Analyse d'Algorithmes et Génération Aléatoire (5I550)

M2 STL 2016/2017

Les Arbres et l'Étiquetage : Analyse et Génération Aléatoire

Auteurs
Morane Gruenpeter
Guillaume Hivert
Thibault Krebs

Responsable :
Antoine Genitrini

Version 0.1 du
26 janvier 2017

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 Résumé

Le sujet de ce rapport est l'étude et l'implémentation d'un algorithme de génération aléatoire d'arbres binaires et d'arbres généraux étiquetés croissants. Nous nous sommes basés sur la thèse de N.Rolin [1] et le livre de Flajolet et Sedgewick [2] présentés dans le cours d'Antoine Genitrini, Analyse d'Algorithmes et Génération Aléatoire. Premièrement nous allons introduire quelques définitions. Dans le deuxième chapitre nous traitons les structures de base, les arbres binaires étiquetés. Puis dans le troisième chapitre nous analysons les structures complexes, les arbres généraux. Dans le quatrième chapitre nous allons présenter l'algorithme et les expérimentations avec notre implémentation en Ruby.

1.2 Définitions

Arbres binaires complets étiquetés sont des arbre binaires dont les nœuds ont soit 0, soit 2 fils. Les nœuds ayant deux fils sont appelés les nœuds internes, ces nœuds ont une étiquette entre 1 et la taille de l'arbre.

Arbres étiquetés croissants sont des arbres où l'étiquetage des nœuds internes de la racine vers une feuille est une suite de nombre croissant.

Arbres généraux sont des arbres ayant des nœuds où le nombre de fils n'est pas limité à deux.

Arbres généraux étiquetés croissants reprennent toutes les définitions ci-dessus, ce sont des arbres ayant des nœuds où le nombre de fils n'est pas limité à deux et ces nœuds sont étiquetés de manière croissante de la racine aux feuilles.

Chapitre 2

Les notions de bases : l'énumération d'arbres croissants

2.1 Fonction génératrice exponentielle

1. La fonction génératrice exponentielle associé à la suite $(A_n)_n \in \mathsf{N}$

$$A(z) = \sum_{n \geq 0}^{\infty} A_n \frac{z^n}{n!}$$

2. Une classe combinatoire d'objets ayant la fonction génératrice ordinaire $A(z) = \frac{1}{1-z}$: est la classe des arbres généraux de profondeur 1.
3. Une classe combinatoire d'objets ayant la fonction génératrice exponentielle $A(z) = \frac{1}{1-z}$: est la classe du nombre de permutation, par exemple le jeu de carte.
4. La représentation graphique des arbres binaires complets étiquetés de taille 0,1 et 2 :

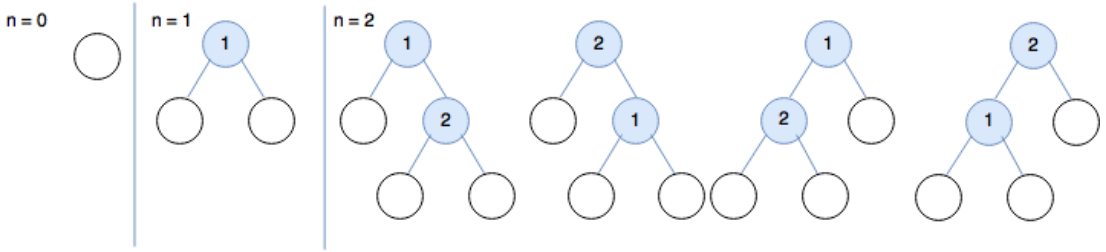

FIGURE 2.1 – Arbres binaires étiquetés

5. Le nombre d'arbres de taille 3 :

$$A_3 = 3! \cdot 5 = 30$$

6. Ci-dessous l'équation de récurrence vérifiée par B_n le nombre d'arbres de taille n avec $b_0 = 1$:

$$B_n = \sum_{0 \leq k \leq n-1} b_k \cdot b_{n-k-1}$$

Nous savons que B_{n-k} a la forme suivante :

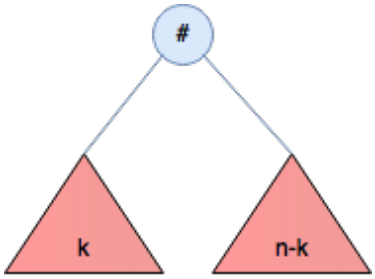

FIGURE 2.2 – Spécification sur B

$(\# = k)$ nous devons placer l'étiquette $n + 1$ aléatoirement. Dans la figure ci-dessous où $A = B_k$ (A contient les étiquettes de 1 à k) et $A' = B_{n-k}$ (A' contient les étiquettes de 1 à n-k-1).

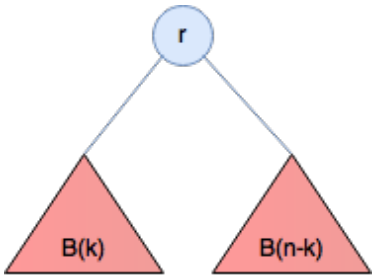

FIGURE 2.3 – L'étiquetage de B

Pour le choix de l'étiquette de la racine, nous avons n choix : { 1, 2, ..., r-1, r, r+1, ..., n-1, n }, Puis il nous reste n-1 choix à redistribuer dans les sous-arbres gauche et droit. Pour les étiquettes du sous-arbre gauche nous avons :

$$\#choix \binom{n}{k}$$

Connaissant la relation d'ordre entre les entiers, nous allons pouvoir ordonner les étiquettes et les répartir entre le sous-arbre gauche et le sous-arbre droit pour obtenir $n + 1$ étiquettes différentes. Ainsi on déduit la formule suivante :

$$B_{n+1} = \sum_{k=0}^{n} (n+1) \binom{n}{k} B_k B_{n-k-1}$$

7. Ci-dessous une spésification vérifié par B :

$$B = \epsilon + ZB \cdot B$$

8. L'équation fonctionnelle par la méthode symbolique vérifiée par $B(z)$ (la série génératrice exponentielle) :

$$B(z) = 1 + zB^2(z)$$

$$\begin{aligned} B_{n+1} &= (n+1)![z^{n+1}]B(z) \\ &= (n+1)![z^{n+1}](z_n B^2) \\ &= (n+1)![z^n]B^2(z) \\ &= (n+1)! \sum_{k=0}^{n} \binom{n}{k} \frac{B_k - B_{n-k-1}}{n!} \\ &= \sum_{k=0}^{n} (n+1) \binom{n}{k} B_k B_{n-k-1} \end{aligned}$$

2.2 Produit avec contrainte de croissance

9. Les classes A_g et A_d contiennent des structures étiquetées. La relation suivante s'appelle Boxed product, celle-ci fait référence au produit du sous-ensemble situé à droite où se trouve le plus petits éléments :

$$A = A_g \square^\star A_d$$

La relation signifie, dans ce cas là, que la plus petite étiquette appartient à A_g.

10. La classe A est la classe des arbres binaires complets étiquetés où la plus petite étiquette se situe dans le sous-arbre droit.
Rappel sur B :

$$B = \epsilon + Z \cdot B \cdot B$$

La spécification de A est :

$$A = \epsilon + (ZB) \star \square B$$

11. L'équation intégrale vérifiée par A_z est

$$A(z) = 1 + \int_0^z (tB(t))B'(t)\,\mathrm{d}t$$

12. Nous utilisons la formule de Leibeniz afin d'obtenir l'intégrale :

$$A'(z) = z \cdot B(z) \cdot B'(z)$$

13. La formule suivante démontre la récurrence sur les nombres d'arbres de A :

$$A_{n+1} = (n+1)![z^{n+1}]A(z)$$

$$A_{n+1} = (n+1)![z^n]\frac{A'(z)}{n+1}$$

Car nous avons $[z^n]A(z) = [z^{n-1}]\frac{A'(z)}{n}$

$$\begin{aligned} A_{n+1} &= n![z^n]A'(z) \\ &= n![z^n](zB(z)B'(z)) \\ &= n![z^{n-1}](B(z)B'(z)) \\ &= n!\sum_{k=0}^{n-1}\binom{n-1}{k}\frac{b_k b_{n-k}}{(n-1)!} \\ &= n\sum_{k=0}^{n-1}\binom{n-1}{k}b_k b_{n-k} \end{aligned}$$

14. Les arbres binaires complets croissants, notés C, contiennent des étiquettes de la racine à la feuille d'ordre strictement croissant. La spécification de C est :

$$C = \epsilon + Z\square \star C \cdot C$$

En utilisant la formule, on obtient :

$$C(z) = 1 + \int_0^z C^2(t)\,\mathrm{d}t$$

avec $C'(z) = C^2(z)$ nous savons aussi que :

$$C'(z) = \sum_{n\in N}\sum_{k=0}^{n}\binom{n}{k}b_k b_{n-k}z^n$$

donc,

$$\begin{aligned} C_n &= n![z^n]C(z) \\ &= n![z^n - 1]\frac{C'(z)}{n} \\ &= n!\sum_{k=0}^{n-1}\binom{n-1}{k}\frac{b_k b_{n-k-1}}{n!} \\ &= \sum_{k=0}^{n-1}\binom{n-1}{k} b_k b_{n-k-1} \end{aligned}$$

Chapitre 3

La structure complexe : l'arbre croissant

3.1 Arbres généraux croissants

15. La classe G est la classe des arbres généraux. La spécification vérifiée par G est :

$$G = Z\Box \star SEQG,$$

où SEQ est l'opérateur Séquence, avec lequel les éléments de la classe G sont les successions d'un nombre appartennant à G. Cette spécifcation indique qu'un noeud de G est composé d'une séquence fini de fils étant aussi eux même des noeuds de G.De plus, chaque noeud fils doit posseder un entier supérieur à celui du père.

La taille d'une feuille est de 1 car G ne peut pas avoir une taille de 0 car cela entrainerait des séquences de taille infinie.

16. La récurrence sur les nombres des arbres de G :

$$G(z) = 1 + \int_0^z (\frac{1}{1 - G(t)})\mathrm{d}t$$

En utilisant la règle de Leibniz : $G'(z) = \frac{1}{1-G(z)}$ on obtient :

$$\begin{aligned}
g_{n+1} &= n![z^n]\frac{1}{1-G(z)} \\
&= n![z^n]\sum_{k\geq 0} G^k(z) \\
&= n![z^n]\sum_{k=1}(\sum_{k>0}\frac{g_n z^n}{n!})^k \\
&= n!\sum_{k=1}^{n}[z^n]\sum_{r_1,\cdots,r_k} g^{r_1}(z)g^{r_2}(z)g^{r_3}(z)\cdots \\
&= n!\sum_{k=1}^{n}\sum_{r_1,\cdots,r_k}\binom{n}{r_1 r_2 ... r_n}\frac{G^{r_1}G^{r_2}G^{r_3}\cdots}{n!} \\
&= \sum_{k=1}^{n}\sum_{r_1,\cdots,r_k}\binom{n}{r_1 r_2 ... r_n} G^{r_1}G^{r_2}G^{r_3}\cdots
\end{aligned}$$

Chapitre 4

Algorithme et expérimentation

4.1 Algorithme de génération d'arbre généraux étiquetés croissants

l'algorithme 1 présente la première étape de la génération aléatoire sans l'étiquetage. Ensuite la deuxième étape est l'étiquetage de la structure. Finalement l'algorithme 2 retourne un **arbre général étiquté croissant**.

Algorithm 1 Algorithme génération aléatoire d'arbres généraux

1: **procedure** TREE : GENSILOUHETTE(n) ▷ la taille de l'arbre
2: $r \leftarrow random([0, G[n])$
3: $k \leftarrow choisirArbreTailleN(r)$
4: $tailleChildren \leftarrow choisirForme(k)$
5: $children \leftarrow map(getSilouhette, taille_children)$
6: **return** *tree* ▷ retourne la structure

Algorithm 2 Algorithme génération aléatoire d'arbres généraux étiquetés croissants

1: **procedure** TREE : GEN(n) ▷ la taille de l'arbre
2: $t \leftarrow genSilouhette(n)$ ▷ génère structure
3: $t \leftarrow genLabels(n)$ ▷ étiquetage
4: **return** t

4.2 Expérimentations et statistiques

Afin d'obtenir des statistiques significatives, des tests de génération d'arbre en fonction de leur taille ont été effectués. 15 arbres de chaque taille ont été générés, de 2 à 25, puis la moyenne de différentes données a été calculé : le temps d'exécution de l'algorithme, la profondeur maximale de l'arbre, ainsi que la profondeur moyenne des noeuds de la structure. En procédant ainsi, il est possible d'obtenir les courbes suivantes.

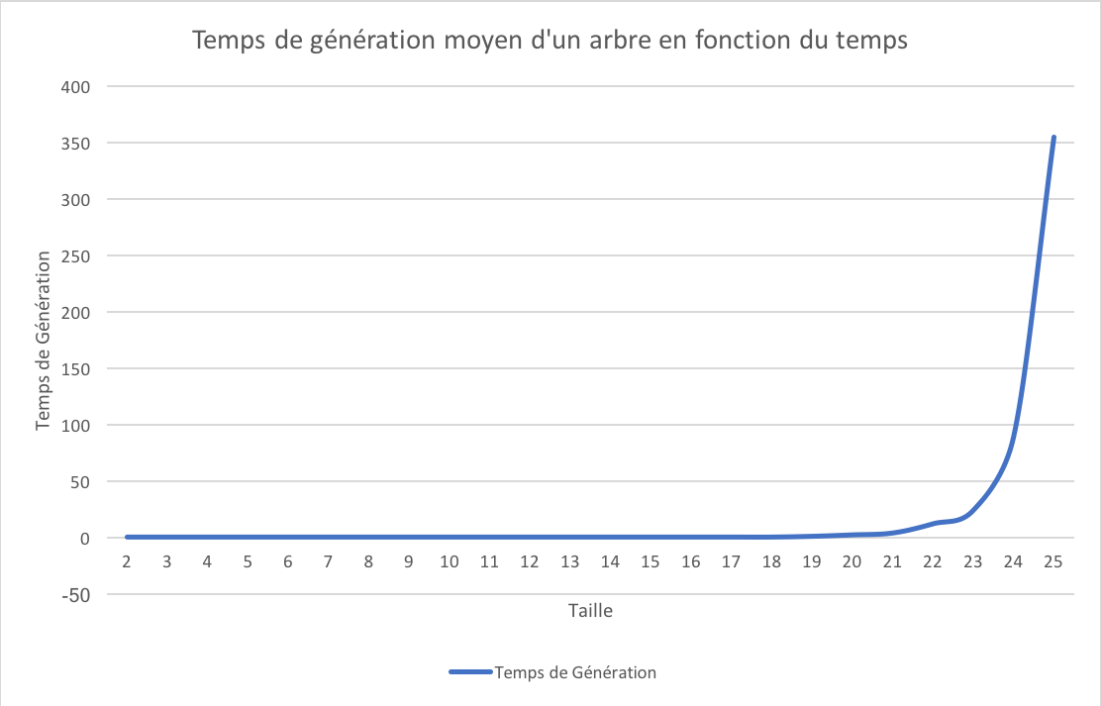

FIGURE 4.1 – Temps de génération moyen d'un arbre en fonction du temps

Cette première courbe donne le temps d'exécution de l'algorithme en fonction de la taille de l'arbre. Il est possible d'en déduire que la génération d'un arbre de petite taille (inférieur ou égal à 20 noeuds au total) est instantané — ou quasiment instantané. À partir d'une taille de 21 noeuds, le temps d'exécution commence à croître, avant d'atteindre une durée exponentielle. Cela s'explique par la redondance de nombreux calculs, particulièrement pour les arbres de petites tailles. Par exemple, les compositions pour un arbre de taille 5 sont calculés de nombreuses fois pour une arbre de taille 21, alors qu'un calcul unique serait bien plus performant à l'usage.

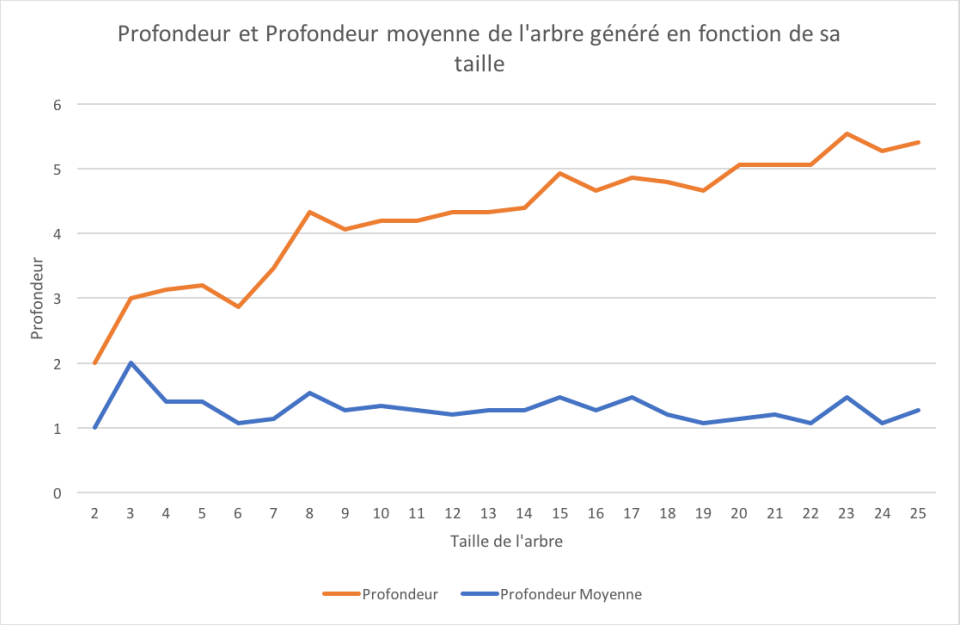

FIGURE 4.2 – Profondeur et profondeur moyenne de l'arbre généré en fonction de sa taille

Cette seconde courbe donne la profondeur maximale et la profondeur moyenne d'un arbre. La profondeur moyenne permet d'estimer la silhouette de l'arbre. Puisque celle-ci est très petite, il est possible d'en déduire que l'arbre est bien plus large que dense, et est donc homogène sur l'ensemble des silhouettes d'un arbre. En effet, il existe bien plus de silhouettes d'un arbre à 5 fils, que de silhouettes d'un arbre à 1 ou 2 fils. La probabilité d'obtenir une faible profondeur avec une largeur importante est donc concordante avec la courbe.

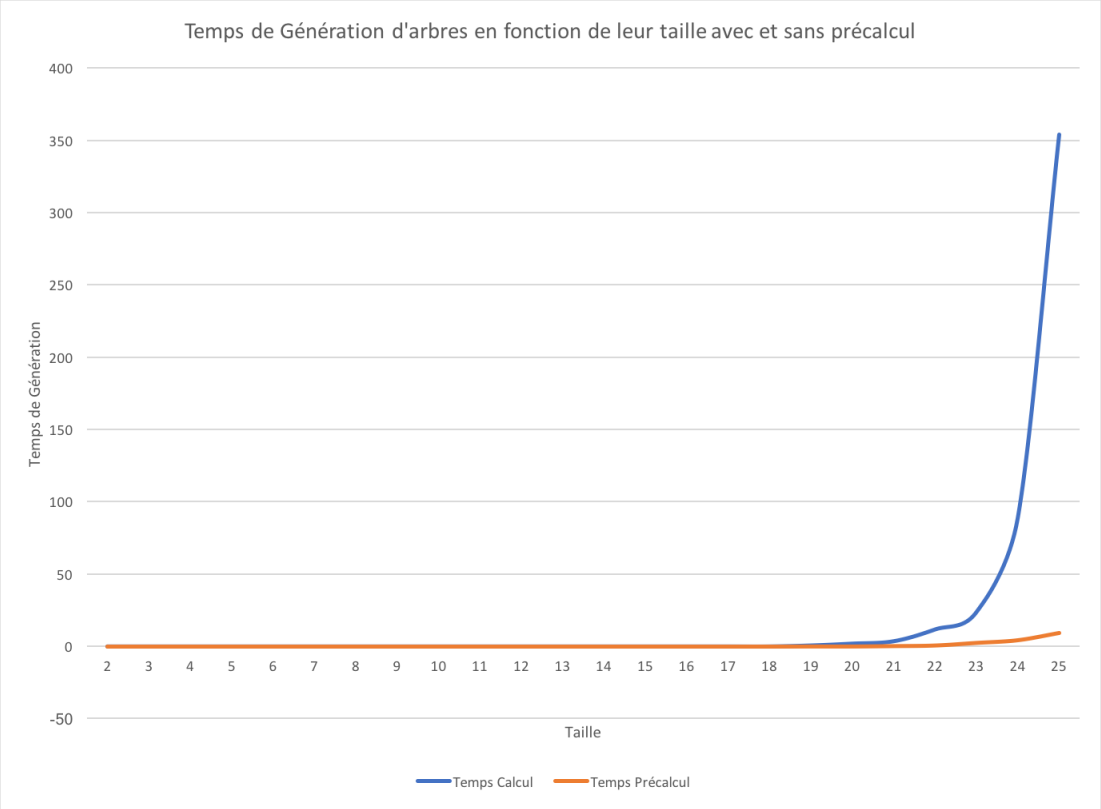

FIGURE 4.3 – Temps de génération d'arbres en fonction de leur taille avec ou sans précalcul

Enfin, ces deux courbes illustrent la différence entre le calcul brut de chaque arbre comme illustré dans la 1e courbe, ainsi que la génération d'arbre avec une première période de précalcul pour obtenir les données manipulées par l'algorithme. De cette façon, les compositions pour un arbre de taille 5 — par exemple — ne sont calculés qu'une seule et unique fois. Il est donc bien plus rapide de chercher en RAM l'information que de recalculer le tout à chaque itération. La génération d'un arbre avec précalcul est donc nettement plus rapide que sans, si une première période de calcul est abordable pour l'utilisateur.

Chapitre 5

Conclusion

La génération d'arbres de grandes tailles se heurtent rapidement à des limitations matérielles. A partir d'une taille supérieure à 20, il faut renoncer soit à obtenir un arbre rapidement, soit à une consommation modérée de RAM. Dans le premier cas, une génération d'arbre de grande taille prendra de plusieurs minutes ou heures à plusieurs jours, le temps demandé étant exponentiel. Il est toutefois possible de réaliser des passes de précalcul pour obtenir les valeurs nécessaires à la génération d'arbres (notamment les compositions, les allures des arbres, les factoriels et multinomials nécessaires). La consomation de RAM devient alors rapidement conséquente, mais permet d'optimiser le temps de calcul en sacrifiant une première période de calcul préalable. Avec une architecture adéquate qui nécessite ensuite de nombreuses générations d'arbres, il est donc intéressant de précalculer. En revanche, sur des petites instances, la différence est invisible.

Bibliographie

[1] N. Rolin. De l'usage des opérateurs en combinatoire : construction,analyse et génération aléatoire. Master's thesis, Paris 13 | Ecole doctorale Galilée, Octobre 2016.

[2] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, New York, NY, USA, 1 edition, 2009.

Annexe A

Annexe

A.1 Affichage d'un exemple d'arbre général généré