Computer Science Journal of Moldova, vol.**3**, no.2(8), 1995

Le codage des arbres binaires

Vicentiu Vajnovszki

1 Introduction

Les arbres sont principalement la structure de donnée utilisés pour stocker des données ordonnées et d'après Knuth la plus importante structure non-linéaire intervenant dans l'informatique. Ils sont très utilisés dans tous les domaines, parce que bien adaptés à la représentation naturelle d'information organisée homogène, et d'une grande rapidité et commodité de manipulation. On trouve cette structure dans tous les domaines de l'informatique, que ce soit par exemple en compilation (arbres syntaxiques pour représenter les expressions ou productions possibles d'un langage), en imagerie (les arbres quaternaires), en algorithmique (par exemple elle est le support de méthodes de tris ou de gestion d'information en tables) ou encore dans les domaines de l'intelligence artificielle (arbres de jeux, arbres de décisions, de résolution etc.). Les arbres sont constitués des *nœuds* (ou *nœuds internes*) et des *feuilles* (ou *nœuds externes*). Un arbre est soit vide soit il a une racine qui a un ou plusieurs fils qui sont récursivement des arbres. Un arbre binaire a la popriété suivante : chaque nœud (interne) a exactement deux fils, le fils gauche et le fils droit. Dans ce papier nous allons nous restreindre aux arbres binaires ordonnés, orientés, qui soront appelés brièvement arbres. Voir [10] et [11] pour détailles sur le traitement des arbres. Le nombre des différents arbres avec n nœuds est donné par le bien connu nombre de Catalan : $c_n = (2n)!/n!(n+1)!$ Plusieurs algorithmes combinatoires donnent des méthodes pour énumérer tous ces arbres. Dans ce cas il est avantageux d'avoir quelques représentations concises des arbres. Une situation idéale est d'avoir une bijection entre une suite de longueur n

(ou $2n$) et les c_n arbres avec n nœuds. Heureusement il y a plusieurs méthodes de codage. Le sujet de cette article est leur passage en revue.

2 Le codage

Supposons que l'on veuille générer des données pour tester un programme qui travaille sur des arbres et que l'on dispose d'une bijection entre certaines suites et les arbres de taille n. Alors, une solution consiste à générer aléatoirement une telle suite, qui peut être une tâche plus facile que de générer effectivement l'arbre, puis d'appliquer cette bijection pour créer l'arbre sur lequel le programme sera testé. Une autre aire d'applications peut être la compression des arbres : si l'on veut stocker des arbres de grande taille, on peut leurs associer un entier ou une suite qui code l'arbre et qui sera stocké à la place de l'arbre. Cette méthode devient efficace si les données sont stockées longtemps et l'accès aux données se fait rarement. Le procesus de codage doit satisfaire certains desiderata. Premièrement la représentation des arbres par des suites doit être bijective : à chaque arbre correspond une unique suite et réciproquement. Deuxièmement il faut avoir une caracterisation des suites qui sont les codes de certains arbres de ceux qui ne le sont pas. Troisièmement il faut que l'on dispose d'un procédé efficace pour passer d'un arbre à son code et vice versa. Considérons un arbre à n nœuds. La structure de l'arbre engendre un ordre, appelé *ordre symétrique* pour les nœuds, en respectant le procédé récursif suivant : on traverse d'abord en ordre symétrique le sous-arbre gauche, on visite la racine, puis on traverse en ordre symétrique le sous-arbre droit. La figure 1 montre un arbre où les nœuds ont été étiquetés en ordre symétrique.

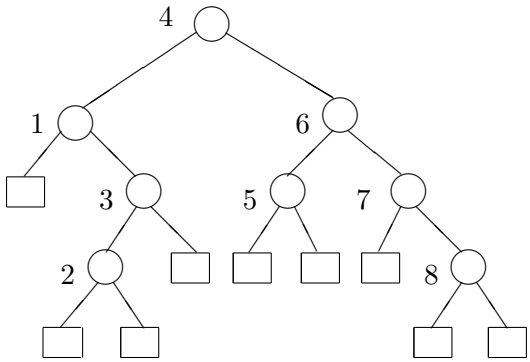

Figure 1

Une autre façon de parcourir un arbre est le *préordre*: on visite d'abord la racine, on traverse en préordre le sous-arbre gauche, puis on traverse en préordre le sous-arbre droit. Parce que les nœuds internes d'un arbre déterminent complètement sa structure, ses feuilles seront considérées seulement si elles sont essentielles pour le codage en question. Considérons maintenant le schèma général pour construire le code. Soit un arbre avec n nœuds. Nous associons à chaque nœud interne ou chaque feuille ou à tous les deux une étiquette (qui est un entier ou une lettre). Les méthodes de codage diffèrent par le choix de ces étiquettes. En traversant les nœuds de l'arbre (les étiquettes) dans un ordre spécifique on obtient un n-tuple. Dans la plupart des applications ce n-tuple est suffisant. Pourtant parfois on a besoin de transformer le n-tuple en nombre entier. Cette conversion s'appelle *rang* et l'opération inverse s'appelle *rang inverse*. Une suite s'appelle *faisable* pour une certaine méthode de codage si elle représente un arbre avec cette méthode. Il existe quelques exceptions au schèma général décrit ci-dessus. Premièrement il existe des techniques pour passer d'un arbre à son rang (un entier) et vice versa sans passer par son code (qui est une suite); une telle méthode est présentée en [29]. D'autre part il existe des méthodes qui utilisent deux traversées de l'arbre : une

pour étiqueter les nœuds et/ou les feuilles et une autre pour lire ces étiquettes. Dans la section 3.3 on présente plusieurs permutations des arbres qui utilisent différents types de traversée. Une autre exception est le codage basé sur la rotation des arbres qui sera présenté dans la section 5. A la fin de cette section on va présenter plusieurs notations nécessaires en suite. Si u est le nœud d'un arbre alors son sous-arbre gauche (resp. droit) sera noté par $gauche(u)$ (resp. $droit(u)$). Le nombre des nœuds de l'arbre T est noté par $taille(T)$.

3 Codages basés sur permutations

Dans cette section on présentera les différents codages basés sur les permutations et la possibilité de caractériser de façon unique un arbre par deux traversées consécutives en deux ordres différents. Le fait de traverser deux fois l'arbre et l'espace nécessaire pour indiquer les deux ordres de traversées font que les codages basés sur la permutation sont moins efficaces que les autres codages qui seront décrits plus tard. Pourtant leur étude a un certain intérêt théorique.

3.1 Permutations d'arbres

Un codage simple peut être obtenu en étiquetant les nœuds par leur occurrence en ordre symétrique et en lisant les étiquettes en préordre. La suite d'entiers obtenue s'appelle permutation d'arbres [12].

La permutation d'arbre qui correspond à l'arbre de la figure 1 est : (41326578). La permutation d'arbre peut être obtenue dans une seule traversée si pour chaque nœud on connaît la taille de chaque sous-arbre. L'item x_i du code qui correspond au nœud i (le $i$ème nœud en préordre) est :

$taille(gauche(i))$, si i est la racine

$p_i - taille(droit(i))$, si i est fils gauche

$p_i - taille(gauche(i))$, si i est fils droit

où p_i dénote l'item qui correspond au parent du nœud i. Les permutations d'arbre ont une caractérisation assez simple : une permutation d'entiers $x = (x_1, x_2, ..., x_k)$ est une permutation d'arbre si et seulement si il n'existe pas une sous-suite $x_i x_j x_k$ de x telle que $x_i x_j x_k$ [12]. Etant donnée une permutation d'arbre $(x_1, x_2, ..., x_n)$, comment on peut construire l'arbre correspondant ? La construction se base sur les observations suivantes. Premièrement, x_1 est l'étiquette de la racine. Deuxièmement les $x_1 - 1$ items suivants forment le code du sous-arbre gauche. Et troisièmement, le code du sous-arbre droit est formé avec les items restants ayant ajouté $x_1 - 1$ à chacun d'entre eux.

3.2 Séquence de ballottage

Les permutations d'arbres peuvent être représentées aussi par les tableaux d'inversions leur correspondant. Soit $(x_1, x_2, ..., x_n)$ une permutation d'arbre. On définit son tableau d'inversion $(y_1, y_2, ..., y_n)$ par : y_i est le nombre d'items de $(x_1, x_2, ..., x_n)$ plus grands que i et situés à sa droite, pour $i = 1, 2, ..., n$. Les éléments du tableau d'inversion satisfont $0 \leq y_i \leq n - 1$ et $y_i \geq y_{i+1}$ pour $i = 1, 2, ..., n$. De plus y_i est toujours 0. Ce type de tableau d'inversion (associé à une permutation d'arbre) s'appelle séquence de ballottage [23]. La séquence de ballottage de l'arbre de la figure 1 est : (64442210). Pour l'étymologie de l'expression considérons une permutation d'arbre p et une pile vide. On peut trier les items de p par deux opérations : (1) mettre l'item suivant dans la pile, et (2) transférer l'élément du haut de la pile vers la sortie. Ces opérations sont d'habitude notées par S et X respectivement. Pendant le processus de tri le nombre d'opérations X n'excède jamais le nombre d'opérations S. L'opération ressemble au comptage des voix où on a deux candidats S et X, et X n'a jamais la majorité pendant le comptage [24].

3.3 Paire de permutations

Dans la sous-section 3.1 on a présenté la permutation d'arbres où les nœuds ont été étiquetés par leur occurrence après une traversée en préordre et lus pendant une traversée en ordre symétrique. Ce type

de codage peut être généralisé à d'autres paires de type de traversée [9]. On peut caractériser les traversées par les notations suivantes : r signifie la racine, G et R le sous- arbre gauche et droit. Donc GrD est la traversée en ordre symétrique et rGD signifie la traversée en préordre et la traversée GDr (sous-arbre gauche, sous-arbre droit, racine) s'appelle *postordre*. Il existe 6 types de traversées correspondants aux 3! permutations des trois lettres. Les traversées DrG, rDG , DGr s'appellent *ordre symétrique inverse*, *préordre inverse* et *postordre inverse* respectivement. La permutation d'arbre présentée dans la sous-section 3.1 est obtenue par la paire [GrD, rGD]. Il existe 6x6=36 paires de permutations, et les paires de traversée qui établissent une correspondance entre les arbres et une permutation des n premiers entiers s'appellent classe non-dégénérée. Knuth a montré qu'une classe est non-dégénérée si exactement un ordre de visiter (d'étiqueter ou de lecture) est en ordre symétrique, donc il existe 16 classes non-dégénérées. Trojanovski a étudié la paire de permutations [rGD,GrD] et la permutation [rGD,GrD] de l'arbre de la figure 1 est : (32416578).

4 Codage suivant le chéma général

4.1 Suite de niveaux

On définit le niveau de la racine d'un arbre par 0, et si un nœud a le niveau k ces fils auront le niveau $k+1$. En parcourant les feuilles de l'arbre de gauche à droite, les nombres de niveaux des feuilles forment une suite appelée suite de niveaux [25]. La suite de niveaux de l'arbre de la figure 1 est : (244333344). Les suites de niveaux ont quelques propriétés intéressantes. Soit $x = (x_1, x_2, ..., x_n)$ une suite de niveaux et $x_{k-1} = x_k = q$ la paire la plus à gauche (resp. à droite) d'entiers ayant la même valeur. La réduction à gauche (it resp. à droite) est le processus qui cosiste à remplacer la suite initiale par $x = (x_1, x_2, ..., x_{i-1}, q, x_{i+1}, ..., x_n)$. Une suite d'entiers de longueur $n+1$ est la suite de niveaux d'un arbre de taille n si et seulement si après n réductions à gauche ou à droite la suite se réduit au seule entier 0. Une autre propriété intéressante (et assez intuitive) des suites

de niveaux est que la somme de 2^{-x_i} pour $i = 1, ..., n$ vaut 1. Cette propriété est nécessaire pour caractériser les suites de niveaux, mais elle n'est pas suffisante.

4.2 Suite de Zaks

Avec la méthode de Zaks, on étiquette les feuilles avec 0 et les nœuds avec 1. Lisant ces étiquettes en préordre, on obtient une suite de longueur $2n+1$ (n fois 1 et $n+1$ fois 0). Parce que la dernière étiquette lue est toujours 0 elle peut être tée de la suite. La suite de Zaks de l'arbre de la figure 1 est : (1101100011001010). Une suite composée de n zéros et n uns est la suite de Zaks d'un arbre si et seulement si dans chaque préfixe il y a au moins autant de uns que de zéros. Donc, trouver les suites de Zaks de longueur $2n$ revient à trouver les solutions du problème de la queue : $2n$ personnes font la queue devant un guichet, n d'entre eux ayant des pièces d'un franc et les autres n personnes ayant des pièces de deux francs. Les billets cotent un franc chacun. Au moment de l'ouverture il n'y a pas de pièces dans la caisse. Combien de façons d'ordonner la queue y a-t-il pour que la caisse puisse rendre la monnaie à chacun ? Si dans une suite de Zaks on remplace les 1 par le symbole '$\circ$' et les 0 par le symbole '$\square$'on obtient la représentation préfixée polonaise de l'arbre. Pour l'arbre de la figure 1 cette représentation est : $\circ\circ\square\circ\circ\square\square\square\circ\circ\square\square\circ\square\circ\square$. Les suites de Zaks ont une autre caractérisation intéressante : déduisant dans une suite de Zaks la configuration 10 autant de fois qu'il est possible, on arrive à une suite vide.

4.3 P-suite

Une autre façon de coder les arbres est la P-suite $(x_1, x_2, ..., x_n)$ où x_i est le nombre des nœuds lus avant la feuille i, parcourant l'arbre en préordre. La P-suite peut-être obtenue à partir de la suite de Zaks associée à l'arbre : si pour chaque $\square$ on compte le nombre des $\circ$ qui le précèdent dans la suite de Zaks on obtient la P-suite correspondant à l'arbre. La P-suite de l'arbre de la figure 1 est : (244466788). Les suites faisables sont facile à caractériser. Il est clair que si $(x_1, x_2, ..., x_n)$ est

la P-suite d'un arbre alors $x_i \leq x_{i+1}$ pour $i = 1, 2, ..., n-1$ et $x_1 = n$. De plus, $x_i \geq i$ est vérifiée pour $i = 1, 2, ..., n$. Ces trois conditions ensemble sont nécessaires et suffisantes qu'un n-tuple soit une P-suite.

4.4 Deux codages asymétriques

Dans cette sous-section on présente deux codages qui sont asymétriques dans le sens que l'on considère seulement les sous-arbres gauches pour obtenir le code. Naturellement, il existe une contrepartie droite que l'on omettra. Les deux méthodes de codages présentées ci-dessous peuvent être réalisées dans un seul parcours de l'arbre si pour chaque nœud on connaît la taille du sous-arbre gauche dont il est la racine.

4.4.1 L-suite

On étiquette les feuilles d'un arbre par 1 et les nœuds par la taille du sous-arbre dont il est la racine (autrement dit, par la somme des étiquettes de leurs fils). Lisant ces étiquettes en préordre, mais uniquement pour la racine et pour les nœuds qui sont fils gauche, on obtient la L-suite de l'arbre [15]. La L-suite de l'arbre de la figure 1 est : (941212111). Un arbre de taille n aura la L-suite un $(n+1)$- tuple. Si $(x_1, x_2, ..., x_n, x_{n+1})$ est la L-suite d'un arbre de taille n alors : $x_1 = n + 1$ et si $0 \leq k < i < k + x_k$ alors $i + x_i \leq k + x_k$ pour $i = 1, ...n - 1$. Cette condition est nécessaire et suffisante pour qu'un n-tuple soit une L-suite.

4.4.2 Suite de poids

Une autre codage asymétrique est la suite de poids qui est obtenue en lisant en ordre symétrique les nœuds après qu'ils aient été étiquetés chacun avec le nombre de feuille de son sous-arbre gauche [16]. La suite de poids de l'arbre de la figure 1 est : (112412119). La suite de poids peut être obtenue en parcourant de gauche à droite les feuilles de l'arbre après que chacune ait été étiquetée avec le nombre de feuilles du sous-arbre dont elle est la dernière feuille. Soit $(x_1, x_2, ..., x_n)$ une suite de poids. Pour chaque nœud j situé dans la sous-arbre gauche

du nœuds i on a : $i - x_i \leq j - x_j$. Cette condition est une condition nécessaire et suffisante.

5 Codage basé sur rotations

Une rotation est une transformation qui change un arbre dans un autre arbre comme suit : si un nœud v est le fils gauche d'un nœud u alors la rotation à droite transforme u dans le nouveau fils droit de v, l'ancien fils droit de v (s'il existe) devient le fils gauche de u et l'ancien parent de u (s'il existe) devient le nouveau parent de v. La rotation à gauche transforme l'arbre ainsi obtenu dans l'arbre d'origine (voir la figure 2).

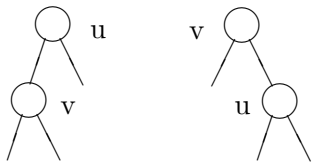

Figure 2

Dans un arbre on peut appliquer la rotation à droite (*resp.* gauche) pour chacun de ses nœuds qui ont un fils gauche (*resp.* droit) autre qu'une feuille. Le codage basé sur rotations compte dans une suite d'entiers le nombre de rotations nécessaires pour passer d'un arbre fixé à cet arbre. On note par ω l'arbre où chaque nœud a comme fils droit une feuille et il sera choisie comme l'arbre fixe. La suite $(x_1, x_2, ..., x_n)$ d'entiers s'appelle suite de rotations [35] pour un arbre, s'il peut être obtenu à parti de ω en lui faisant subir les transformations suivantes : x_n rotations à droite au nœud de niveau $n-1$, dans l'arbre ainsi obtenu x_{n-1} rotations à droite au nœud de niveau $n-2$, ... , dans l'arbre obtenu à l'étape précédente x_1 rotations à droite au nœud de niveau 0 (la racine). L'arbre de la figure 1 a la suite de rotations :

(30001011). La suite $(x_1, x_2, ..., x_n)$ est la suite de rotation d'un arbre si $x_j + x_{j+1} + ... + x_n \leq n - j$, pour $j = 1, ..., n$. Cette condition est nécessaire et suffisante pour qu'une suite soit une suite de rotation.

6 Codages avec quatre letters

On aura remarqué qu'à une exception près tous les codes présentés jusqu'ici sont des suites ayant comme items des entiers qui dépendent de la taille de l'arbre codé. Dans cette section on présentera des codages où les items peuvent prendre 4 valeurs. Avec la relation $3^n < c_n < 4^n$ (c_n est le nombre des arbres avec n nœuds) on peut déduire que quatre est le nombre minimal d'items qu'on peut utiliser pour coder les arbres avec des suites de longueur la taille de l'arbre.

6.1 ll'rr'-suite

Dans ce codage on étiquette la racine par r et les fils d'un nœud sont étiquetés selon les règles suivantes :

nœud	fils gauche	fils droit
r	l'	r
r'	l	r'
l	l	r
l'	l'	r'

Le code d'un arbre, appelé $ll'rr'$-suite [18], est obtenu en lisant de gauche à droite les feuilles ainsi étiquetées, donc l'arbre de la figure 1 aura le code : $(l'lrr'l'r'l'l'r)$. Il faut observer que l'arbre doit être parcouru une seule fois en étiquettant tous les nœuds et feuilles rencontrées. Pourtant, le code peut être obtenu en parcourant uniquement les feuilles de gauche à droite si pour chaque feuille on dispose de deux informations : 1) la feuille est un fils gauche ou fils droit de son parent et 2) le nombre $ch(i)$, des changements de direction dans l'unique chemin qui mène de la racine de l'arbre à la feuille i. On va noter x_i l'étiquette de la feuille i et $g(i)$ (*resp.* $d(i)$) si la feuille i est le fils

gauche (*resp.* droite) de son parent. L'étiquette x_i de la feuille i est donnée par les quatre règles suivantes [18] :

r si $d(i)$ et $ch(i)$ mod 4=0 ou 3,

r' si $d(i)$ et $ch(i)$ mod 4=1 ou 2,

l si $g(i)$ et $ch(i)$ mod 4=2 ou 3,

l' si $g(i)$ et $ch(i)$ mod 4=0 ou 1,

où x mod y signifie le reste de la division de x par y.

Pour caractériser les $ll'rr'$-suite on défini les réductions : la transformation de la configuration : $l'r$ en r, lr en l, lr' en r', $l'r'$ en l' est appelée réduction. Un $ll'rr'$-suite $(x_1, x_2, ..., x_n)$ est le code d'un arbre si après $n-1$ réductions la suite se réduit à la suite avec une seule lettre : r. Cette condition est nécessaire et suffisante.

6.2 lrud-suite

Ce codage a été introduit dans [4] en mettant en bijection les arbres binaires et les chemins du demi-plan gauche qui commencent et qui s'achèvent toujours sur l'axe des x. Le chemin est décrit par une suite composée des lettres : u (up), d (down), l (left) et r (right).

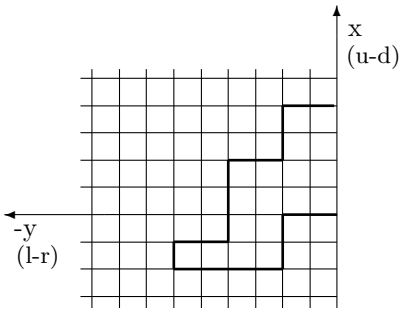

Figure 3

Dans un arbre chaque nœud, sauf la racine, peut vérifier l'un des quatre cas suivants :

- être fils droit ayant une feuille comme frère gauche (figure 4.a), il sera étiqueté avec u;
- être fils gauche ayant une feuille comme frère droit (figure 4.b), il sera étiqueté avec d;
- être fils droit ayant un nœud comme frère gauche (figure 4.c), il sera étiqueté avec r;
- être fils gauche ayant un nœud interne comme frère droit (figure 4.c), il sera étiqueté avec l.

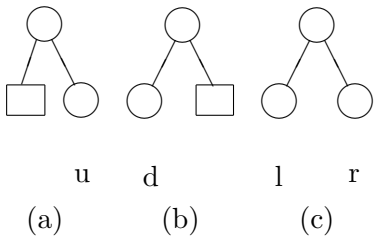

Figure 4

En lisant en préordre les étiquettes ainsi obtenues on obtient une suite qui s'appelle $udlr$-suite. La $lrud$-suite de l'arbre de la figure 1 est : $(ludrlru)$ La caractérisation d'une $udlr$-suite ressemble à celle d'une suite de Zaks : dans chaque préfixe d'une $udlr$-suite il y a au moins autant de l que de r et le nombre total des l est égal à celui des r (il n'y a pas de restriction sur les u et d). Cette condition est une condition nécessaire et suffisante.

7 Conclusions

On a présenté plusieurs méthodes pour représenter la structure d'un arbre, sans privilégier une certaine méthode pour une application spécifique. Généralement un critère de sélection n'existe pas. Certains codages exprime mieux que d'autres certaines propriétés des arbres et le choix du codage peut être essentielle. Si la simplicité du codage est recherchée alors on peut choisir ceux présentés dans les sections 4 et 6. Si le critère est de minimiser l'espace mémoire pour la représentation des arbres alors les méthodes présentées dans la section 6 semblentles meilleures, et si on veut que l'ordre lexicographique induise un ordre naturel sur les arbres alors on utilisera les codages présentés dans la section 4. Le codage de Zaks et par L-suites peuvent être généraliser naturellement pour les arbres k-aires, les $ll'rr'$-suites pour les arbres neuronaux et les $lrud$-suites pour les arbres binaires non-ordonnés [31]. Enfin pour les codages présentés dans les sections 4, 5, 6 on dispose d'algorithmes de génération, de rang et de rang inverse efficaces. Le plus performant d'entre eux est ce qui utilise la suite de poids comme codage [16], en dépensant en moyenne 4/3 comparaisons par arbre généré (donc indépendant de la taille des arbres). Avec les suites de rotations on dispose d'algorithmes de génération sans boucle ([3], [14]), où dans le pire des cas le temps nécessaire pour passer d'une suite à la suite suivante est inférieure à une constante donnée. L'avantage de cette algorithme par rapport à un algorithme de complexité moyenne constante est que lorsqu'on implémente l'algorithme sur une machine avec des circuits séquentiels synchrones il faut compter sur le pire des cas.

References

[1] M. D. Atkinson and J. R. Sack, Generating binary trees at random, *Information Processing Letters* **41** (1992), 21-23.

[2] D. Rœlants van Baronaigien and F. Ruskey, Generating t-ary trees in A-order, *Information Processing Letters*, **27** (1988), 205-213.

[3] D. Rœlants van Baronaigien, A loopless algorithm for generating binary tree sequences, *Information Processing Letters* **39** (1991), 189-194.

[4] E. Barcucci and M. C. Verri, Some more properties of Catalan's numbers, *Discrete Mathematics* **102** (1992), 229-237.

[5] T. Beyer and S. M. Hedetniemi, Constant time generation of rooted trees., *SIAM J. Comput.* **9**(4) (1980), 706-712.

[6] A. Bonnin and J. M. Pallo, Sur la génération des arbres binaires par les B-suites, *Discrete Mathematics* **51** (1984), 111-117.

[7] N. Dershowitz and S. Zaks, Enumeration of ordered trees, *Discret Math.* **31** (1980), 9-28.

[8] D. K. Gupta, Generation of Binary trees from (0-1) Codes, *Intern. J. Computer Math.*, **42** (1991), 157-162.

[9] R. F. Hille, Binary Trees and Permutations, *The Australian Computer Journal*, **17** (1985), 85-87.

[10] D. E. Knuth, The art of computer programming, Vol. 1, (Addison-Wesley, 1968).

[11] D. E. Knuth, The art of computer programming, Vol. 3, (Addison-Wesley 1972).

[12] G .D. Knott, A numbering system for binary trees, *Comm. ACM*, **20**(2), (1977), 113-115.

[13] J. M. Lucas, The rotation graph of binary trees is Hamiltonian, *J. Algorithms* **9** (1988), 503-535.

[14] J. M. Lucas, D. Rœlants van Baronaigien and F. Ruskey, On rotation and the generation of binary trees *J. Algorithms* **15** (1993), 343-366.

[15] J. M. Pallo and R. Racca, A note on generating binary trees in A-order and B-order, *Intern. J. Computer Math.* **18**(1) (1985), 27-39.

[16] J. M. Pallo, Enumerating, ranking and unranking binary trees, *Comput.J.* **29** (1986), 171-175.

[17] J. M. Pallo, Generating trees with n nodes and m leaves, *Int. J. Comp Math.* bf 21 (1987), 133-144.

[18] J. M. Pallo, Coding trees by words over on alphabet with four letters, *Journal of Information and Optimization Sciences*, **13** (1992), 257-266.

[19] H. Prodinger, A correspondence between ordered trees and non-crossing partitions, *Discrete Math.* **46** (1983), 205-206.

[20] A. Proskurowski and F. Ruskey, Binary tree gray codes, *Journal of Algorithms*, **6** (1985), 225-238.

[21] V. Ramanan and C. L. Liu, Permutation representation of k-ary trees, *Theoretical Computer Science*, **38** (1985), 83-98.

[22] D. Rotem, On a correspondence between binary trees and a certain type of permutation, *Information Processing Letters*, **4** (1975), 58-61.

[23] D. Rotem and V. L. Varol, Generation of binary trees from ballot sequences, *Journal of the ACM* **25**(3) (1978), 396-404.

[24] D. Rotem, Stack sortable permutations, *Discrete Mathematics*, **46** (1981) 185-196.

[25] F. Ruskey and T.C. Hu, Generating binary trees lexicographically, *SIAM J. Comput.* **6** (1977), 745-758.

[26] F. Ruskey and A. Proskurowski, Generating binary trees by transpositions, *Journal of algorithms* **11** (1990), 68-84.

[27] P. Schultz, Enumerations of rooted trees with an application to group presentations, *Discret Math.* **41** (1992), 199-214.

[28] I.Semba, Generation of binry trees from stak permutation, *Journal of Information Processing*, **5** (1982) 102-105.

[29] M. Solomon and R.A. Finkel, A note on enumerating binary trees, *Journal of the ACM*, **27**(1) (1980), 3-5.

[30] V. Vajnovszki and J. Pallo, Generating binary trees in A-order from codewords defined on four-letter alphabet, *Journal of Information and Optimization Sciences*, **15**(3)(1994) 345-357.

[31] V. Vajnovszki, Generating, ranking and unranking for binary unordered trees, *Bulletin of EATCS*, **57**(95), p.221–229

[32] S. Zaks and Richards, Generating trees and other combinatorial objects lexicografically, *SIAM J. Comput.* **8**(1) (1979), 73-81.

[33] S. Zaks, Lexicografic generation of ordered trees, *Theoretical Computer Science*, **10** (1980), 63-82.

[34] S. Zaks, Generating and ranking of k-ary trees, *Information Processing Letters*, **14**(1) (1982) 44-48.

[35] D. Zerling, Generating Binary Trees Using Rotations, *Journal of the Association for Computer Machinery*, **32**(3) (1985) 694-701.

Vicentiu Vajnovszki
Département d'Informatique
Université de Bourgogne
B.P.138 21004 Dijon Cédex
e–mail: *vincent@depinfo.u − bourgogne.fr*

Received 18 September, 1995