UPMC
SORBONNE UNIVERSITÉS

Rapport Projet STL

Génération générique récursive de structures décomposables

Camara Sidi

Encadrants :
Dien Matthieu
Genitrini Antoine
Rovetta Christelle

21 mai 2017

Table des matières

1 Introduction

Étant donné une famille de structures, définies par une grammaire, et une taille cible, un générateur aléatoire uniforme construit une structure de la famille pour la taille demandée. La contrainte d'uniformité d'un générateur assure que chaque structure de la taille a exactement la même probabilité d'être générée. Le but de ce projet est d'implémenter, en Ocaml, un algorithme de génération aléatoire uniforme des arbres binaires et ternaires de taille exactement la taille cible. Cet algorithme s'appelle "algorithme de génération récursive". Il se base sur une décomposition inductive des arbres. Le choix du langage Ocaml est du au fait que les algorithmes implémentés seront intégrés dans l'outil arbogen.

Dans ce rapport, on commencera par définir les classe combinatoire, elle intervient lors d'analyse des séries génératrices.

Après avoir défini et spécifié les arbres binaires et ternaires, on s'intéressera ensuite au rôle des séries génératrices ordinaires dans le dénombrement de structures combinatoires ouvrant ainsi une introduction à la méthode symbolique.

Avant d'entamer la dernière partie on se penchera sur la génération aléatoire, de ces fouilles d'arbres ainsi que l'étude de leur complexité.

On terminera par une description d'impléméntation et une étude expérimentale des algorithmes présentés , sans oublier de présenter des exemples d'arbres générés.

2 Classes Combinatoire

Définition Une classe combinatoire est par définition un ensemble $\mathcal{A}$ muni d'une application $|.| : \mathcal{A} \to \mathbb{N}$ appelée taille. On demande de plus que, pour chaque n, le nombre d'éléments de taille n soit fini.
Pour les arbres binaires on considéra la taille comme étant le nombre de nœuds internes de l'arbre.

Exemple 1
La classe combinatoire $\mathcal{B}$ des arbres binaires est définie de manière récursive, un arbre binaire est soit vide, soit un noeud constitué de deux sous-arbres binaires.

FIGURE 1 – arbres binaires de taille un

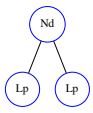


FIGURE 2 – arbres binaires de taille deux

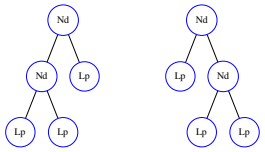
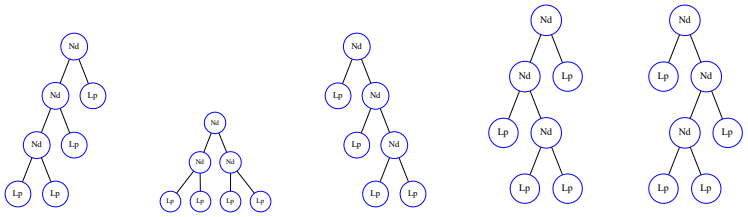


FIGURE 3 – arbres binaires de taille trois



Exemple 2

La classe combinatoire des mots sur l'alphabet A :={a, b}, notée A*, contient les suites finies d'éléments de A. De manière alternative, A* peut se définir comme étant la classe combinatoire obtenue par l'opération suite appliquée à A ce dernier est vu comme une classe combinatoire qui possède exactement deux objets de taille 1. Ses éléments de taille inférieure à trois sont

taille 0	taille 1	taille 2	taille 3
ϵ	a, b	aa, ab, ba, bb	aaa, aab, aba, abb, baa, bab, bba, bbb

avec ϵ un objet de taille 0.

3 Arbre binaire et ternaire

Les outils présentés précédemment nous permettrons de bien étudier certaines propriétés des arbres binaires et ternaires.

Définition 1 : Les arbres de Catalan, ou arbres binaires sont les arbres dont tous les noeuds internes ont une arité de 2. Les arbres ternaires sont des arbres dont tous les noeuds internes ont soit 1, soit 2, soit 3 comme arité. On utilisera le nombre de noeuds internes comme taille pour les arbres.

Spécification

On considéra les notations suivantes pour spécifier les arbres binaires et ternaires.
ε : classe combinatoire contenant un seul objet de taille 0
z : classe combinatoire contenant un seul objet de taille 1

Définition 2

Soient $\mathcal{A}$ et $\mathcal{B}$ deux classes d'objet combinatoires l'union disjointe de $\mathcal{A}$ et $\mathcal{B}$ donne $\mathcal{A} + \mathcal{B}$ dont les élément sont soit issus de $\mathcal{A}$ soit $\mathcal{B}$. Produit cartésien de $\mathcal{A}$ et $\mathcal{B}$ donne $\mathcal{A} \times \mathcal{B}$ dont les éléments sont constitués des paires ordonnées d'objets de $\mathcal{A}$ et $\mathcal{B}$.

Arbre binaire

Un arbre binaire est soit une feuille, soit un noeud interne composé d'un fils gauche et droit qui sont aussi des arbres binaires.
$\mathcal{B}=\varepsilon+z\times\ \mathcal{B}\times\ \mathcal{B}$.

Arbres ternaire

Un arbre ternaire est soit une feuille, soit un noeud interne constitué soit d'un sous-arbres, soit de deux sous-arbres, soit de trois sous-arbres qui sont aussi des arbres ternaires.
$\mathcal{T} = \varepsilon + z \times \mathcal{T} + z \times \mathcal{T} \times \mathcal{T} + z \times \mathcal{T} \times \mathcal{T} \times \mathcal{T}$.

4 Séries génératrice ordinaire

Les fonctions génératrices constituent un outil très puissant pour résoudre des récurrences et des problèmes de dénombrement. Elles sont particulièrement utiles pour calculer la complexité en moyenne d'algorithmes, qui nécessite de calculer le coût moyen sur toutes les structures d'un certain type de taille n. Soit b_n le nombre d'éléments de taille n de B. La série génératrice ordinaire associée à B est par définition la série B(z) définie par

$$B(z) = \sum_{n=0}^{\infty} b_n.z^n$$

avec z une variable.

Opérations sur les séries générations ordinaires Étant données les séries génératrices

$$A(z) = \sum_{n=0}^{\infty} a_n z^n, B(z) = \sum_{n=0}^{\infty} b_n z^n,$$

représentant les suites $\{a_0, a_1,....a_k,...\}$ et $\{b_0, b_1,....b_k,...\}$, on cherche à avoir d'autres séries génératrices et leur suite associée en effectuant certain nombre d'opération sur les séries $A(z)$ et $B(z)$.

Théorème 4.1 *[SF13] Soient A et B deux classes d'objet combinatoires. Si A(z) et B(z) sont les séries génératrices ordinaires énumératives respectives de A et B alors :*
A(z) + B(z) est la série génératrice ordinaire énumérative de A + B
A(z)B(z) est la série génératrice ordinaire énumérative énumérative de A ×B.

Le théorème 4.1 établie une correspondance simple entre une opération sur deux classes combinatoires et la série génératrice associée. Il permet de déterminer directement des relations fonctionnelles de séries génératrices à partir de définitions constructives d'objet combinatoires.

Addition

$$A(z) + B(z) = \sum_{n=0}^{\infty} a_n + b_n z^n$$

Produit cartésien

$$A(z)B(z) = \sum_{n\geq 0}^{\infty} \sum_{0\leq k\leq n}^{\infty} a_k b_{n-k} z^n$$

5 Méthode Symbolique

La méthode symbolique transforme la spécification en une équation fonctionnelle vérifier par la fonction génératrice.

Équation fonctionnelle Les équations fonctionnelles nous permettent de résoudre des problèmes de dénombrement.
Considérons les deux classes combinatoires suivantes :
A=$\{\varepsilon\}$ la classe combinatoire contenant un seul objet de taille 0 et B=$\{z\}$ la classe combinatoire contenant un seul objet de taille 1. La série génératrice associée à A est A(z)=1 et la série génératrice associée à B est B(z)=z

Équation fonctionnelle associée aux arbres binaire
Le théorème 4.1 nous permet de traduire directement la spécification en une équation fonctionnelle.

$$B = \varepsilon + z \times B \times B \Longrightarrow B(z) = 1 + z \times B(z)^2$$

Équation fonctionnelle associée aux arbres ternaires
Le théorème 4.1 nous permet de traduire directement la spécification en une 'équation fonctionnelle.

$$\mathcal{T} = \varepsilon + z \times T + z \times T \times T + z \times T \times T \times T \Longrightarrow T(z) = 1 + zT(z)^2 + zT(z)^3.$$

Exemple classe des mots binaires
Soit M la classe des mots binaires. Un mot binaire est soit un mot vide ou bien une paire ordonnée comportant un 0 ou bien un 1 suivi d'un mot binaire. Symboliquement,

$$M = \epsilon + \{0, 1\} \times M.$$

Le théorème 4.1 nous permet de traduire cette équation sous forme symbolique par une équation fonctionnelle vérifier par la série génératrice de dénombrement. On a ainsi

$$M(z) = 1 + 2M(z).$$

Équation de Récurrence

Équation de Récurrence associé aux arbres binaires
Soit $[z^{n+1}]B(z) = b_{n+1}$ le coefficient de z^{n+1} dans $B(z)$, n $\in \mathbb{N}$ on a

$$[z^{n+1}]B(z) = [z^{n+1}](1 + zB^2(z))$$

$$= [z^n]B^2(z) = [z^n] \sum_{k=0}^{\infty} b_k z^k \sum_{j=0}^{\infty} b_j z^j$$

$$= [z^n] \sum_{k=0}^{n} b_k b_{n-k} z^n$$

$$\Longrightarrow \begin{cases} b_{n+1} = \sum_{k=0}^{n} b_k . b_{n-k} \\ b_0 = 1 \end{cases}$$

Exemple arbre binaire de taille 7 :
b_7=b_0b_6 + b_1b_5 + b_2b_4 + b_3b_3 + b_4b_2 + b_5b_1 + b_6b_0 = 132 + 42 + 28 + 25 + 28 + 42 + 132 = 429
il y a donc 429 possibilités différentes de construire un arbre binaire de taille 7.

Équation de Récurrence associé aux arbres ternaires

Soit $[z^{n+1}]T(z) = t_{n+1}$ le coefficient de z^{n+1} dans $T(z)$, n $\in \mathbb{N}$ on a

$$[z^{n+1}]T(z) = [z^{n+1}](1 + zT(z) + zT^2(z) + zT^3(z))$$

$$= [z^n]T(z) + [z^n]T^2(z) + [z^n]T^3(z) = t_n + [z^n]\sum_{k=0}^{\infty} t_k z^k \sum_{j=0}^{\infty} t_j z^j + [z^n]\sum_{i=0}^{\infty} t_i z^i \sum_{k=0}^{\infty} t_k z^k \sum_{j=0}^{\infty} t_j z^j$$

$$= t_n + [z^n]\sum_{k=0}^{n} t_k t_{n-k} z^n + [z^n]\sum_{i=0}^{n}\sum_{k=0}^{n-i} t_i t_k t_{n-i-k} z^n$$

$$\Longrightarrow \begin{cases} t_{n+1} = t_n + \sum_{k=0}^{n} t_k t_{n-k} + \sum_{i=0}^{n}\sum_{k=0}^{n-i} t_i t_k t_{n-k-i} \\ t_0 = 1 \end{cases}$$

Exemple arbre ternaire de taille 3 :

$$t_3 = t_2 + t_0 t_2 + t_1 t_1 + t_2 t_0 + t_0 t_0 b_2 + t_0 t_1 t_1 + t_0 t_2 t_0 + t_1 t_0 t_1 + t_1 t_1 t_0 + t_2 t_0 t_0$$
$$= 18 + 18 + 9 + 18 + 18 + 9 + 18 + 9 + 9 + 18 = 144$$

il y a donc 144 possibilités différentes de construire un arbre ternaire de taille 3.

6 Génération aléatoire

Un générateur aléatoire de structures combinatoires est un algorithme qui tire aléatoirement un objet b d'une classe combinatoire B selon une distribution donnée.
On considère la génération dans la loi uniforme. L'algorithme prend un entier n en entrée et renvoie un objet de B de taille n tel que tous les objets de taille n de B aient la même probabilité d'être tirés.
On utilise une méthode à la base de génération exhaustive. Chaque arbre a un numéro, on génère aléatoirement(uniformément) un numéro et on construit l'arbre associé.

Méthode récursive

Après spécification d'une classe combinatoire, on peut obtenir un générateur aléatoire pour cette classe en composant des générateurs élémentaires.
Pour construire un arbre de taille n on aura besoin de générer uniformément un entier
dans l'intervalle [0,b_n -1].

Algorithme de génération aléatoire d'arbres binaires

Exemple de construction d'un arbre de taille 7

1. b_7=b_0b_6 + b_1b_5 + b_2b_4 + b_3b_3 + b_4b_2 + b_5b_1 + b_6b_0 = 132 + 42 + 28 + 25 + 28 + 42 + 132 = 429
2. généré aléatoirement $n \in [0, b_7 - 1]$ soit $n = 300$ on veut donc construire l'arbre numéro 300 parmi les 429 arbre
   les arbres de numéro $n \in [0, 131]$ sont de la forme b_0b_6
   les arbres de numéro $n \in [132, 132 + 41]$ sont de la forme b_1b_5
   les arbres de numéro $n \in [132 + 42, 132 + 42 + 27]$ sont de la forme b_2b_4
   les arbres de numéro $n \in [132 + 42 + 28, 132 + 42 + 28 + 24]$ sont de la forme b_3b_3

   .
   .
   ect.

3. $n - 132 = 168 > 0 \Longrightarrow$ l'arbre numéro 300 n'est pas de la forme $b_0 b_6$
   $168 - 42 = 128 > 0 \Longrightarrow$ l'arbre numéro 300 n'est pas de la forme $b_1 b_5$
   $128 - 28 = 100 > 0 \Longrightarrow$ l'arbre numéro 300 n'est pas de la forme $b_2 b_4$
   $100 - 25 = 75 > 0 \Longrightarrow$ l'arbre numéro 300 n'est pas de la forme $b_3 b_3$
   $75 - 28 = 45 > 0 \Longrightarrow$ l'arbre numéro 300 n'est pas de la forme $b_4 b_2$
   $45 - 42 = 3 > 0 \Longrightarrow$ l'arbre numéro 300 n'est pas de la forme $b_5 b_1$
   $3 - 132 = -129 < 0 \Longrightarrow$ l'arbre numéro 300 est de la forme $b_6 b_0$

4. retourne l'arbre dont le sous-arbre gauche est un arbre binaire de taille 6 (construit récursivement) et le sous-arbre droit est une feuille.

Les algorithmes :

calculeBn : calcule le nombre d'arbres binaires de taille n.

taillefilsGD : retourne la taille du fils droit et gauche d'arbre de taille n.

binaire : construit l'arbre binaire de taille n.

Calcul des b_n Lorsqu'on veut construire un arbre binaire de taille n on doit calculer au préalable le nombres d'arbres possibles de taille i, $i \in [1..n]$.

Algorithm 1 calcule des b_n

```
1: function CALCULEBN(n)
2:     b_0 ← 1
3:     for i = 1 to n do
4:         s ← 0
5:         for j = 0 to i − 1 do
6:             s ← s + b_j.b_{i−j−1}
7:         end for
8:         b_i ← s
9:     end for
10: end function
```

Complexité en $\mathrm{O}(n^2)$: $\forall i \in |1, n|$ on effectue i multiplication pour calculer b_i.
Le nombre total d'opérations de multiplication est donc $1 + 2 + 3 + ... + n = \frac{n(n+1)}{2}$.
$\Longrightarrow$l'algorithme est en $\mathrm{O}(n^2)$

Calcule de la taille des sous arbre L'algorithme suivant retourne la taille des sous-arbres d'un arbre binaire de taille n son principe est le suivant :

1. On tire un r $\in [0...b_n - 1]$
2. On cherche à avoir i,j tel que l'arbre à construire est de la forme $b_i b_j$ avec i et j la taille du fils gauche et droit pour cela on soustrait $b_k b_{n-1-k}$, k$\in$ $[0..n-1]$ à r ,jusqu'à ce que la valeur obtenue après soustraction soit négative .

Algorithm 2 calcule de la taille des fils

```
1: function TAILLEFILSGD(n,r)
2:     k ← 0
3:     while r − b_k.b_{n−k−1} > 0 do
4:         r ← r − b_k.b_{n−k−1}
5:         k ← k + 1
6:     end while
7:     k ← k − 1
8:     r_l ← [r ÷ b_k]
9:     r_r ← r mod b_k
10:    return (r_l, k) et (r_r, n − 1 − k)
11: end function
```

Complexité en O(n) : Pour connaître la taille des sous arbres l'algorithme détermine la forme de l'arbre $b_l.b_r$ avec l et r la taille respective du fils gauche et droit. Au pire l'arbre est de la forme $b_0.b_{n-1}$ on itère donc n fois.

Algorithme de construction d'arbre binaire Principe :

1. Tirer uniformément une valeur r$\in [0..b_n - 1]$ avec n la taille de l'arbre à construire.
2. Si n=0 on retourne une feuille
3. On appelle Algorithm 2 pour avoir la taille du fils gauche et droit.
4. On construit récursivement les fils gauche et droit.

Algorithm 3 construction de l'arbre binaire

```
1: function BINAIRE(n,r)
2:     (r_l, l) et (r_r r) ← TAILLEFILSGD(n,r)
3:     if l = 0 and r = 0 then
4:         return (ε, Z, ε)
5:     else if l > 0 and r = 0 then
6:         ag ← BINAIRE(r_l, l)
           return (ag, Z, ε)
7:     else if l = 0 and r > 0 then
8:         ad ← BINAIRE(r_r, r) return (ε, Z, ad)
9:     else
10:        ad ← BINAIRE(r_r, r)
11:        ag ← BINAIRE(r_l, l) return (ad, Z, ag)
12:    end if
13: end function
```

Construction aléatoire d'arbres ternaires

Exemple 1 construction d'arbre ternaire de taille 3

1. $t_3 = t_2 + t_0t_2 + t_1t_1 + t_2t_0 + t_0t_0b_2 + t_0t_1t_1 + t_0t_2t_0 + t_1t_0t_1 + t_1t_1t_0 + t_2t_0t_0 = 18+18+9+18+18+9+18+9+9+18 = 144$
2. généré aléatoirement $n \in [0, b_3 - 1]$ soit $n = 82$ on veut donc construire l'arbre ternaire numéro 82 parmi les 144 arbres ternaires
   les arbres de numéro $n \in [0, 17]$ sont de la forme t_2
   les arbres de numéro $n \in [18, 18 + 17]$ sont de la forme t_0t_2
   les arbres de numéro $n \in [18 + 18, 18 + 18 + 8]$ sont de la forme t_1t_1
   les arbres de numéro $n \in [18 + 18 + 9, 18 + 18 + 9 + 17]$ sont de la forme t_2t_0
   les arbres de numéro $n \in [18 + 18 + 9 + 18, 18 + 18 + 9 + 18 + 17]$ sont de la forme $t_0t_0t_2$
   .
   .
   ect.
3. $n - 18 = 64 > 0 \Longrightarrow$ l'arbre numéro 82 n'est pas de la forme t_2
   $64 - 18 = 46 > 0 \Longrightarrow$ l'arbre numéro 82 n'est pas de la forme t_0t_2
   $46 - 9 = 37 > 0 \Longrightarrow$ l'arbre numéro 82 n'est pas de la forme t_1t_1
   $37 - 18 = 19 > 0 \Longrightarrow$ l'arbre numéro 82 n'est pas de la forme t_2t_0
   $19 - 18 = 1 > 0 \Longrightarrow$ l'arbre numéro 82 n'est pas de la forme $t_0t_1t_1$
   $1 - 9 = -8 < 0 \Longrightarrow$ l'arbre numéro 82 est de la forme $t_0t_2t_0$
4. retourne l'arbre ternaire dont le premier sous-arbre est une feuille, le second un arbre ternaire de taille 2(construit récursivement) et le troisième sous-arbre une feuille.

Exemple 2 construction d'arbre ternaire de taille 3

1. $t_3 = t_2 + t_0t_2 + t_1t_1 + t_2t_0 + t_0t_0b_2 + t_0t_1t_1 + t_0t_2t_0 + t_1t_0t_1 + t_1t_1t_0 + t_2t_0t_0 = 18+18+9+18+18+9+18+9+9+18 = 144$
2. généré aléatoirement $r \in [0, b_3 - 1]$ soit $n = 10$ on veut donc construire l'arbre ternairenuméro 10 parmi les 144 arbres ternaires
3. $n - 18 = -8 < 0 \Longrightarrow$ l'arbre numéro 10 est de la forme t_2
4. retourne l'arbre ternaire qui a un seul sous-arbre ternaire de taille 2(construit récursivement).

Fonction calculant les t_n

L'algorithme 4 a une fonction similaire à l'algorithme 1, il calcule le nombre d'arbres ternaires de taille i, $i \in [1..n]$.

Algorithm 4 calcul des t_n pour arbre ternaire

```
1: function Tn3(n)
2:     t_0 ← 1
3:     for i = 1 to n do
4:         s ← 0
5:         for j = 0 to i − 1 do
6:             s ← s + t_j.t_{i−j−1}
7:         end for
8:         for j = 0 to i − 1 do
9:             for k = 0 to i − 1 − j do
10:                s ← s + t_j.t_k.t_{i−1−j}
11:            end for
12:            t_i ← s
13:        end for
14:    end for
15: end function
```

Une amélioration possible est d'éviter une opération de multiplication.

Algorithm 5 calcul des b_n amélioré pour arbre ternaire

```
1: function Tn3A(n)
2:     t_0 ← 1
3:     for i = 1 to n do
4:         s ← 0
5:         for j = 0 to i − 1 do
6:             x ← i − j − 1
7:             m_{jx} ← s + t_j.t_{i−j−1}
8:             s ← s + m_{jx}
9:         end for
10:        for j = 0 to i − 1 do
11:            for k = 0 to i − 1 − j do
12:                s ← s + t_{i−1−j}.m_{kj}
13:            end for
14:            t_i ← s
15:        end for
16:    end for
17: end function
```

Complexité

L'algorithme est en $\mathrm{O}(n^3)$

Preuve :

$\forall i \in [1..n]$ on effectue i + 1+2+3+.....+i=$i + \frac{i(i+1)}{2}$

$$= \sum_{i=1}^{n} i + \frac{1}{2}(\sum_{i=1}^{n} i^2 + i)$$

$$= \frac{n(n+)}{2} + \frac{1}{2}(\frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2})$$

$\Longrightarrow$ algorithme en $\mathrm{O}(n^3)$

Algorithm 6 Arbre ternaire

function TERNAIRE(n,r)
  $r \leftarrow r - t_{n-1}$
  $p \leftarrow t_{n-1}$
  while $r > 0$ **do**
    $r \leftarrow r - p$, $p \in [t_0.t_{n-1}, t_1.t_{n-2}, ..., t_{n-1}.t_0, t_0.t_0.t_{n-1}, t_0.t_1.t_{n-2}, ..., t_{n-1}.t_0.t_0]$
  end while
  $r \leftarrow r + p$
  if $r = t_{n-1}$ **then**
    $r1 \leftarrow$ r mod t_{n-1}
    return $(\mathcal{Z}, taillefilsGD(n-1, r1))$
  else if $r \leq t_0.t_{n-1}$ **then**
    $r_l \leftarrow [r \div t_k]$
    $r_r \leftarrow$ r mod t_k
    return $(\mathcal{Z}, taillefilsGD(k, r_l), taillefilsGD(n-k-1, r_r))$
  else
    $r_1 \leftarrow$ r mod t_i
    $p \leftarrow [r \div t_i]$
    $r_2 \leftarrow$ p mod t_j
    $r_3 \leftarrow [p \div t_j]$
    return $(\mathcal{Z}, taillefilsGD(i, r_1), taillefilsGD(j, r_2), taillefilsGD(n-1-i-j, r_3))$
  end if

end function

Algorithme de la construction des l'arbre

7 Implémentation

OCaml est le langage de programmation utilisé pour implémenter les algorithmes du projet. La principale difficulté rencontrée lors de l'implémentation été que je ne connaissait pas le langage OCaml, l'implémentation de la première partie du projet (arbre binaire) m'a donc pris plus du temps que le reste. Pour implémenter les arbres binaire j'avais utilisé un tableau pour stocker les valeurs des b_n et une matrice pour les calcules intermédiaire de $b_i.b_j$ histoire de ne pas effectuer plusieurs fois les même calculs. Conséquence d'utilisation des tableaux on ne peut pas construire un arbre de taille $n > 2000$(problème mémoire). La solution est d'utiliser des fichiers pour stocker les valeurs, ainsi on peut construire un arbre de toute taille.

Exemple d'arbres générés

Pour la représentation graphique d'arbres générés j'utilise l'application GraphViz.

Figure 4 – arbre n=10

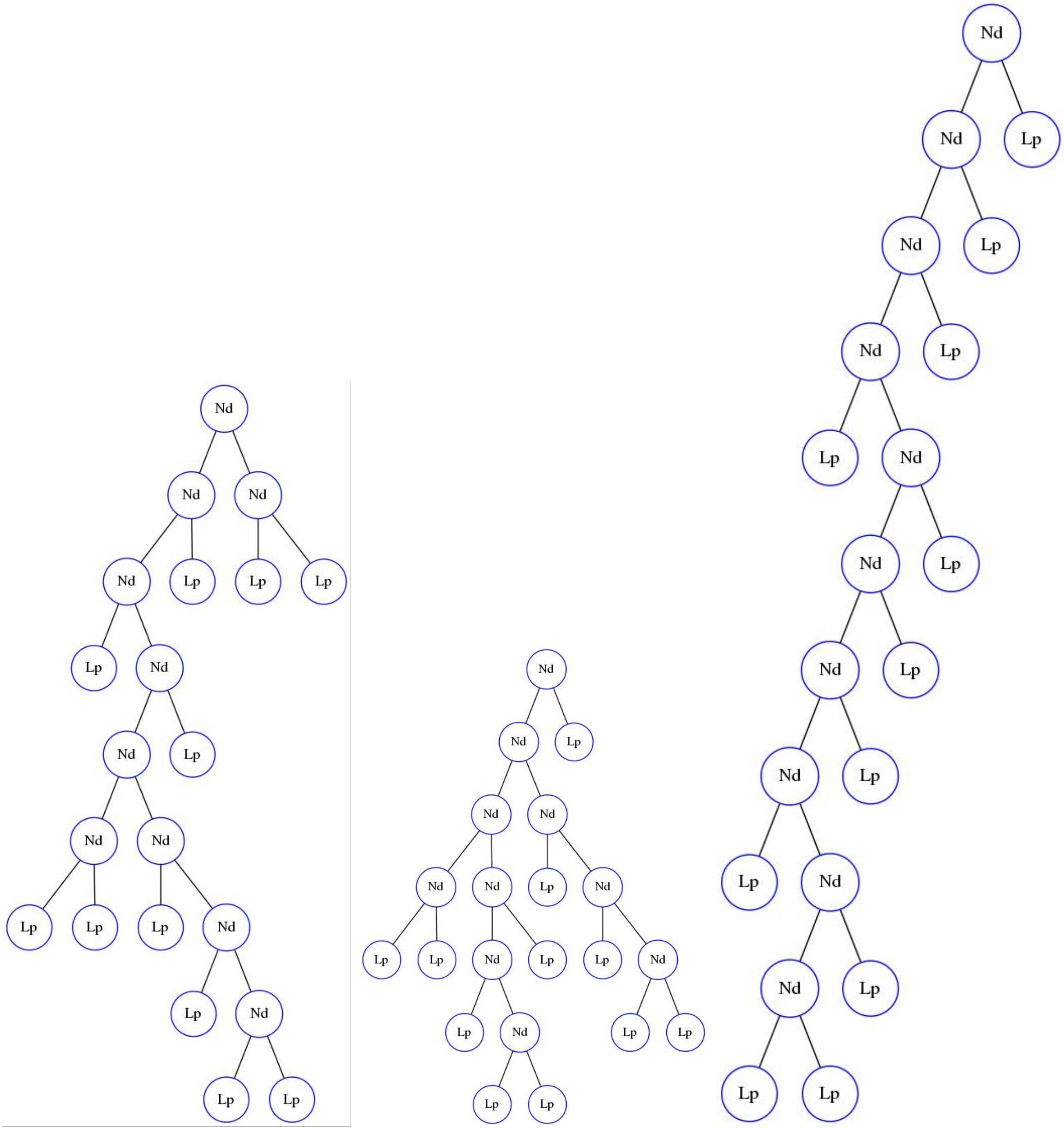
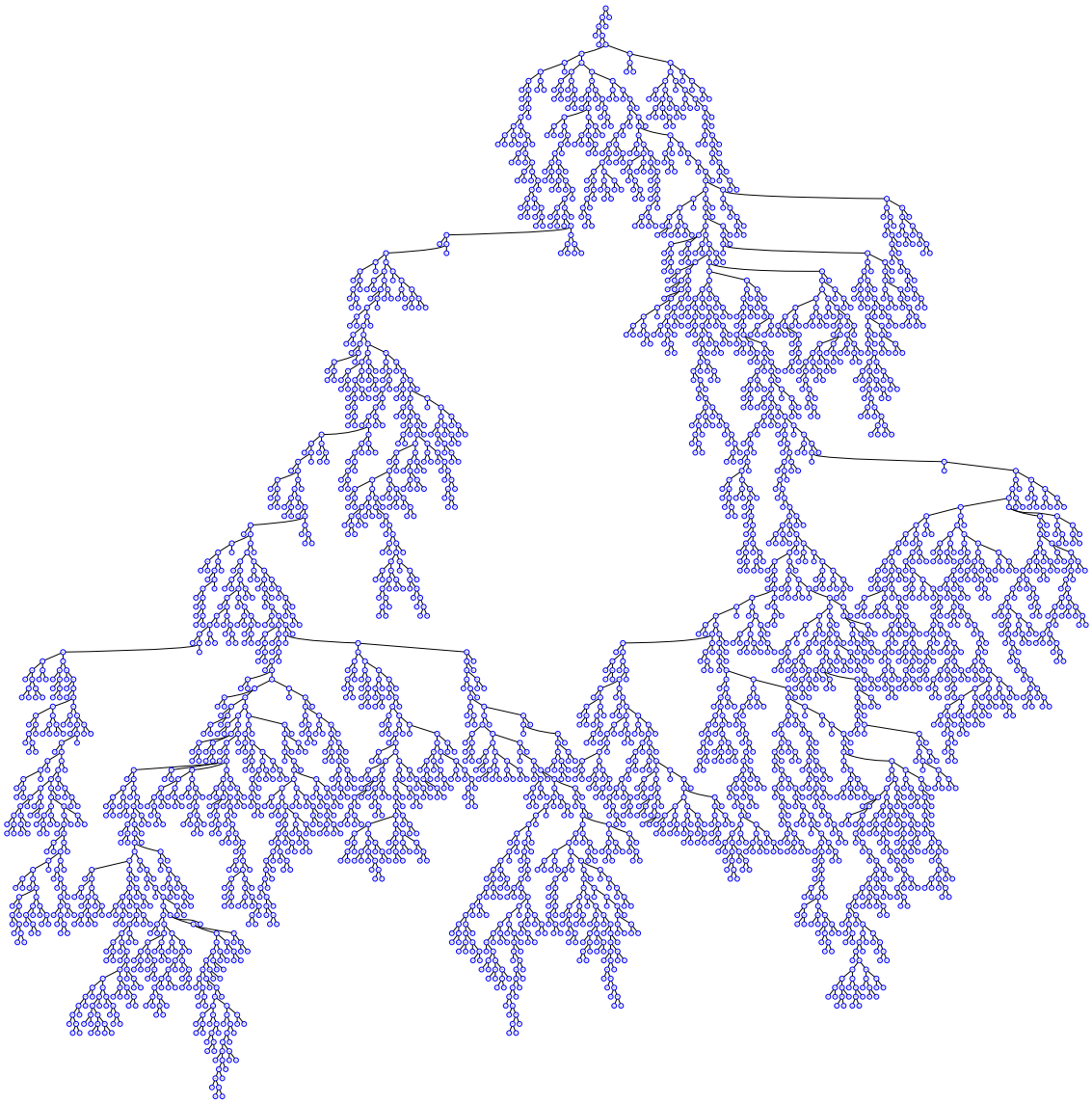


FIGURE 5 – arbre n=2000



Arbre binaire

FIGURE 6 – Arbre Ternaire n=10

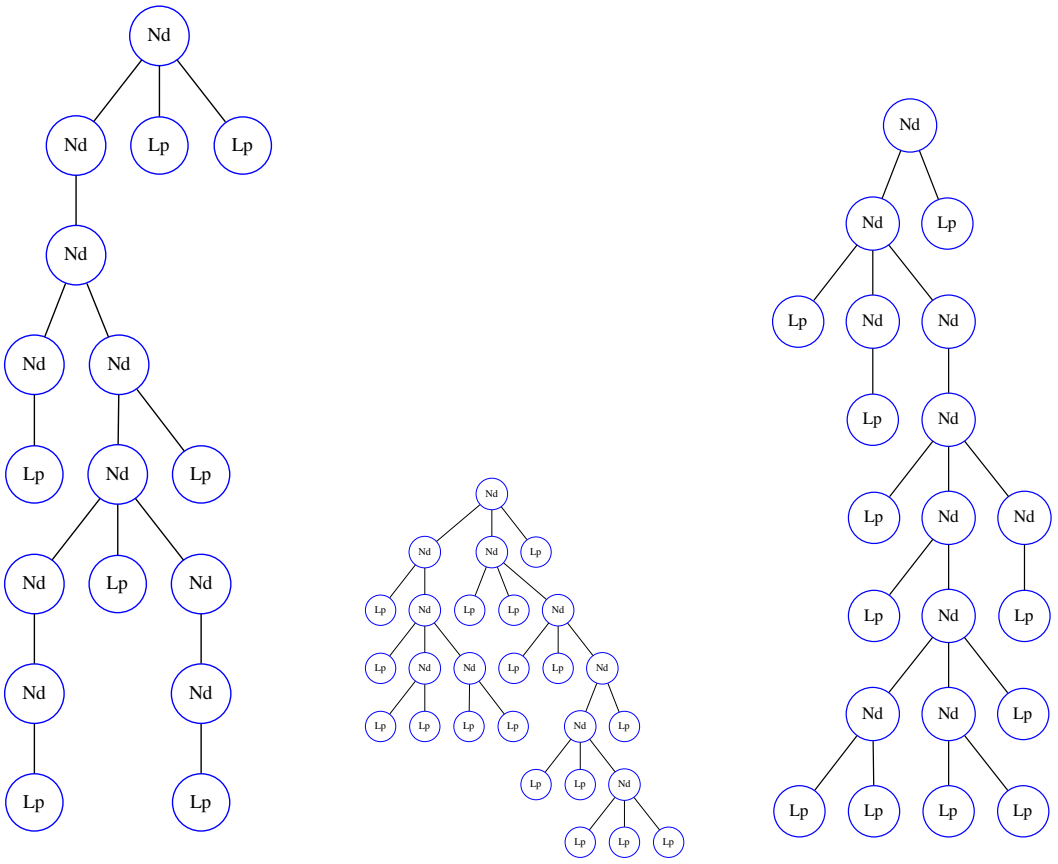
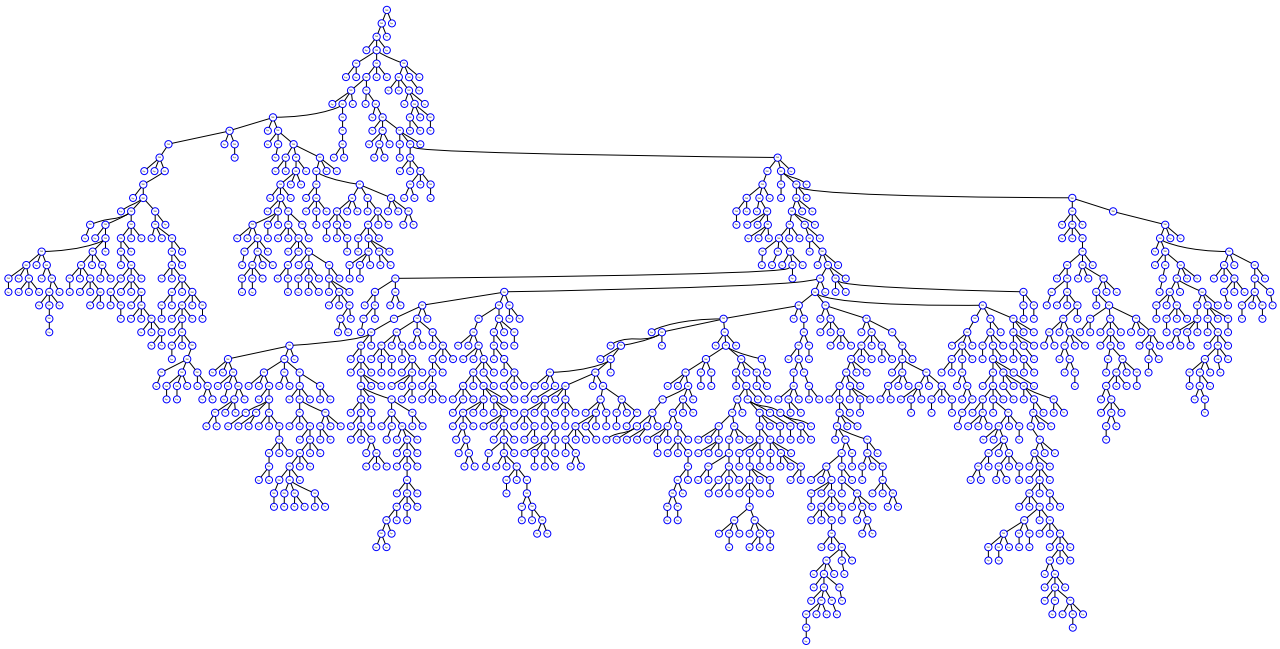


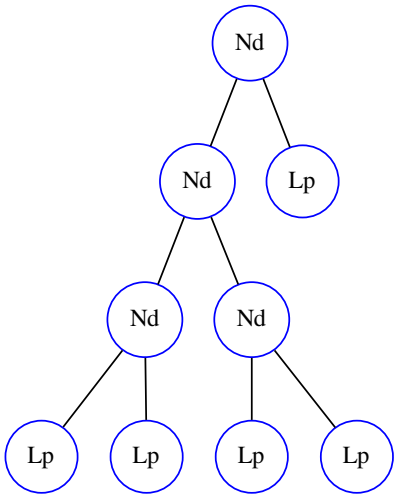
FIGURE 7 – Arbre Ternaire n=600



Arbre ternaire

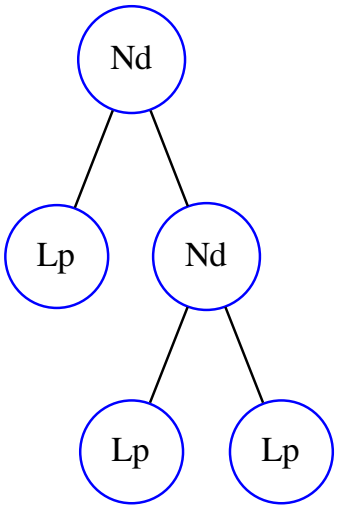
Représentation des arbres binaires et ternaires (Systèmes de parenthèse) Un arbre est constitué d'une racine et des sous-arbre, en renferment chaque arbre dans une parenthèse l'arbre final obtenu est alors représenté par un ensemble de n paires de parenthèses. Une méthode de procéder est d'afficher une parenthèse gauche suivi de la représentation parenthésée du sous arbre gauche puis celle du sous arbre droit.

FIGURE 8 – binaire n=4



Exemple d'affichage parenthésé Son affichage parenthésé est :((((())(()()))())

FIGURE 9 – n=2

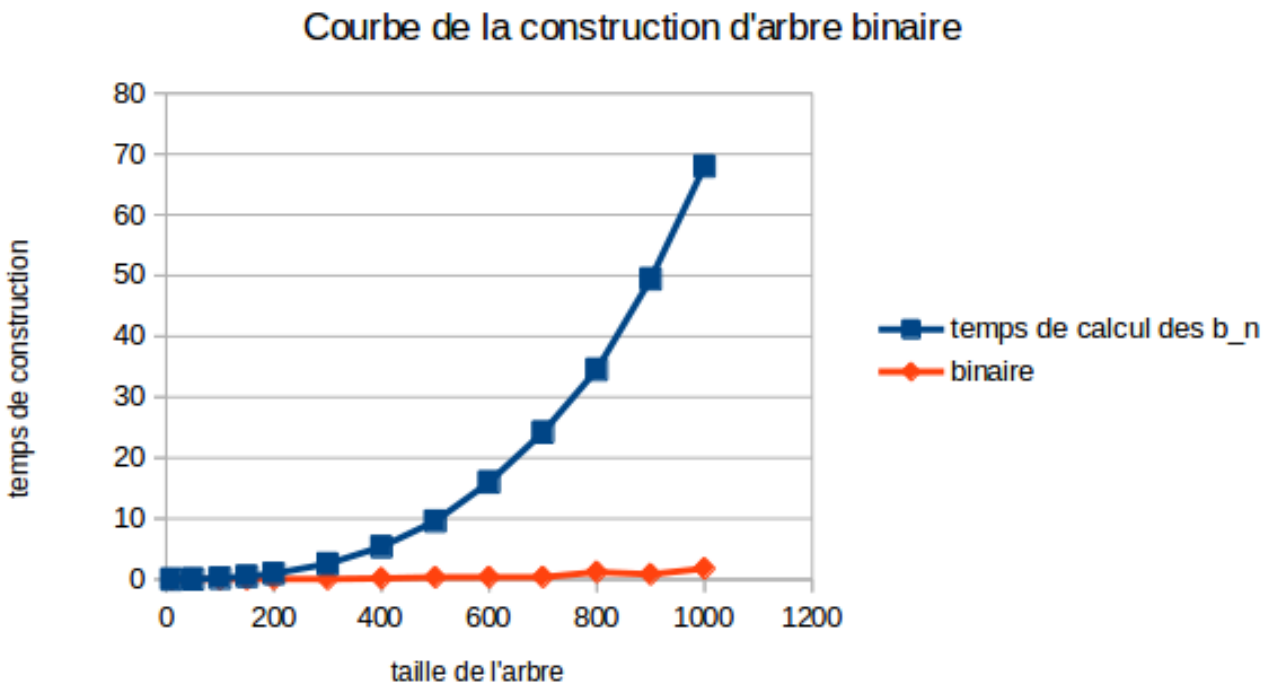


Son affichage parenthésé est : (()(()())).

8 Résultat expérimental

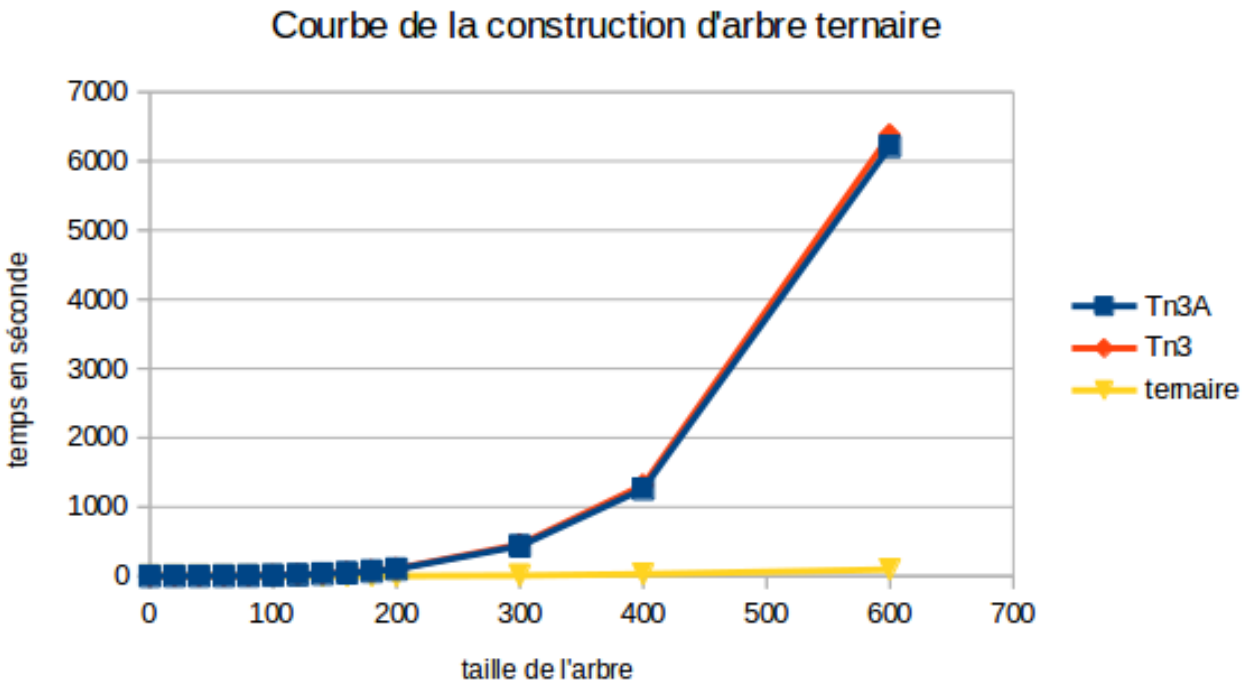
Dans cette partie nous présentons des courbes représentant le temps de la construction des arbres par rapport à leurs tailles.

FIGURE 10 – Courbe des arbres binaires



La figure 10 montre le temps de la construction d'arbres binaires par rapport à leur tailles. L'expérience est effectuée sur des tailles variables de l'arbre [10..1000]. La courbe bleu correspond à l'algorithme 1 (calcul des b_n) et la rouge à l'algorithme 3(tirage d'arbre), on peut voir que la phase de calcul des b_n est la plus coûteuse.

FIGURE 11 – Courbe des arbres ternaires



La figure 11 montre la variation du temps de calcul des t_n la courbe bleu correspond à l'algorithme 5(version amélioré), le rouge à l'algorithme 4 et le jaune à l'algorithme 6 (tirage d'arbre). Notre expérience est portée sur des tailles de 0 a 600.

9 Conclusion

Dans ce projet, on s'est intéressé à l'implémentation en Ocaml d'un outil de génération aléatoire des arbres binaires et ternaires, le projet a pu être mené à son terme. L'outil final est donc capable de générer aléatoirement des arbres binaires et ternaires d'une taille donné. On a pu voir les rôles des séries génératrices dans la génération aléatoire des structures ainsi que la méthode symbolique dont le rôle est de transformer la spécification en une équation fonctionnelle vérifier par la fonction génératrice.

10 Bibliographie

Références

[SF13] Robert SEDGEWICK et Philippe FLAJOLET. *An introduction to the analysis of algorithms*. Addison-Wesley, 2013.