *arbs* est associé l'arbre dont la racine a pour fils gauche l'arbre associé à $r$et pour fils droit l'arbre associé à $s$.

À l'inverse, lors d'un parcours en profondeur d'un arbre binaire, il suffit de noter par un $a$ le passage à gauche d'un nœud et par un $b$ le passage sous un nœud pour reconstituer le mot à partir de l'arbre. Par exemple, à l'arbre ci-dessous est associé le mot *aaabbaabbbabaabbab* qui correspond à cette expression bien parenthésée (((())(()))()(())() :



## 3.4 Algorithme de KNUTH

Notre programme de génération d'arbres binaires et éssentiellement basé sur l'algorithme **P** extrait du livre de **The Art of computer Programming** de **Knuth**.

Cet algorithme est basé sur le principe des mots de Dyck et génère pour un nombre $n$ donné ($n \geq 2$), tous les mots de Dyck de longueur $2n$.

**Comment l'algorithme fonctionne t-il ?**
L'algorithme se subdise en 5 parties.

**P1 :** On commence par générer le mot de Dyck le plus simple en générant $n$ paires de parenthèses successives.
Soit $A$ un tableau ; on initialise $A$ tel que $A_{2k-1}$ := '(' et $A_{2k}$ := ')' avec $1 \leq k \leq n$ ainsi pour pour $n = 3$ on aura ()()().
Tout mot bien parenthésé commence par une parenthèses ouvrante et se termine par une parenthèses fermente. On sait que $A_1$ = ')' et $A_{2n}$ = ')', on initialise alors

un indice $m := 2n-1$ qui permettra de parcourir les préfixes du mot bien parenthésé.

**P2 :** On vérifie que le mot est bien parenthésé, pour cela on vérifie que les préfixes modifés du mot bien parenthésé sont biens parenthèsés en s'assurant que $A_m =$ '(' et $A_{2k} =$ ')' pour $m < k \leq 2n$.

**P3 :** Dans la suite on parcours le mot de Dyck de la gauche vers la droite en commençant par la $m$-ième parenthèse.Dans cette partie on observe les cases $A_m$ et $A_{m-1}$ de notre tableau.

1. Si elles sont de la forme ')(' alors elles forment un préfixe mal parenthésé ; on le transforme en un préfixe bien parenthésé. Le tableau a été modifié , on a donc un nouveau mot bien parenthésé, on retourne en **P2** pour la vérification et on décrémente $m := m - 1$ ;
   **illustration**
   pour $n = 3$, on a ()()(), $m = 2n - 1$ donc $m = 5$ $A_m :=$ ')' si $A_{m-1} =$ '(' alors $A_m$ et $A_{m-1}$ formaient un préfixe mal parenthésé de la forme ')(', $A_{m-1} =$')'
   On passe de ()()() $\rightarrow$ ()(())
2. sinon on passe à **P4**.

**P4 :** A cet niveau, $A_m$ et $A_{m-1}$ ne formaient pas un préfixe mal parenthésé. On parcours donc le reste du mot à la recherche d'un autre préfixe mal parenthésé.

Pour le faire on initialise une variable $j := m - 1$ et $k := 2n - 1$ et tant que $A_j$ = '(' ; $A_j :=$ ')'$A_k :=$ '(' ; $j := j - 1$ et $k := k - 2$ après on continue en **P5**.
**illustration**
en **P3** on avait obtenu ()(()). On suppose qu'on a effectué la vérification en **P2**.On arrive de nouveau en **P3** $m = m - 1$, $m = 4$, $A_m :=$ ')'et $A_{m-1} \neq$ '(' on constate aisément que $A_m$ et $A_{m-1}$ ne forment pas ')(', on passe de ()(()) $\rightarrow$ ()())) on arrive en ici en **P4**

,

1. $j = 3$, $k = 5$, $A_j :=$ ')'et $A_k :=$ '(' on passe de ()())) $\rightarrow$ ()))() ;
2. $j = 2$, $k = 3$ et $A_j \neq$ ')' on continue en **P5**

**P5** Ici si $j = 0$ cela signifie qu'on a parcouru le mot bien parenthésé sans au moins un préfixe mal parenthésé et ce mot doit être de la forme $((((\cdots))\cdots))$.
Sinon $A_j =$ '('. On a trouvé un préfixe mal parenthésé qui a été remplacé par un préfixe bien parenthésé ; on a donc un autre mot bien parenthésé $m = 2n - 1$ et on retourne en **P2** pour la vérification.
**illustration**
On a $j = 2$ donc $A_j :=$ ')' on passe de ()))() $\rightarrow$ (())() qui le nouveau mot parenthésé obtenu et retourne encore en **P2** pour la vérification.

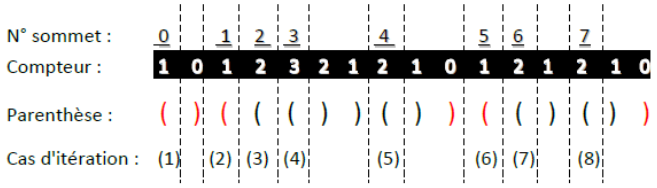
L'algorithme refait le différentes étapes sur le nouveau mot obtenu. Le but de cet algorihme est de parcourir une chaine de parenthèse bien parenthésée, de remplacer le premier préfixe mal parenthésé rencontré par un préfixe bien parenthésé de même longueur.On obtient alors un nouveau mot bien parenthésé qui sera lui aussi parcouru

à la recherche d'un préfixe mal parenthésé. Il fonctionne de manière de recursive à partir d'un mot bien parenthésé pour en obtenir un autre.

L'algorithme contient une fonction d'affichage qui coûte $2n$ pour chaque mot bien parenthésé, la partie **P4** avec la boucle tantQue est executé que le tiers du temps et coûte $m-2$ fois pour mot bien parenthésé et $m = 2n-1$ ; le tout se fait $n$ fois . D'ou la complexité en $O(n^2)$.

## 3.5 Transformation des parenthèses en forêt

Après la génération des parenthèses, nous avions une fonction 'foret()' qui permettait de les transformer en forêt. Pour cela nous avons déclaré un compteur *cpt* pour compter les parenthèses entrantes '(' et sortantes ')' qui s'incrémente de 1 les parenthèses entrantes et se décrémente de 1 les parenthèses sortantes , un autre $'num_sommet'$ pour compter le nombre de sommets en comptant seulement les parenthèses entrantes et une varaible 'tmp[]' contiendra la valeur du père d'un sommet lors du parcours des parenthèses tel que 'tmp[cpt-1]' donne le numéro du sommet et $num_sommet - 1$ nous donne le père du sommet. Lors du parcours si on rencontre un préfixe de la forme )( cela signifie que le nouveau sommet a le mêmes père que le précédent par contre si on rencontre un préfixe de la forme (( alors le nouveau sommet a un père différent du précédent et on calcule à nouveau $num_sommet - 1$.Plus de détails sur le schémas ci-dessous :

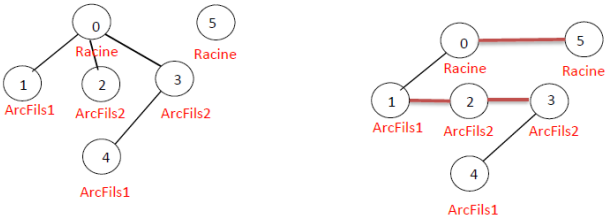



On suppose que les parenthèse se trouvent dans un tableau a[] préalablemen

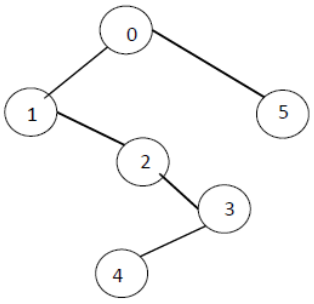
1. tmp[1-1]=tmp[0]=0-1= -1 Donc, le sommet '0' n'a pas de père .
2. tmp[1-1]=tmp[0] car a[i-1]=')' et a[i]='(' . On change rien pour 'tmp[0]'et sa valeur est $-1$ . Donc le sommet '1' n'a pas de père ;
3. tmp[2-1]=tmp[1]=2-1=1 car a[i-1]='(' et a[i]='(' on insère une nouvelle valeur dans tmp[1]', le sommet '2' est le fils de '1' ;
4. tmp[3 -1]=tmp[2]=3-1=2 idem comme (3), le sommet '3' est le fils de '2' ;
5. tmp[2-1]=tmp[1] car a[i-1]=')' et a[i]='(' . On change rien pour 'tmp[1]'et sa valeur est 1, le sommet '4' est le fils de '1' ;
6. tmp[1-1]=tmp[0] car a[i-1]=')' et a[i]='(' . On change rien pour 'tmp[0]'et sa valeur est - 1 . Donc le sommet '5' n'a pas de père ;
7. tmp[2-1]=tmp[1]=6-1=5 car a[i-1]='(' et a[i]='(' .Nous insére un nouveau valeur dans 'tmp[1]', le sommet '6' est le fils de '5' ;
8. tmp[2-1]=tmp[1] car a[i-1]=')' et a[i]='(' . On change rien pour 'tmp[1]'et sa valeur est 5 ,le sommet '7' est le fils de '5'.

### 3.6 Transformation des forêts en arbres binaires

Pour les forêts on nomme l'arc entre un sommet et du premier fils gauche par 'ArcFils1' et les arcs avec ses autres fils par 'ArcFils2' et tous les sommets sans père sont nommés 'Racine'. Notre fonction de transformation en arbre binaire se nomme 'binary()', on lie d'abord toutes les 'Racines' comme des branches successives à droite. La valeur du premier fils gauche 'ArcFils1'd'un noeud est conservé car il sera le père des frères qu'il avait dans la forêt. Ensuite , les autres fils 'ArcFils2' de ce même noeud seront les fils à droit de leur frère le plus à gauche.
Exemple pour une forêt à 5 sommets on a :




L'arbre binaire correspondant est :




# 4 Conclusion

Il existe plusieurs types d'arbres et différentes manières de les representer. Les mots de dyck sont une façon bien particulère de representer les arbres binaires. Si on s'étend aux différentes bijections l'on pourrait se demander s'ils ne permettent pas aussi de representer les arbres planaires