ELSEVIER

Discrete Mathematics 256 (2002) 791–800

DISCRETE MATHEMATICS

www.elsevier.com/locate/disc

Tirage à pile ou face de mots de Fibonacci

Jean Guy Penaud*, Olivier Roques

Lab. Bordelais de Recherche en Informatique (LaBRI), Université Bordeaux 1, 351 cours de la Libération, F-33405 Talence Cedex, France

Received 10 October 2000; received in revised form 5 February 2001; accepted 28 May 2001

Abstract

In this paper, we give a semi-algorithm to sample uniformly at random from the Fibonacci language $\mathbf{L} = \varepsilon + \mathbf{L}a + \mathbf{L}bb$. It needs a uniform bit generator and the average complexity is linear. We show that we can transform this semi-algorithm to a true algorithm still with a linear complexity. Finally, we discuss a generalization to a class of regular languages.

Résumé

Dans cet article, nous donnons un semi-algorithme qui permet d'engendrer de façon aléatoire et rigoureusement uniforme un mot du langage de Fibonacci, $\mathbf{L} = \varepsilon + \mathbf{L}a + \mathbf{L}bb$. Il utilise un générateur uniforme de bits et la complexité moyenne en temps et en espace est linéaire. Nous montrons comment transformer ce semi-algorithme en algorithme à terminaison assurée, tout en gardant une complexité linéaire en moyenne. Enfin, on donne une famille de langages rationnels pour lesquels notre méthode s'applique.

Keywords: Uniform random generation; Regular language

0. Introduction

Les nombres de Fibonacci 0,1,1,2,3,5,8,13,..., (cf. Section 2) énumèrent de nombreux objets combinatoires, allant des couplages de n points alignés jusqu'a certaines familles de polyominos [2]. Parmi ces objets, le langage de Fibonacci est un des exemples les plus élémentaires. Il est formé des mots sur l'alphabet à 2 lettres $X = \{a, b\}$ obtenus en concaténant de toutes les façons possibles les facteurs a et bb (facteurs

* Corresponding author.
E-mail addresses: penaud@labri.u-bordeaux.fr (J.G. Penaud), roques@labri.u-bordeaux.fr (O. Roques).

0012-365X/02/$ - see front matter
PII: S0012-365X(02)00348-5

constitutifs que nous appelerons *motifs* dans la suite). C'est un langage rationnel d'équation (cf. Section 2) $\mathbf{L} = \varepsilon + \mathbf{L}a + \mathbf{L}bb$.

La génération aléatoire *uniforme* de mots d'un langage rationnel, uniforme par rapport à la longueur des mots engendrés, est un problème que l'on sait résoudre par une méthode récursive, en considérant ces langages comme cas particuliers de langages algébriques [6] ou de structures décomposables [5]. Cependant, la mise en œuvre de cette méthode conduit à manipuler des nombres *exponentiels* en la longueur du mot que l'on souhaite engendrer, et donc cela ne facilite pas la génération des mots de très grande taille. Signalons toutefois que des progrès sensibles ont permis récemment (voir [3,4]), en utilisant l'arithmétique flottante, d'atteindre un complexité bit à bit quasilinéaire ($O(n^{1+\varepsilon})$), après un précalcul en $O(n^{2+\varepsilon})$ (réduit à $O(n^{1+\varepsilon})$ dans le cas algébrique).

Dans cet article, nous proposons a contrario une méthode à la fois légère, paresseuse, mais juste, précise et efficace.

Légère car elle ne nécessite ni précalcul, ni aucune opération arithmétique sur des grands nombres. Paresseuse car aucun calcul n'est effectué avant sa stricte nécessité. Juste, car la génération se fait sans biais, et fournit un échantillon avec une probabilité rigoureusement uniforme. Précise car cette méthode ne met en jeu que des calculs bits à bits, sans utiliser d'arithmétique flottante. Efficace car la complexité moyenne en bits tirés est de l'ordre de $O(n)$, et elle permet réellement selon l'algorithme 1 ci-dessous, d'engendrer en quelques secondes un mot de plusieurs millions de lettres du langage de Fibonacci.

Le principe de la méthode est double:

- d'une part elle est fondée sur l'utilisation d'un semi-algorithme à rejet [1,9,10], qui construit le mot à partir du mot vide en concaténant les motifs les uns après les autres, motifs choisis avec les probabilités appropriées, en gardant la possibilité de rejeter le mot et de recommencer lorsque celui-ci n'est pas correct;
- d'autre part, pour décider quel motif choisir entre "a" et "bb" pour le concaténer au mot courant en cours de construction, on effectue un tirage paresseux : il consiste à tirer juste le nombre de bits nécessaires du développement binaire d'un réel aléatoire pour pouvoir décider du choix. Les mots engendrés le sont de façon rigoureusement uniforme sous l'hypothèse que l'on sache également tirer des bits de façon uniforme, comme présenté dans le très bel article fondateur de Knuth et Yao, [7].

La première section décrit la méthode paresseuse du tirage à pile ou face. La deuxième montre comment appliquer cette méthode à la construction d'un mot de Fibonacci. Puis la troisième section présente la complexité moyenne en nombre de bits aléatoires pour engendrer un mot de longueur n. Nous montrons également comment on peut transformer notre semi-algorithme en algorithme, tout en gardant une complexité linéaire. Dans la dernière section, nous discutons les extensions possibles de notre algorithme à une famille de langages rationnels, les *étoiles de motifs*.

1. Tirage paresseux

L'objectif est de tirer dans un espace d'épreuves E à deux éléments mutuellement exclusifs A et B, l'événement A avec la probabilité donnée p. Dans le cadre de l'arithmétique flottante, la façon usuelle de procéder est de tirer selon la loi continue et uniforme un nombre réel r entre 0 et 1, et de choisir A si $0 \leqslant r \leqslant p$, et B si $p < r \leqslant 1$. Ce cas sort du cadre que l'on s'est fixé.

Un autre approche, utilisant un flot de bits uniformes, consisterait à tirer un nombre réel r bit par bit, à le comparer, après chaque bit tiré, à la probabilité cible p supposée parfaitement connue, et s'arrêter dès que l'on serait assuré que r diffère (en plus grand ou plus petit) de p. Mais cela ne pourrait convenir que si p a une écriture binaire finie, ce qui n'est évidemment pas le cas dans le tirage des mots de Fibonacci, oú la probabilité fait intervenir (cf. Section 2) le *nombre d'or* $\phi = (1 + \sqrt{5})/2$.

Nous présentons ici une méthode qui suppose que la probabilité cible p est donnée comme racine simple d'un polynôme P de degré d à coefficients entiers, supposé, pour simplifier, monotone dans l'intervalle $[0, 1]$.

1.1. Méthode

On tire le nombre r bit par bit. Le tirage est paresseux car on s'arrête de tirer dès que l'on peut déterminer si, quelle que soit la suite des bits tirés ultérieurement, la valeur globale du nombre r construit sera définitivement inférieure ou supérieure à la probabilité cible p, rendant donc superflu de continuer. Un tel bit r_i sera dit *bit décisif*, et on montrera que le nombre moyen de bits *utiles*, c'est-à-dire le nombre de bits tirés jusqu'au bit décisif est exactement 2.

La comparaison du nombre tiré r avec la probabilité cible p se fera en évaluant le signe du polynôme définissant la probabilité, pour la valeur de la séquence tirée. Ce calcul se fait en binaire avec une complexité (évaluée en multiplications bit à bit) finie.

1.2. Bit décisif

Soit r un nombre réel r compris entre 0 et 1; on note r_i le ième bit de l'écriture binaire de sa partie fractionnaire. Inversement, étant donné un mot w (possiblement infini) sur l'alphabet $\{0, 1\}$, on note w_i la ième lettre de w et $\mathrm{val}(w)$ le nombre qui admet w comme développement fractionnaire, c'est-à-dire $\mathrm{val}(w) = \sum_{i \geqslant 1} w_i (1/2^i)$.

Avec ces notations nous pouvons préciser la notion de bit décisif.

Définition 1. Soient deux nombres p et r dans [0,1]. On dira que le bit r_i du développement de r est décisif si i est le rang du premier bit qui diffère dans les développements de p et de r.

Plus précisement, le bit décisif vérifie,

1. $1 \leqslant j < i \Rightarrow r_j = p_j$,
2. $(r_i = 1 \wedge p_i = 0) \vee (r_i = 0 \wedge p_i = 1)$.

Ainsi la connaissance des i premiers bits permet de décider, quelle que soit la continuation du développement de r, si $r > p$ ou si $r \leqslant p$.

Le nombre p étant donné, la probabilité que le bit r_i soit égal à p_i est $\frac{1}{2}$, tout comme la probabilité qu'il en diffère. La probabilité que le bit décisif soit le bit de rang i est donc $1/(2^{i-1}) * \frac{1}{2} = 1/2^i$. D'où la propriété,

Propriété 2. *Si p est donné, et si r est tiré uniformément bit par bit, le rang moyen du bit décisif r_i est égal à $\sum_{i \geqslant 1} i/2^i = 2$.*

Plaçons nous maintenant dans le cadre spécifique de l'article, c'est-à-dire le développement binaire de la probabilité cible p est inconnu, mais p est déterminé uniquement comme racine d'un polynôme P donné, racine supposée unique dans l'intervalle $[0, 1]$.

Les propriétés suivantes montrent qu'il est toujours possible de déterminer efficacement le bit décisif de r.

1.3. Complexité bit à bit

Une conséquence directe de la définition du bit décisif conduit à la propriété suivante, oú il est intéressant de noter que, dans les deux cas, la valeur de la même chaîne de bits est comparée à p.

Propriété 3. *Sous les mêmes hypothèses que la Propriété* 2, *on a*,

1. $r_i = 1 \Rightarrow (p_i = 0 \Leftrightarrow (\mathrm{val}(r_1 r_2 \ldots r_{i-1} 1) > p)$,
2. $r_i = 0 \Rightarrow (p_i = 1 \Leftrightarrow (\mathrm{val}(r_1 r_2 \ldots r_{i-1} 1) \leqslant p)$.

Preuve. Pour le point 1 c'est évident car précisément $(r_i = 1 \wedge p_i = 0) \Leftrightarrow \mathrm{val}(r_1 r_2 \ldots r_i) > p$.

De même pour le point 2 c'est vrai *a fortiori* car $(\mathrm{val}(r_1 r_2 \ldots r_{i-1} 1) \leqslant p) \Leftrightarrow p_i = 1$. □

Après chaque tirage d'un bit r_i, il convient de vérifier s'il est décisif. Pour cela il suffit, d'après la Propriété 3, et sous les hypothèses choisies pour le polynôme P dont p est racine (à savoir p racine simple unique dans l'intervalle [0,1]), de calculer le signe de $P(\mathrm{val}(r_1 \ldots r_{i-1} 1))$.

La complexité en opération bit à bit de l'évaluation du signe de ce polynôme dépend polynomialement du rang i du bit testé, ce qui donne une complexité moyenne constante pour cette évaluation, en raison de la décroissance exponentielle de la probabilité du nombre de bits tirés.

Cependant on peut aisément donner une majoration plus précise de cette complexité. Appelons d le degré du polynôme P, et l, la longueur maximale des chaînes de bits codant les coefficients de P. On a la proposition suivante,

Propriété 4. *A l'étape i le nombre d'exécutions d'opérations binaires est majoré par $i^2 d((d-1)/2) + id(l+d) + d(l + (d+1)/2)$. Le coût moyen est donc majoré par $6d^2 - 2d + 4dl$.*

Preuve. Selon le schéma de Horner, le polynôme P peut s'écrire,
$P(x) = ((a_d x + a_{d-1})x + a_{d-2})x + \cdots + a_0$.

Or une multiplication de nombres binaires a pour complexité bit à bit le produit des longueurs, et le résultat a pour longueur la somme des longueurs. Pour une somme, la complexité et la longueur du résultat sont toutes deux majorées par la longueur du plus long terme augmentée de 1. D'où la première expression de la complexité, en évaluant ces quantités dans l'ordre du schéma de Horner.

L'expression du coût moyen s'en déduit immédiatement car $\sum_{i \geqslant 1} i^2(1/2^i) = 6$. □

1.4. Généralisation

Il est clair que le procédé s'étend au cas où la probabilité cible p est racine multiple (on prend la dérivée adéquate), ou également au cas où l'intervalle d'unicité de racine et de monotonie du polynôme, déterminé par deux rationnels α et β, $\alpha < \beta$, est strictement inclus dans [0,1]. De même, la technique du résultant de deux polynômes permet de traiter le cas où l'on a plus de deux événements mutuellement exclusifs à tirer. Il suffit d'exprimer la seconde (voire la troisième ...) probabilité cible sous forme polynomiale en fonction de la première, (voire de la précédente) ce qui constitue le nouveau polynôme du résultant, le premier restant le polynôme P dont p est racine.

2. Génération aléatoire du langage de Fibonacci

Il est bien connu que les nombres de Fibonacci [11] sont définis par les conditions initiales et la récurrence linéaire suivantes,

$f_0 = 0, f_1 = 1$, et pour $n \geqslant 2, f_n = f_{n-1} + f_{n-2}$.

Ils s'expriment en fonction du *nombre d'or* $\phi = (1 + \sqrt{5})/2$ et de l'opposé de son inverse,

$\hat{\phi} = -1/\phi = (1 - \sqrt{5})/2$, et l'on a, $f_n = (\sqrt{5}/5)(\phi^n - \hat{\phi}^n)$.

Le *langage de Fibonacci* **L**, défini dans l'introduction, est un langage rationel sur l'alphabet $\{a, b\}$, de grammaire,

$$\mathbf{L} = \varepsilon + \mathbf{L}a + \mathbf{L}bb. \tag{1}$$

On obtient ainsi un ensemble infini dont les premiers mots, rangés par longueur croissante, sont,

$$\mathbf{L} = \{\varepsilon, a, aa, bb, aaa, abb, bba, \ldots\}.$$

On note $\mathbf{L}_n$ l'ensemble des mots de Fibonacci de longueur n et L_n leur nombre. Ce dernier est clairement le nombre de Fibonacci f_{n+1}.

Pour tirer un mot de Fibonacci de façon équiprobable parmi l'ensemble des mots de longueur n, on construit le mot, motif par motif, en choisissant de concaténer au

mot courant le motif a avec la probabilité p et le motif bb avec la probabilité q. On doit avoir d'une part $q = p^2$ pour respecter l'uniformité selon la longueur, et d'autre part $p+q=1$ car ce sont les deux seuls motifs possibles. D'où l'équation polynômiale $1 - p - p^2 = 0$ dont $p = \phi^{-1}$ est racine dans l'intervalle $[0,1]$.

Par ce procédé, chaque mot de L de longueur n est engendré avec la probabilité ϕ^{-n}. Si la longueur du mot obtenu est $n+1$ (le mot appartient alors à $\mathbf{L}_{n-1}.bb$), on le rejette et on recommence le procédé. Ceci est illustré par l'algorithme 1. Dans la suite, on appellera *essai*, le tirage par cet algorithme d'un mot de $\mathbf{L}_n \cup \mathbf{L}_{n-1}.bb$, (ce qui revient à supprimer les lignes 5–7 de l'algorithme 1).

Algorithme 1. Tirage uniforme d'un mot de Fibonacci de longueur n (semi-algorithme)

Entrée un entier n
Sortie un mot de Fibonacci aléatoire et uniforme de longueur n
1: $w \leftarrow \varepsilon$
2: **tant que** $|w| < n$ **faire**
3: $l \leftarrow$ Motif_aléatoire
4: $w \leftarrow wl$
5: **si** $|w| = n+1$ **alors**
6: $w \leftarrow \varepsilon$
7: **fin si**
8: **fin tant que**

A chaque étape de la construction du mot on effectue un tirage paresseux afin de choisir entre les deux motifs, et ceci sans connaître le développement binaire de ϕ^{-1}, comme expliqué dans la Section 1.

L'algorithme 2 met en oeuvre ce procédé. Pour comparer la valeur de la suite binaire courante construite par l'algorithme avec ϕ^{-1}, il suffit d'évaluer le signe du polynôme $P(t) = 1 - t - t^2$. Ce polynôme, décroissant sur $[0,1]$, vérifie bien les hypothèses de validité de la méthode exposée Section 1.

3. Complexité

On évalue la complexité de l'algorithme 1, en nombre d'appels à la fonction Motif_aléatoire (algorithme 2) nécessaires à l'obtention d'un mot de Fibonacci de longueur n (i.e. à la terminaison de l'algorithme 1).

De l'équation (1) du langage de Fibonacci, on déduit la série génératrice de $\mathbf{L}$ selon la longueur

$$\mathbf{L}(x) = \sum_{n \geqslant 0} L_n x^n = \frac{1}{1 - x - x^2} = \frac{1}{(1 - \phi x)(1 - \hat{\phi} x)}.$$

Algorithme 2. Motif_aléatoire

Entrée $P(t)=1-t-t^2$
Sortie le motif a avec probabilité ϕ^{-1} ou le motif bb avec probabilité $1-\phi^{-1}$

```
1:  r ← ε
2:  tant que on n'a atteint le bit décisif faire
3:    z ← un bit aléatoire uniforme
4:    si z = 1 alors
5:      si p(val(r1)) < 0 alors
6:        z est décisif; retourner bb
7:      fin si
8:    sinon {z = 0}
9:      si p(val(r1)) < 0 alors
10:       z est décisif; retourner a
11:     fin si
12:   fin si
13:   r ← rz
14: fin tant que
```

où $\hat{\phi}$ est le pôle de plus petit module de $\mathbf{L}(x)$. On en déduit immédiatement le comportement asymptotique du nombre de mots de Fibonacci de longueur n, $L_n \sim (\sqrt{5}/5)\,\phi^{n+1}$.

La complexité moyenne de l'algorithme 1 s'exprime comme le produit du nombre moyen d'appels à la fonction Motif_aléatoire( ) lors de la génération d'un mot, par le nombre moyen d'essais nécessaires à l'obtention d'un mot de Fibonacci de longueur n.

Il est clair que le nombre d'appels de la fonction Motif_aléatoire, égal au nombre de motifs qui ont servi à construire le mot, est compris entre $\lceil n/2\rceil$ et n, donc sa moyenne également. Pour avoir une valeur plus précise, nous définissons une variable aléatoire X qui compte le nombre de motifs a et de motifs bb lors d'un essai de l'Algorithme 1. La moyenne de X, notée $E(X)$ s'exprime comme la somme du nombre moyen de motifs a et de motifs bb dans les mots de $\mathbf{L}$ de longueur n et du nombre moyen de motifs a et de motifs bb dans les mots de $\mathbf{L}bb$ de longueur $n+1$.

Soit $A_n=\sum_{f\in\mathbf{L}_n}|f|_a+|f|_{bb}$, le nombre total de motifs a et de motifs bb dans les mots de longueur n. D'après (1) $\mathbf{L}_n=\mathbf{L}_{n-1}a+\mathbf{L}_{n-2}bb$. On peut donc écrire

$$\begin{aligned}A_n &= \sum_{f\in\mathbf{L}_{n-1}a}|f|_a+|f|_{bb}+\sum_{f\in\mathbf{L}_{n-2}bb}|f|_a+|f|_{bb}\\ &= \sum_{f\in\mathbf{L}_{n-1}}(|f|_a+1)+|f|_{bb}+\sum_{f\in\mathbf{L}_{n-2}}|f|_a+(|f|_{bb}+1)\\ &= A_{n-1}+A_{n-2}+L_{n-1}+L_{n-2}.\end{aligned}$$

Les valeurs initiales sont $A_0=0$, $A_1=1$, $A_2=3$, $A_3=7$, et puisque $L_n=L_{n-1}+L_{n-2}$, on obtient pour $n\geqslant 4$, l'expression,

$$A_n=2A_{n-1}+A_{n-2}-2A_{n-3}-A_{n-4}.$$

On en déduit alors (voir [11])

$$A_n = [x^n]\,\frac{x+x^2}{(1-x-x^2)^2} \sim \frac{\phi^2}{(1+2\phi^{-1})^2}\,\phi^n n,$$

ce qui donne

$$\begin{aligned} E(X) &= A_n\phi^{-n} + (A_{n-1}+L_{n-1})\phi^{-n-1}, \\ &\sim n\frac{\phi^2+1}{5}. \end{aligned} \tag{2}$$

On calcule maintenant le nombre moyen d'essais nécessaires à l'obtention d'un mot de Fibonacci. Soit Y la variable aléatoire qui compte le nombre moyen d'essais pour obtenir un mot correct. Y suit une loi géométrique avec comme probabilité de succès $L_n\phi^{-n}$. On a immédiatement,

$$E(Y) \sim \sqrt{5}\phi^{-1}. \tag{3}$$

En multipliant (2) par (3) on obtient le nombre moyen de tirages de motifs a ou de motifs bb, ce qui fait l'objet de la proposition suivante.

Proposition 5. *La complexité moyenne $\mathscr{C}_n$ de l'algorithme* 1 *en nombre d'appels de la fonction Motif_aléatoire est*,

$$\mathscr{C}_n \sim n.$$

D'après la Propriété 2, il faut en moyenne 2 bits pour tirer un motif aléatoire. Ainsi, la complexité moyenne de l'algorithme 1 en nombre de bits aléatoires est asymptotiquement égale à $2n$. Si l'on compte les opérations binaires, cette complexité est majorée par $2kn$ où k est la constante calculée dans la Section 1 (Propriété 4), soit $k = 6d^2 - 2d + 4dl$ avec $d = 2$ et $l = 1$ ce qui donne une majoration de 28 opérations bit à bit par appel.

4. Algorithme à terminaison assurée

La complexité dans le pire des cas de l'algorithme 1 est infinie. Ceci est une caractéristique des algorithmes à rejet, et peut ne pas convenir à certaines applications qui requièrent une réponse sûre. Néammoins, puisque l'on connait un algorithme dont le pire des cas est $O(n^2)$ [3], on peut décider de lancer l'algorithme quadratique si le semi-algorithme (algorithme 1) n'est pas terminé au bout de T essais (algorithme 3).

Soit $K(n) = O(n^2)$ la complexité moyenne d'un algorithme fondé sur la méthode récursive (par exemple l'algorithme proposé dans [3]) pour engendrer un mot de

Fibonacci de longueur n. La complexité moyenne de l'algorithme 3 est alors

$$E(X)\sum_{i=1}^{T} i\,\mathrm{Prob}(Y=i)+\mathrm{Prob}(Y>T)(K(n)+TE(X))$$

$$=E(X)\frac{1-(T+1)q^T+Tq^{T+1}}{1-q}+q^T(K(n)+TE(X)).$$

où $q=\phi^{-2}$. En prenant par exemple $T=n$, la complexité de l'algorithme 3 est $O(n)$.

Algorithme 3. Tirage uniforme d'un mot de Fibonacci de longueur n (algorithme)

Entrée un entier n, un entier T
Sortie un mot de Fibonacci aléatoire et uniforme de longueur n

```
1:  w ← ε
2:  i ← 0
3:  tant que i < T et |w| < n faire
4:    l ← Motif_aléatoire
5:    w ← wl
6:    si |w| = n + 1 alors
7:      w ← ε
8:    fin si
9:  fin tant que
10: si ||w|| = n + 1 alors
11:   Lancer un autre algorithme (qui termine).
12: fin si
```

5. Extensions

L'algorithme 1 peut aisément s'étendre aux langages rationnels du type *étoile de motifs*, (voir (Fig. 1), langages qui apparaissent souvent lors de la génération aléatoire de chemins du plan à l'aide d'algorithmes à rejet (cf. [9] ou [8]). Ils s'écrivent,

$$\mathbf{L}=(m_1+m_2+\cdots+m_k)^{\star},$$

où $m_1,\ldots,m_k$ sont des motifs de longueur quelconque. Dans ce cas, chaque motif de longueur l doit être tiré avec une probabilité p^l, où p est racine réelle positive du polynôme

$$1-\sum_{i=1}^{k} x^{\|m_i\|}.$$

Cette racine est l'unique racine positive d'après la *loi des signes* de Descartes (voir par exemple [12, pp. 89–93]).

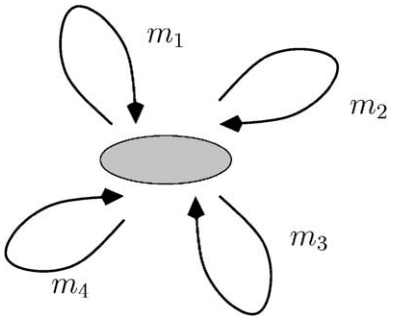

Fig. 1. Une étoile à quatre motifs.

Le nombre de rejets reste en O(1) et le nombre moyen de motifs à choisir pour construire un mot de longueur n diminue bien évidemment lorsque la longueur des motifs est grande. Pour choisir un motif avec une probabilité donnée, on tire d'abord la longueur l du motif par la méthode étendue du tirage paresseux (cf. Section 1). Puis on choisit parmi les motifs de même longueur l, à nouveau par un tirage bit à bit.

Remerciements

Les auteurs remercient les arbitres pour le soin et la pertinence de leur correction.

References

[1] L. Alonso, R. Schott, Random Generation of Trees, Kluwer Academic Publishers, Dordrecht, 1995.
[2] E. Barcucci, R. Pinzani, R. Sprugnoli, Directed column-convex polyominoes by recurrence relation, Lecture Notes in Computer Science, Vol. 668, Springer, Berlin, 1993, pp. 282–298.
[3] A. Denise, Génération aléatoire et uniforme de mots de langages rationnels, Theoret. Comput. Sci. 159 (1996) 43–63.
[4] A. Denise, P. Zimmermann, Uniform random generation of decomposable structures using floating-point arithmetic, Theoret. Comput. Sci. 218 (2) (1999) 233–248 (Caen '97).
[5] P. Flajolet, P. Zimmermann, B. Van Cutsem, A calculus for the random generation of labelled combinatorial structures, Theoret. Comput. Sci. 132 (1994) 1–35. (A preliminary version is available in INRIA Research Report RR-1830.)
[6] T. Hickey, J. Cohen, Uniform random generation of strings in a context fre language, SIAM J. Comput. 12 (1983) 645–655.
[7] D.E. Knuth, A.C. Yao, The complexity of nonuniform random number generation, [CA] algorithms and complexity, Proceedings of the Symposium, Pittsburgh, 1976, pp. 357–428.
[8] J.G. Penaud, E. Pergola, R. Pinzani, O. Roques, Chemins de Schröder et hiérarchies aléatoires, Theoretical Computer Science 255 (2001) 345–361.
[9] J.G. Penaud, O. Roques, Génération de chemins de Dyck à pics croissants, Discrete Math. 246 (2002) 255–267.
[10] E. Schröder, Vier combinatorische probleme, Z. Math. Phys. 15 (1870) 361–370.
[11] R. Sedgewick, Ph. Flajolet, Introduction à l'analyse des algorithmes, International Thomson Publishing Company, Paris, 1996.
[12] D.J. Struik (Ed.), A Source Book in Mathematics 1200–1800, Princeton University Press, New York, 1986.