Malaysian Journal of Computer Science, Vol. 17 No. 1, June 2004, pp. 24-31

GENERATION OF BINARY TREES IN B-ORDER FROM (0-1) SEQUENCES

Hayadeh Ahrabian
Department of Mathematics and Computer Science
Faculty of Science, University of Tehran
Tehran, Iran
Tel. and Fax: 6412178
email: ahrabian@khayam.ut.ac.ir

Abbas Nowzari-Dalini
Department of Mathematics and Computer Science
Faculty of Science, University of Tehran
Tehran, Iran
Tel. and Fax: 6412178
email: nowzari@khayam.ut.ac.ir

ABSTRACT

An efficient recursive algorithm has been developed to generate binary trees in B-order from 0-1 sequences. The generation algorithm produces each tree in constant average time O(1). The ranking and unranking algorithms with O(n) time complexity are also presented.

Keywords: ***Binary trees, B-order, 0-1 Sequences, Recursion***

1.0 INTRODUCTION

Binary trees are widely used to represent and maintain ordered data of various types. There have been many algorithms published for listing binary trees. In the tree generation problem initially, we present an equivalence between some set of integer sequences and all the binary trees with *n* internal nodes. Once the equivalence is established, an algorithm is designed to generate all the sequences. There exists a vast literature on the problem of generating binary trees (Gupta, 1993; Lucas, Roelants Van Baronaigien, and Ruskey, 1993; Pallo and Racca, 1985; Roelants Van Baronaigien, 1991; Rotem and Varol, 1978; Ruskey and Hu, 1997; Vajnovszki, 1998; Zaks, 1980). In most published papers, the trees are encoded as 0-1 sequences and these sequences are generated (Gupta, 1993; Ruskey and Hu, 1977; Zaks, 1980). Any generation algorithm imposes an ordering on the set of trees. The most well known orderings on binary trees are the A-order and the B-order (Pallo and Racca, 1985). The A-order definition uses global information concerning the tree nodes whereas the B-order definition uses local information (Zaks, 1980).

In (Ahrabian and Nowzari, 1998), the authors exhibit generating, ranking and unranking algorithms of binary trees in Ballot-order, with 0-1 sequences. It is noted that Ballot-order is an order for the codes not for binary trees and this order is the same as Ballot sequence (Rotem and Varol, 1978; Rotem, 1975). The running time of their generation algorithm is $O(1)$ on the average per tree and for ranking and unranking algorithms is $O(n^2)$. In this paper a new generation algorithm in B-order is given and also the new ranking and unranking algorithms with $O(n)$ time complexity are presented.

2.0 DEFINITIONS AND NOTATIONS

We state here definitions and notations that we shall use throughout this paper. In a rooted, ordered, unlabeled binary tree T, every node except the root has a parent. Every internal node O has a left and a right child (the order is significant), and each of those children is also a tree called subtree of the internal node. External nodes or leaves □ have no children (Knuth, 1973). We denote by T_L and T_R the left and right subtrees of tree $T \neq$ □, and $|T|$ shows the number of nodes in the tree T. Let r_T be the degree of the root of tree T, *i.e.* $r_T = 0$ if T □ and $r_T = 2$ otherwise.

Definition 1. Two trees T and T' are in B-order, $T<T'$ if:

1) $r_T < r_T$, or
2) $r_T = r_{T'}$ and $T_L < T'_L$, or
3) $r_T = r_{T'}$ and $T_L = T'_L$ and $T_R < T'_R$.

Definition 2. Two trees T and T' are in A-order, $T<T'$ if:

1) $|T|<|T'|$, or
2) $|T|=|T'|$, and $T_L < T_L'$, or
3) $|T|=|T'|$ and $T_L = T_L'$, and $T_R < T_R'$,

It should be noted that we use Definition 1 throughout this work. Definition 1 and 2 are not equivalent. For example; $T<T'$ (by Definition 2) but $T'<T$ (by Definition 1) for the trees T and T' in Fig. 1.

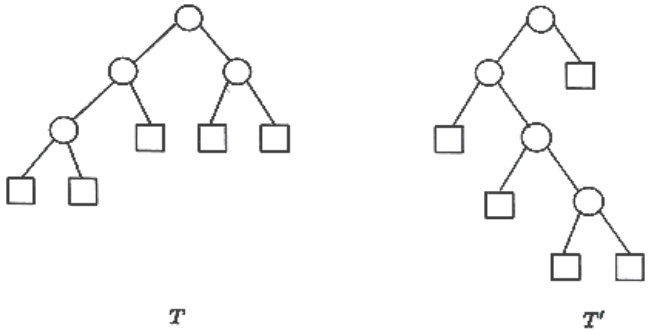

Fig. 1: Ordering of two trees T and T'

Definition 3. Let T be a binary tree of n internal nodes; it is represented by n ones and n zeros. This code is constructed by labeling the internal nodes of the tree by 1, and the external nodes by 0, and then traversing the binary tree by pre-order (staring from the root, for each node recursively visit a node, move left, and move right). Since the last digit is always zero, therefore this digit is discarded.

For example the corresponding code for the tree in Fig. 2, is represented with 11100011001100 (as we can see by Definition 3, last zero is ignored).

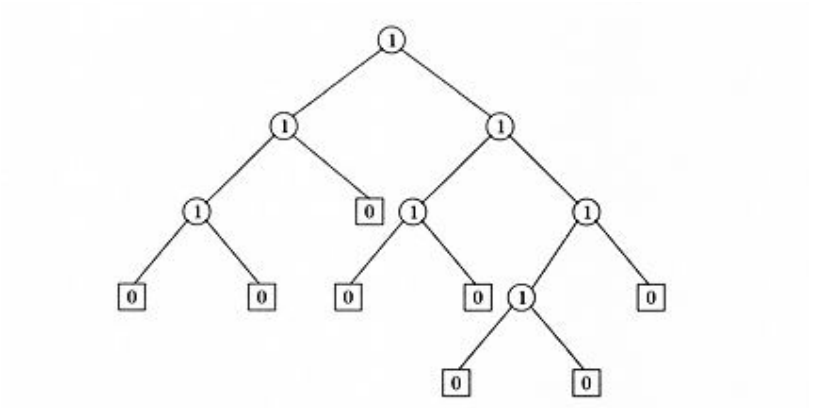

Fig. 2: A7-node binary tree

Theorem 1. A 0-1 sequence $X = \{x_i\}_1^{2n}$ is feasible iff:

1) it has n 0's and n 1's, and
2) the number of ones in any prefix the sequence is equal to or greater than the number of zeros (dominating property).

Theorem 2. The following sets are in a 1-1 correspondence with one another:

1) all binary trees with n internal nodes,
2) all 0-1 sequences $X = \{x_i\}_1^{2n}$ which are feasible.

It is clear that the number of the above sequences is equal to

$$C_n = \frac{1}{n+1}\binom{2n}{n}.$$

Let $\mathbf{X_n}$ be the set of all bit strings with n zeros and n ones having the dominating property. The elements of $\mathbf{X_n}$ are also a classic way to represent well-formed parentheses (W.F.P in short) strings (Lucas, Roelants Van Baronaigien, and Ruskey, 1993; Bultena and Ruskey, 1998; Germain and Pallo, 1996). By definition, W.F.P is the word of the language generated by the grammar

$$S \rightarrow (S) \mid S\,S \mid \lambda,$$

where λ is the empty word. We can represent a W.F.P. string with n left and n right parentheses by a 0-1 sequence, which is obtained by replacing the left parentheses by a 1 and the right parentheses by a 0 (Zaks, 1980; Er, 1983).

3.0 GENERATION ALGORITHM

In this section, a recursive algorithm is given to produce a list of tree sequences. The algorithm produces the tree sequences by interchanging any adjacent 10 by 01. The generation sequence starts with the string $1^n\,0^n$. By the first possible right interchange the n th 1 in the string is shifted one position to the right and the next sequence is obtained, continuously the other sequences are generated by shifting this 1 until we get to $1^{n-1}\,0^{n-1}\,1\,0$. Later we repeat the above process for the remaining 1's.

The generation algorithm *GenX-Seq* illustrated in Fig. 3 has double underlying recursions and initially is called with: $X = 1^n 0^n,\quad k = n,\quad l = n-1,$ and $q = 1$. The construction process is demonstrated by the recursion tree $\mathbf{T_n}$ (for $n = 4$) in Fig. 4.

```
Procedure GenX-Seq (X: Xseq ; k, l, q : Integer ) ;
Begin
      If( k < n ) Then Begin
            x_{2n-k-l} := 0 ;
            x_{2n-k-l+1} := 1 ;
      End ;
      WriteSeq ( X );
      If( k > 1 ) Then Begin
            GenX-Seq( X, k-1,  1,  l );
            If (l < k) And (l < q) Then
                GenX-Seq( X, k,  l+1,  q );
      End;
End;
```

Fig. 3: Generation algorithm in the reverse order of B-order

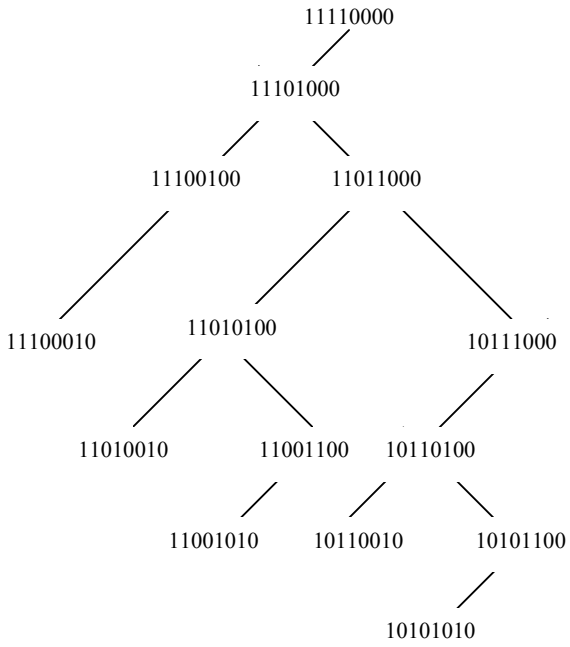

Fig. 4: The recursion binary tree T_n, for $n = 4$

4.0 ANALYSIS OF THE ALGORITHM

With regard to the underlying recursions in the algorithm the time complexity of the algorithm, can be obtained from the following recursive formula:

$$Gx(k, l, q) = \begin{cases} 1 & \text{if } k = 1, \\ 1 + Gx(k-1,\ 1,\ l) + Gx(k,\ l+1,\ q) & \text{if } k > l \text{ and } q > l, \\ 1 + Gx(k-1,\ 1,\ l) & \text{otherwise}. \end{cases}$$

Where $Gx(k, l, q)$ denotes the number of times that the algorithm *GenX-Seq* is recalled with the parameters *'k'*, *'l'*, and *'q'*. Here 1 stands for the unique generated code in each recalling of the algorithm. Since in the first time the algorithm is called with $k = n$, $l = n - 1$, and $q = 1$, therefore we can write: $Gx(k,\ l,\ 1) = 1 + Gx(k - 1,\ 1,\ l)$.

Lemma 1. For $k \geq 1$ and $l \leq k$,

$$Gx(k,\ l,\ 1) = 1 + \sum_{\substack{j=1 \\ j<k}}^{l} Gx(k-1,\ j, 1).$$

Since the third parameter $Gx(k,\ l,\ 1)$ is a constant value, therefore we denote it with G_l^k, and we have:

$$G_l^k = 1 + \sum_{\substack{j=1 \\ j<k}}^{l} G_j^{k-1},$$

where G_1^1 is equal to 1.

Now if we show that G_{n-1}^{n} is equal to C_n the verification of the algorithm is proven.

Lemma 2. For $k \geq 1$ and $l \leq k$

$$G_l^k = G_{l-1}^k + G_l^{k-1},$$

where $G_0^k = 1$.

Theorem 3. The total number of generated codes by *Gen X-Seq* is equal to C_n.

Proof: Employing Lemma 2, we can write:

$$G_l^k = \binom{k+l}{l} \frac{k-l+1}{k+1}.$$

Clearly

$$G_{n-1}^{n} = C_n = \frac{1}{n+1}\binom{2n}{n}. \qquad \square$$

In order to generate C_n sequences, the algorithm is repeated C_n times, and each time one code is generated. Therefore the algorithm generates each sequence in constant average time $O(1)$.

5.0 RANKING AND UNRANKING ALGORITHMS

To represent a binary tree as an integer, we need to know its index with respect to the generation scheme of the procedure *GenX-Seq*. This is achieved by the ranking algorithm. The ordering of the generation is according to the pre-order traversal of the recursion binary tree $\mathbf{T_n}$. This recursion binary tree can be transformed to an equivalent recursion tree. The equivalent recursion tree $\mathbf{T'_n}$ to the recursion binary tree $\mathbf{T_n}$ for $n = 4$ is illustrated in Fig. 5.

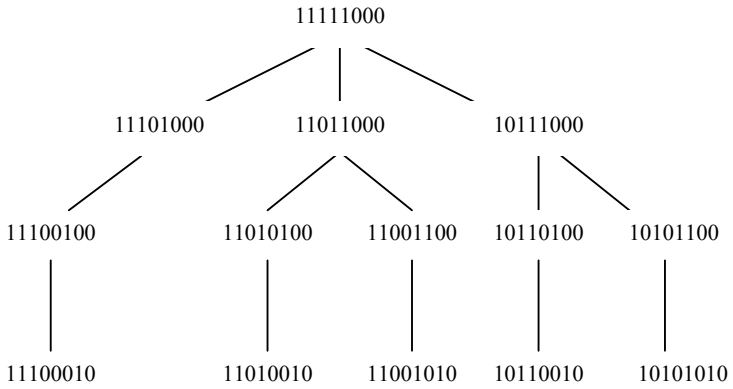

Fig. 5: The recursion tree T'_n transformed from T_n, for $n = 4$

Clearly, pre-order traversal of the recursion binary tree is the same as the depth-first search of the equivalent recursion tree, here moving to the left children of the recursion tree $\mathbf{T_n}$ is equivalent to moving down on the levels of recursion tree $\mathbf{T'_n}$, and moving to the right children is equivalent to moving on the adjacent subtrees. The first level of the tree shows all possible right shifts of the first zero, and the second level shows all possible right shifts of the

second zero, and so on. Clearly, the number of nodes in the *j*th level of tree $\mathbf{T'_n}$ is equal to the number of nodes in the *j*th subtree of $\mathbf{T'_n}$.

In order to compute the rank of a tree, we count the number of generated trees before this tree. From Lemma 2, G_0^k is equal to 1, therefore

$$G_l^k = \sum_{\substack{j=0 \\ j<k}}^{l} G_j^{k-1} ,$$

and

$$C_n = G_{n-1}^n = \sum_{j=0}^{n-1} G_j^{n-1} .$$

By the definition of G_l^k and considering $\mathbf{T'_n}$, it is clear that G_l^k counts the number of sequences beginning with *k* - *l* + 1 consecutive ones. Therefore we can easily observe that $G_0^{n-1} = 1$ counts the root of $\mathbf{T'_n}$ and G_j^{n-1} $(1 \le j \le n-1)$ counts the number of nodes in the *j*th subtree of $\mathbf{T'_n}$. Let m_i denotes the number of ones in the right side of the *i*th zero of any sequence. The numbers $(m_1, \cdots, m_n)$ would be used for computing the rank of a tree corresponding to the sequence. Any tree code appears in the m_1 th subtree of $\mathbf{T'_n}$ and $\sum_{j=0}^{m_1-1} G_j^{n-1}$ would show the number of nodes of generated codes in the previous subtree. If we consider the m_1 th subtree as an independent tree then m_2 will show that the tree code has appeared in the m_2 th subtree in the tree and recursively $\sum_{j=0}^{m_2-1} G_j^{n-2}$ will show the number of generated codes before this subtree and so on. With regard to the value of *r*, the values of m_i can be obtained. Consequently we can write:

$$r = 1 + \sum_{i=1}^{n-1} \sum_{j=0}^{m_i-1} G_j^{n-i} .$$

Using Lemma 2, we have $\sum_{j=0}^{m_i-1} G_j^{n-i} = G_{m_i-1}^{n-i}$, and we can write :

$$r = 1 + \sum_{i=1}^{n-1} G_{m_i-1}^{n-i} .$$

Using the above formula, an algorithm for computing the rank of a tree sequence can be implemented to run in time *O*(*n*). This algorithm is given in Fig. 6. It is assumed that the values of G_l^k $(1 \le k , l \le n)$, are computed in advance and stored in a two dimensional array.

The unranking algorithm given in Fig. 7 performs the reverse operations of the ranking algorithm. For a given *r* as a rank, the corresponding sequence $X = (x_1, \cdots, x_{2n})$ is computed by the following operations: initially all x_i's $(1 \le i \le 2n)$ are set to zero, and we let *m* = *n* - 1, *k* = *n,* and for any $i (1 \le i \le 2n)$ the values of G_m^k are compared with *r* and if *r*> G_m^k then x_i is set to 0 and *r* is changed to *r* - G_m^k and *k* is decremented, otherwise x_i is set to 1 and *m* is decremented. Considering the above discussion, the complexity of the unranking algorithm is *O*(*n*).

6.0 CONCLUSION

An efficient recursive algorithm for the generation of the 0-1 sequences in B-order is presented. The algorithm has two underlying recursions and a binary recursion tree presents the construction process of the generation algorithm. The maximum depth of the underlying recursion tree in the algorithm is *O*(*n*), therefore the stack space required for the algorithm is also *O*(*n*). Each node of the recursion tree shows a sequence generation and the total number of

nodes in the recursion tree is equal to C_n. Therefore, the algorithm generates each sequence in constant average time $O(1)$. The time complexity of the ranking and unranking algorithms is $O(n)$.

```
Function Rank (X: Xseq ) : Integer ;
Var r, m, k, i : Interger ;
Begin
        r := 0 ; k := n ;
        i := 1 ; m := n − 1 ;
        While (i ≤ 2n) And (m ≥ 0) Do Begin
            If (x_i = 1) Then
                 m := m − 1 ;
            Else Begin
                    r := r + G_m^k ;
                    k := k − 1 ;
        End ;
        i := i + 1 ;
    End ;
    Rank := r + 1 ;
End;
```

Fig. 6: Rank algorithm

```
Function Unrank (r : Integer ) : Xseq ;
Var X : Xseq ; k , m , i : Integer ;
Begin
        For i := 1 To 2n Do
            x_i := 0 ;
        i := 1 ;  m := n − 1 ;  k := n ;
        While (i ≤ 2n) And (m ≥ 0) Do Begin
            If (r > G_m^k) Then Begin
                    x_i = 0 ;
                    r := r − G_m^k ;
                    k := k − 1 ;
            End
            Else Begin
                    x_i := 1 ;
                    m := m − 1 ;
            End :
            i := i + 1 ;
        End ;
        Unrank :=X ;
End ;
```

Fig. 7: Unrank algorithm

REFERENCES

[1] D. K. Gupta, "A Note on The Generation of Binary Trees". *International Journal of Computer Mathematics*, Vol. 48, 1993, pp. 149-152.

[2] J. Lucas, D. Roelants Van Baronaigien and F. Ruskey, "On Rotations and the Generation of Binary Trees". *Journal of Algorithms*, Vol. 15 No. 3, 1993, pp. 343-366.

[3] J. Pallo and R. Racca, "A Note on Generating Binary Tree in A-order and B-order". *International Journal of Computer Mathematics*, Vol. 18, 1985, pp. 27-39.

[4] D. Roelants Van Baronaigien, "A Loopless Algorithm for Generating Binary Trees Sequences". *Information Processing Letters*, Vol. 39 No. 4, 1991, pp. 189-194.

[5] D. Rotem and Y. L. Varol, "Generation of Binary Trees from Ballot-sequences". *Journal of the ACM*, Vol. 25 No. 3, 1978, pp. 396-404.

[6] F. Ruskey and T. C. Hu, "Generating Binary Tree Lexicographically". *SIAM Journal on Computing*, Vol. 6 No. 4, 1977, pp. 745-758.

[7] V. Vajnovszki, "On The Loopless Generation of Binary Tree Sequences". *Information Processing Letters*, Vol. 68 No. 3, 1998, pp. 113-117.

[8] S. Zaks, "Lexicographic Generation of Ordered Tree". *Theoretical Computer Science*, Vol. 10 No. 1, 1980, pp. 63-82.

[9] H. Ahrabian and A. Nowzari-Dalini, "On the Generation of Binary Trees, from (0-1) Codes". *International Journal of Computer Mathematics*, Vol. 69, 1998, pp. 243-251.

[10] D. Rotem, "On a Correspondence Between Binary Tree and Certain Type of Permutation". *Information Processing Letters*, Vol. 4 No. 3, 1975, pp. 58-61.

[11] D. E. Knuth, *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*, (2^{nd} ed.), Reading, MA, Addison-Wesley, 1973.

[12] B. Bultena and F. Ruskey, "An Eades-McKay Algorithm for Well-Formed Parentheses String". *Information Processing Letters*, Vol. 68 No. 5, 1998, pp. 255-259.

[13] C. Germain and J. Pallo, "Two Shortest Path Metrics on Well-Formed Parentheses Strings". *Information Processing Letters*, Vol. 60 No. 6, 1996, pp. 283-287.

[14] M. C. Er, "A Note on Generating Well-Formed Parentheses String Lexicographically". *The Computer Journal*, Vol. 26 No. 3, 1983, pp. 205-207.

BIOGRAPHY

Hayadeh Ahrabian is an Associate Professor of the Department of Mathematics and Computer Science, University of Tehran and is currently the Director of Graduate Studies and Director of the Computer Science Group in this department. Her research interests include combinatorial algorithms, parallel algorithms, DNA computing, and genetic algorithms.

Abbas Nowzari-Dalini received his Ph.D. in computer science from the University of Tehran in 2004. He has been a lecturer in the Department of Mathematics and Computer Science, University of Tehran for nearly 10 years. His research interests include combinatorial algorithms, parallel algorithms, DNA computing, neural networks, and computer networks.