Thus the rank of a sequence would be computed by the following formula:

$$r = 1 + \sum_{i=1}^{n-1} \sum_{j=1}^{m_i} S_j^{n-i+1}$$

The unrank algorithm will perform the reverse operations of the rank algorithm. For a given number r as a rank, the corresponding sequence $B_T = (b_1, b_2, \ldots, b_n)$ is computed by the following operations: Initially all b_i's $(i = 1, \ldots, n)$ are set to zero. Later r is compared with S_j^{n+1}'s $(1 \le j \le n)$ and we get maximum j such that S_j^{n+1} which is less that r. When this j is found, each element of the sequence from b_1 to b_j are added by 1. Later $r = r - S_j^{n+1}$ is computed and we get next maximum j such that $S_j^n < r$ and each element of b_1 to b_j are added by 1, and we set $r = r - S_j^n$. These operations are repeated untill in computing any b_i, we can not find a j such that $S_j^{n+2-i} < r$.

Similar to P-sequences, the time complexity of ranking and unranking algorithms for Ballot-sequences is also $O(n^2)$.

4-Conclusion

New recursive algorithms for the generation of the P-sequences and Ballot-sequences are presented. These algorithms have same construction and produce isomorphic recursion trees with the same properties. Each node of the recursion tree shows a sequence generation and the number of nodes in each recursion tree is equal to C_n. Therefore, the algorithms generate each sequence in constant average time $O(1)$. The time complexity of ranking and unranking algorithms presented for both sequences is $O(n^2)$.

References

[1] H. Ahrabian and A. Nowzari-Dalini, On the generation of binary trees, from (0-1) codes, Intern, J. of Comput. Math., **69** (1998), 243-251.

[2] H. Ahrabian and A. Nowzari-Dalini, On the generation of binary trees in A-order, Intern. J. of Comput. Math., **71** (1999), 1-7.

[3] D. K Gupta, On the generation of P-sequences, Intern. J. of Comput. Math., **38** (1991), 31-35.

[4] D. E. Knuth, The art of computer programming, Vol. 3: Sorting and searching, Addison-Wesley, Reading. Mass., 1973.

[5] J. Pallo and R. Racca. A note on generating binary tree in A-order and B-order, Intern, J. of Comput. Math., **18** (1985), 27-39.

[6] D. Rotem, On a correspondence between binary tree certain type of permutation, Iform. Processing letters, **4** (1975), 58-61.

[7] D. Rotem and Y. L. Varol, Generation of binary trees from Ballot-sequences, J. ACM, **25** (1978). 396-404.

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

[9] S. Zaks, Lexicographic generation of ordered tree, Theoret. Comput. Sci., **10** (1980), 63-82.

modification of the corresponding results for P-sequence, and the ideas behind their proofs are essentially identical.

As it is mentioned before, there is a close connection between a P-sequence $P_T = (p_1, p_2, \dots, p_n)$ and a Ballot-seqeunce $B_T = (b_1, b_2, \dots, b_n)$, such that for any sequence we have $b_j + p_j = n$. For example, the Ballot-sequence for the tree given in Figure 1 would be equal to $B_T = (4, 4, 4, 2, 2, 0, 0)$.

The generation algorithm of the Ballot-sequences GenBallot-Seq in Figure 5 is constructed by modifying one instruction in the algorithm GenP-Seq. This procedure is also called with three parameters $'Bs'$, $'n'$ and $'n-1'$, where $'Bs'$ is an array which holds the sequence and initially is equal to (0,0,...,0), and $'n'$ is the number of nodes. It should be noted that the generation of two sequences are performed independently, without considering their connection. This algorithm produces each code by incrementing the elements of $'Bs'$ by 1, from the leftmost element, one by one.

The recursion tree of the generation algorithm of these sequences is isomorphic to the tree given in Figure 3 with different labeling. As it illustrated, incrementing of an adjacent element of a sequence causes moving to the next sibling of the recursion tree and incrementing of the same element causes going down on the level of the recursion tree. Therefore, the discussion of the verification and the analysis of this algorithm would be similar to the verification and analysis of the GenP-Seq. Consequently, the number of nodes of the recursion tree implicitly constructed by algorithm for generating n-nodes binary trees in equal to C_n, and each code is generated in constant average time $O(1)$.

```
Procedure GenBallot-Seq ( $Bs$ : Bseq ; $k, l$ : Integer ) ;
Var
    $j$ : Integer ;
Begin
    WriteBseq ( $Bs$ ) ;
    If ( $k > 1$ ) Then Begin
        $j := 1$ ;
        While ( $j <= l$ ) And ( $j < k$ ) Do Begin
            $Bs[j] := Bs[j] + 1$ ;
            GenBallot-seq ( $Bs, k-1, j$ ) ;
            $j := j + 1$ ;
        End ;
    End ;
End;
```

Figure (5) Ballot-sequences generation algorithm.

3-2- Ranling and Unranking for Ballot-Sequences

Since, the recursion tree of this algorithm is isomorphic to the recursion tree of the procedure GenP-Seq therefore the number of nodes in each subtree of both trees are the same. S_l^k 's $(k, l = 1, \dots, n+1)$ are also should be employed for the rank and unrank algorithms in this section.

As we can see, all the sequences in the first subtree of the recursion tree have $(n\text{-}1)$ zeros and all the sequences in the second subtree have $(n-2)$ zeros and consequently all sequences in $(n-1)$ th subtree have one zero. Therefore, for a given sequence $B_T = (b_1, b_2, \dots, b_n)$ we will compute a new sequence $m = (m_1, m_2, \dots, m_{n-1})$ where m_i shows the number of elements in B_T which are greater or equal than i. Therefore m_1 will show the number of non-zero elements in the code and shows the subtree that code is appeared. Therefore $\sum_{j=1}^{m_1} S_j^{n-1+1}$ would be the number.of codes generated before this subtree. Now, if we consider m_1th subtree as an independent tree, respectively the code would appear in the m_2th subtree of this new tree.

expanded recursively for all the subtrees. Figure 4 shows all the values of S_j^n for $1 \le n, j \le 5$. By the above results, we can see that the number of nodes in each subtree of the recursion tree in Figure 3 is equal to $S_2^4 = 3, S_3^4 = 5$ and $S_4^4 = 5$.

n	j=1	j=2	j=3	j=4	j= 5
1	1				
2	1	1			
3	1	2	2		
4	1	3	5	5	
5	1	4	9	14	14

Figure (4) The values of S_j^n for *n* and *j*= 1, 2, 3, 4, 5.

Now, for computing the rank of a tree, by employing the above results, it is enough to specify the position of its corresponding code in the recursion tree. The position of a code, corresponds to the position of the corresponding node in the recursion tree. In order to specify the position of the following code $P_T = (p_1, p_2, \ldots, p_n)$, the difference sequence, $(m_1, m_2, \ldots m_{n-1})$, where $m_i = n - p_i (1 \le i \le n-1)$ is computed. The code appears in m_1th subtree of the recursion tree and $\sum_{j=1}^{m_1} S_j^n$ show the number of generated codes in the previous subtrees including the root of the tree. If we consider the m_1th subtree as an independent tree, then m_2 will show that the tree code has appeard in m_2th subtree of this tree and recursively $\sum_{j=1}^{m_2} S_j^{n-1}$ will show the number of generated codes before this subtree and so on. Consequently we can write:

$$r = 1 + \sum_{i=1}^{n-1} \sum_{j=1}^{m_i} S_j^{n-i+1}$$

Using above formula an algorithm for computing the rank of a tree sequence can be implemented to run in time $O(n^2)$.

The unrank algorithm, takes *r* in the range of $1 \ldots C_n$ as input and returns the P-sequence $P_T = (p_1, p_2, \ldots, p_n)$ of the related binary tree. The unranking algorithm essentially reverses the steps carried out in computing the rank. According to the rank of a tree, the position of its sequence in the recurrence tree, the position of its sequence in the recurrence tree, is specified. The position of a sequence in the recurrence tree, depends on the number of times that each element is decremented. Therefore, this position is obtained by using S_l^k's $(1 \le k, l \le n+1)$. In the unranking algorithm, the value of all p_i's ($i = 1, \ldots, n$) are initially assigned to *n*. In the next step, the maximum *j* for which $S_j^{n+1} < r$ is found. Then p_1 is decremented by *j*, and later $r = r - S_j^{n+1}$ is assigned. For evaluating p_2, next maximum *j* is computed such that $S_j^n < r$, and p_2 is decremented by *j*, then we set $r = r - S_j^n$. The above operations are repeated untill in computing any p_i, untill we can not find a *j* such that $S_j^{n+2-i} < r$ in conputing p_i. Considering the above discussion, the complexity of the unranking algorithm is $O(n^2)$.

3-Ballot-Sequences

3-1-The Generating Algorithm

The generation of the Ballot-sequences with a new scheme and the corresponding ranking and unranking algorithms are given in this section. Many of result of this section are simple

Clearly

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

The time required for the generation algorithm can be also obtained by the previous theorem. 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 seqeuence in costatnt average time $O(1)$.

The algorithm needs stack space to implement recursion, since the ordering of the generation is according to the dept-first search of tree T_n, therefore this space would be equal to the depth of T_n. For any *n* the depth of this tree is equal to *n*. Hence this algorithm requires a stack space of $O(n)$.

2-2-Ranking and Unranking for P-Sequences

In this section we will develop two algorithms for determining the rank and unrank of any feasible sequence. To do this a class of numbers will be defined and some of its properties discussed.

In general a ranking function f, for an algorithm generating the element of some set S, is a bijection $f : S \rightarrow \{1, 2, \ldots, |S|\}$ such that $f(s) = i$ if and only if the *i*th element generated by the algorithm is s (s is an element of S), we refer to i as the rank of s. The unranking function is the inverse function of f, f^{-1}: $\{1, 2, \ldots, |S|\} \rightarrow S$ and $f^{-1}(i) = s$ such that i is an index in the range of $\{1, 2, \ldots, |S|\}$ and s is an element of S. We wish to find efficient algorithms for computing f, and f^{-1}

Here for ranking algorithm, we need to know its index with respect to the generation scheme of the procedure GenP-Seq. The ordering of the generation is according to the depth-first search of the recursion tree (for $n = 4$ denoted by Figure 3).

As it is mentioned earlier, G_{n-1}^{n} denotes the number of nodes in the recursion tree. From the previous equations we have:

$$G_{n-1}^{n} = 1 + G_1^{n-1} + G_2^{n-1} + \cdots + G_{n-1}^{n-1},$$

where 1 counts the root of the recursion tree and $G_j^{n-1} (1 \le j \le n-1)$ counts the number of nodes in the *j*th subtree of the recursion tree. For computing the rank of a tree, we count the number of generated trees before this tree, therefore we define:

$$S_l^k = \begin{cases} 0 & \text{for } \; l > k, \\ 1 & \text{for } \; l = 1, \\ G_{l-1}^{k-1} & \text{for } \; k > 1 \text{ and } l \le k. \end{cases}$$

Lemma. For $k > 1$ and $l \le k$,

$$S_l^k = \sum_{j=1}^{l} S_j^{k-1}$$

Clearly $C_n = G_{n-1}^{n} = S_n^{n+1} = \sum_{j=1}^{n} S_j^n$. By definition of G_l^k and by considering the recursion tree, we can easily observe that S_j^n for $2 \le j \le n$ counts the number of nodes in the $(j-1)$th subtree of the recursion tree, and S_1^n denotes the root of recursion tree. This relation can be

```
Procedure GenP-Seq ( $Ps$ : Pseq ; $k, l$ : Integer ) ;
Var
    $j$ : Integer ;
Begin
    WritePseq ( $Ps$ ) ;
    If ( $k > 1$ ) Then Begin
        $j := 1$ ;
        While ( $j <= l$ ) And ( $j < k$ ) Do Begin
            $Ps[n-k+1] := Ps[n-k+1] - 1$ ;
            GenP-seq ( $Ps, k-1, j$ ) ;
            $j := j + 1$ ;
        End ;
    End ;
End;
```

Figure (2) P-sequences generation algorithm in the B-order.

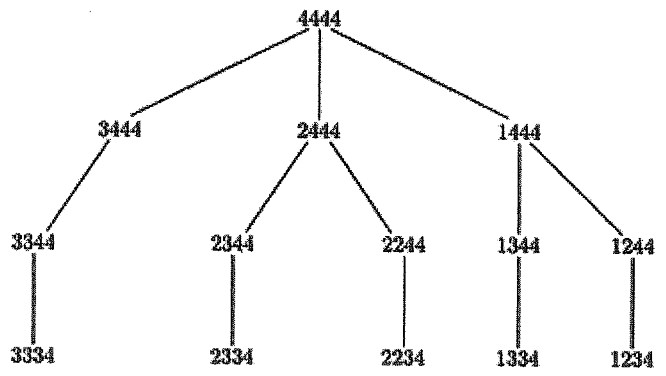

Figure (3) The recursion tree of the algorithm Gen P-Seq for $n = 4$.

Now, the verification and the analysis of the algorithm are discussed. Obviously, the number of nodes of the recursion tree is the number of times that GenP-Seq is recalled and in each recall the nest sequence is generated. With regard to the generation algorithm GenP-Seq, the number of times that GenP-Seq is recalled can be obtained from following recurrence formula:

$$G_l^k = 1 + \sum_{j=1}^{l} G_j^{k-1} \quad \text{for} \quad j \le k-1,$$

where G_l^k denotes the number of time that algorithm GenP-Seq is recalled with the parameters $'Ps', 'k'$ and $'l'$. Here, 1 stands for the unique generated code in each recalling of the algorithm GenP-Seq, and G_l^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.

Theorem. The total number of codes generated by GenP-Seq is eual to C_n.
Proof. Recall from [1], we can write:

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

Similar to Rotem's algorithm [7], this algorithm will also generate all Ballot-sequences of the same length in their lexicographic order, where the rightmost digit is the most significant one. We will call this order Ballot-order.

Each Ballot-sequence is obtained from P-sequence by replacing the integers n, n-1,...,2,1 with 0,1,...,n-1 in the corresponding places. Therefore, the Ballot-sequence of a binary tree T is an integer sequence $B_T = (b_1, b_2, \ldots, b_{|T|})$, where b_i is the number of unvisited internal nodes during visiting the ith external node in the preorder traversal of T. In spite of the mentioned connection the presented algorithms generate each sequence independently, without considering this connection.

For both sequences, P-sequences and Ballot-sequences ranking and unranking algorithm with $O(n^2)$ time complexity are also presented.

2-The P-Sequences

2-1-The Generating Algorithm

In this section, an algorithm for the generation of P-sequences is presented. Clearly, for the given binary tree T denoted in Figure 1, the P-sequence of T would be the integer sequence

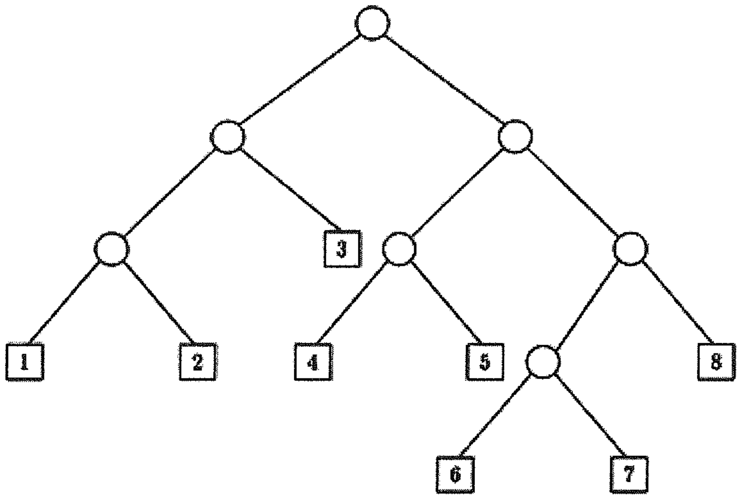

Figure (1) Sample tree.

$P_T = (3,3,3,5,5,7,7)$, where each integer in position i shows the number of visited internal nodes before the ith external node in a preorder traversal.

The algorithm Gen P-Seq given in Figure 2, generates these sequences in the B-order. This algorithm is recursive and is called with three parameters: $'Ps'$, $'n'$ and $'n\text{-}1'$, where $'Ps'$ is an integer array of size n and is equal to $(n,\ldots,n)$ initially, and $'n'$denotes the number of nodes.

The algorithm produces each code by decrementing the elements of $'Ps'$ by 1, from the leftmost element, one by one. The generation starts from $Ps = (n,\ldots,n)$ and all the elements of $'Ps'$ become n-1 except the nth element, then the decrementation restarts from the beginning. The construction process for $n = 4$ is demonstrated in Figure 3 by a recursion tree. As we see, the decrementation of the adjacent elements in a sequence causes downwords move on the levels of the recursion tree and decrementation of the same element causes a move on the adjacent subtrees.

Generating Binary Trees in B-Order and Ballot-Order

H. Ahrabian

Department of Mathematics and Computer Science, Faculty of Science, University of Tehran

A. Nowzari- Dalini

Department of Mathematics and Computer Science, Faculty of Science, University of Tehran

Abstract

We use two recent codings of binary trees by integer sequences called P-sequences and Ballot-sequences in order to generate binary trees in B-order and Ballot-order. These algorithms generate each sequence in constant average time O(1). The ranking and unranking algorithms for both sequences with O(n^2) time complexity are described.

Keywords

Binary tree, Recursion, B-order, P-sequences, Ballot-sequences.

C.R. CATEGORIES: G.2.1 Combinatorial Algorithm, G.2.2 Graph Theory (Trees).

1-Introduction

There is a considerable interest in the computer science community about exhaustive generation of binary trees. In generating trees, a desirable goal is to keep the running time required to create the representation of two consecutives trees constant. There exists several sequential algorithm on the generation of binary trees with a fixed number of nodes in recent years [2, 3, 5, 8, 9]. In some of these algorithms binary trees are represented by integer sequences and the corresponding sequences are generated.

P-sequences introduced by Pallo and Racca [5], are integer sequences characterizing all shapes of n-noded binary trees. The P-sequence of a binary tree T is an integer sequence $P_T = (p_1, p_2, \ldots, p_{|T|})$ where p_i is the number of visited internal nodes before the ith external node in the preorder traversal of T, and $|T|$ denotes the number of internal nodes in the binary tree, it is noted that the assigned value to the $|T|+1$ external node is always equal to $p_{|T|} = n$ and therefore it is discarded. We present here an efficient algorithm that generates each sequence in constant average time $O(1)$. The given algorithm is based on the ideas of the previous algorithm presented by the above authors, which generates the bit representation of binary trees, with adjacent interchange [1]. The corresponding binary trees to the sequences, which our algorithm generate, are in B-order. Recall from [5], given two trees T and T' with n nodes, $T < T'$ are in B-order if P_T is lexicographically less that $P_{T'}$ (for the other B-order generation algorithms see [8,9]).

The next sequences which characterize the binary trees are Ballot-sequences presented by Rotem [6, 7]. Ballot-sequences are some of the entries of the inversion table [4], that must satisfy specific properties. The inversion table of a permutation π on $N = \{1, 2, \ldots, n\}$ is taken to mean an n-tuple $B = (b_1, b_2, \ldots, b_n)$, where b_i is the number of elements in π which are greater than i and appear on its right. Clearly, each entry of the inversion table must satisfy $0 \le b_i \le n-1$. In addition to that, if $b_i \ge b_{i+1}$ for $1 \le i \le n-1$, then B is called a Ballot-sequence [6]. Another algorithm is presented that generates each Ballot-sequence in constant average time. This algorithm is based on the ideas of the P-sequences generation algorithm.