GENERATION OF BINARY TREES FROM BALLOT SEQUENCES*

by

D. Rotem[†]

and

Y. Varol[††]

Research Report CS-77-29

Department of Computer Science

University of Waterloo
Waterloo, Ontario, Canada
October 1977

[†] Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

[††] Department of Applied Mathematics
University of Witwatersrand
Johannesburg

* The results of this paper are contained in the first author's doctoral thesis.

A B S T R A C T

An efficient algorithm for generating and indexing all shapes of n-noded binary trees is described. The algorithm is based on a correspondence between binary trees and the class of stack-sortable permutations, together with a representation of such permutations as ballot-sequences. Justification for the related procedures is given, and their efficiency established by comparison to other approaches.

Key Words and Phrases: binary tree, stack sortable permutation, ballot sequence, tree generation, tree indexing.

1. Introduction

This paper describes an efficient algorithm to generate all 'shapes' of n-noded binary trees. Such algorithms are used in investigating and comparing various deletion schemes in binary trees [2], and can be effectively employed for systematic generation of combinatorial objects which are in 1-1 correspondence with such trees [6,pp.154] .

The algorithm is based on a correspondence between binary trees and the class SS_n of stack-sortable permutations, together with a representation of such permutations as ballot-sequences [5]. Initially, a ballot-sequence of length n is generated. This is then used to construct a binary tree. It is shown that if a ballot-sequence is an inversion table of $\Pi \in SS_n$, then the algorithm generates T_π, the binary tree constructed by the following well known procedure.

Construction-T: given $\Pi = \langle p_1, p_2, \ldots, p_n \rangle$, assign p_1 to the root of tree T; for each p_k, $k=2,3,\ldots,n$, apply the rule -- if p_k is to be inserted into a non-empty subtree rooted by p_i, it must be on the left subtree of p_i if and only if $p_k < p_i$, otherwise p_k must appear to the right of p_i -- until an empty subtree is reached, and then create a root to that subtree and assign the label p_k to it.

It is well known that Construction—T is invertible and establishes a 1-1 correspondence between SS_n and the set of binary trees of order n [4; See 6.2.2]. By generating all ballot-sequences of length n in their lexicographic order, all $c_n = (n+1)^{-1}\binom{2n}{n}$ 'shapes' of binary trees of order n can be obtained.

A unique integer , $num_n(B)$, between 1 and C_n , is associated with each ballot-sequence B of length n , by using some combinatorial properties of such sequences. The lexicographic order is preserved by this association , namely , for any two ballot-sequences B and B' , if B precedes B' then $num_n(B)<num_n(B')$.

A simple recursion relation , is used both in computing $num_n(B)$ from a given B , and its inverse $num_n^{-1}(m)$ from a given integer $m<C_n$. This provides the capability of storing a binary tree of n nodes as an integer smaller than C_n , as well as efficient generation of a random binary tree.

In what follows , the <u>inversion-table</u> of a permutation Π on $N=\{1,2,..,n\}$, is taken to mean an n-tuple $B=<b_1,b_2,...,b_n>$ where for $1\leq i\leq n$, 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 $0\leq b_i\leq n-1$. In addition to that, if $b_i\geq b_{i+1}$ for $1\leq i\leq n-1$, then B is called a <u>ballot-sequence</u> [5] .

A path $<u=a_0,a_1,...,a_\ell=k>$ between nodes u and k in a binary tree T , is denoted by $P_T(u,k)$. This path can be characterized as a product $\alpha_1\cdot\alpha_2\cdot\ldots\cdot\alpha_\ell$ where α_i specifies the relation (left son , right son , left father , right father) between two adjacent nodes a_{i-1} and a_i of the tree. Two paths , $P_T(u,k)=\alpha_1\cdot\alpha_2\cdot\ldots\cdot\alpha_\ell$ and $P_T(r,s)=\alpha'_1\cdot\alpha'_2\cdot\ldots\cdot\alpha'_\ell$ are similar if and only if $\alpha_i=\alpha'_i$ for $1\leq i\leq\ell$. Similarity between two binary trees T and T' with respective roots r and r' , can be defined in terms of paths as follows; T and T' are similar if and only if there is a 1-1 correspondence between their nodes , such that if node u corresponds to u' of T' , then $P_T(r,u)=P_{T'}(r',u')$. It is easy to check that this definition is equivalent to the standard one [3; pp:325].

2. Generation

The generation of the ballot-sequence, and the construction of the binary tree, are given separately in two procedures. Procedure BALLOT generates the next ballot-sequence from the previous one in an array B, and then calls procedure TREE which converts the ballot-sequence into a difference-sequence and constructs the corresponding binary tree. It is possible to generate the difference-sequence directly or to combine the two algorithms below and obtain a marginally faster algorithm. However, this would make the presentation and the subsequent analysis more cumbersome. Algol-like descriptions of the two algorithms with some explanatory notes are given below.

Ballot-sequences

Successive applications of this procedure will generate all ballot-sequences of length n in B, in their lexicographic order, where the rightmost digit is the most significant one. Note that by definition B[n] is always 0, and the values of an entry B[i] range from B[i+1] up to n-i. A new sequence is generated from the previous one by changing only those entries which differ from one sequence to the next.

```
procedure BALLOT ( n,B,MATREE ) ;
integer n; integer array B, MATREE;
begin comment  generate the next ballot-sequence of order n
in B[1],...,B[n], and then call procedure TREE to construct
the corresponding tree in array MATREE. m is a pointer to the
element last changed in B.  first is a global variable which
is initially set to true, it is assigned the value false by the
first entry to BALLOT and remains so until all ballot-sequences
have been generated.;
integer i,k,x; own integer m ;
if first then begin for i:=1 to n do B[i]:=0 ;
                    m:=1 ;
                    first:=false ;
                    go to call ;
              end ;
newballot : B[m]:=B[m]+1 ;
            if B[m] > n-m then begin m:=m+1 ;
                                     if m≠n then go to newballot;
                               end ;
                          else if m=1 then go to call;
                                     else begin k:=m-1;
                                                x:=B[m] ;
                                                for i:=1 to k
                                                do B[i]:=x ;
                                                m:=1; go to call;
                                          end ;
stop : first := true ; go to exit ;
call : TREE ( n,B,MATREE ) ;
exit : end of procedure BALLOT ;
```

Binary trees

Given a ballot-sequence we consider its difference-sequence as the array D , where

$$D[1]=n-B[1] \quad \text{and for } 2\le i\le n \quad D[i]=B[i-1]-B[i] \ . \qquad (1)$$

It follows from the definitions of D and B that for $1\le i\le n$ $D[i]\ge 0$ and $\sum_{i=1}^{n} D[i] = n$.

Initially , a pointer P points to a dummy root labelled O . The array D is scanned from D[1] to D[n] ,and P always points to a node labelled i-1 when D[i] is being processed . This processing depends on the value of D[i] .

If $D[i]=j>0$, then j new nodes $g_1,g_2,\ldots,g_j$ are created , such that g_1 is the right son of node i-1 and for $1\le \ell\le j-1$, $g_{\ell+1}$ is the left son of g_ℓ . Each node except the last one is pushed into a stack after it is created , the last node g_j is assigned the label i and P is made to point to it. In other words , the path $P_T(i-1,i)$ of Figure 1 is constructed and the j-1 unlabelled $g_1,g_2,\ldots,g_{j-1}$ are added to the stack.

If D[i]=O , then the node on top of stack is removed and assigned the label i , again , P now points to i.

In this way , all entries of D are processed to obtain a labelled binary tree of n nodes. If only the 'shape' matters, we may stop even earlier , as soon as n nodes are generated. The actual root of the generated tree will always be the right son of the dummy root.

Since the entries of D are processed one at a time they are computed immediately prior to their use. During the construction the nodes are labelled from 1 to n in symmetric order (postorder). This labelling is not essential to the algorithm but facilitates the proof of its validity given in Theorem 1.

```
procedure TREE ( n,B,MATREE ) ;
integer n ; integer array B, MATREE ;
begin comment  construct in MATREE, the tree corresponding to
 the ballot-sequence given in B.  MATREE is an array of dimen-
 sion [1:n+1,1:3].  For a given index j, MATREE[j,1] and
 MATREE[j,3] are respective pointers (row indexes) to the left
 son and right son of the node whose label is in MATREE[j,2].
 Entries in the first and third columns of MATREE are initially
 assigned special values to represent grounded links.  avail is
 a pointer to the next row in MATREE which has not been labelled
 nor reserved by having its address pushed into the stack.  On
 exit from the procedure, the second row of MATREE will represent
 the root of the generated tree.  ;
 integer i,j,s,p,avail  ;  integer array STACK[1:n] ;
 p:=1 ; s:=1 ; avail:=2 ; MATREE[1,2]:=0 ;
 for i:=1 to n
 do begin if i=1 then j:=n-B[1] ;
                 else j:=B[i-1] - B[i]  ;
```

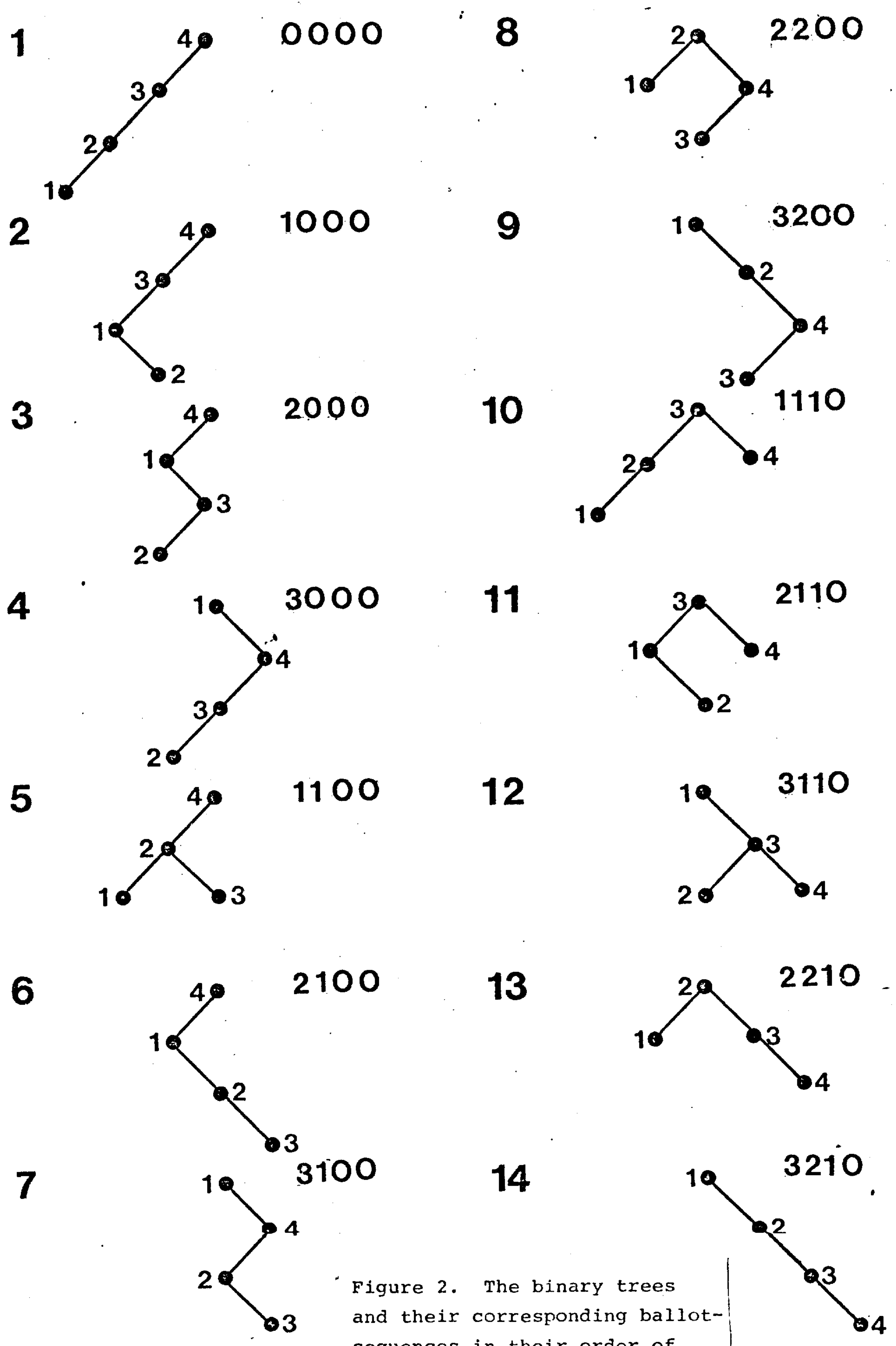

Figure 2. The binary trees and their corresponding ballot-sequences in their order of generation by procedure BALLOT.

```
if j=0 then begin s:=s-1 ; p:=STACK[s] ; end ;
       else if j=1 then begin MATREE[p,3]:=avail ;
                             p:=avail; avail:=avail+1 ;
                             end ;
                  else begin MATREE[p,3]:=avail ;
                             p:=avail; STACK[s]:=p ;
                             s:=s+1; avail:=avail+1;
                       loop: j:=j-1; MATREE[p,1]:=avail;
                             p:=avail; avail:=avail+1;
                             if j≠1 then begin STACK[s]:=p;
                                                s:=s+1 ;
                                                go to loop;
                                         end;
                             end ;
MATREE[p,2]:=i ;
end ;
end of procedure TREE ;
```

As an example , all binary trees of order 4 are shown in Figure 2 in their order of generation.

Theorem 1 Let $\Pi \in SS_n$ have an inversion-table B . Then TREE constructs from B a binary tree T_b , such that $T_b = T_\pi$.

Proof For convenience let the number 0 be added to Π as its leftmost member, and let us consider the generated tree T_b with its dummy root. We have to prove that $P_{T_b}(0,i) = P_{T_\pi}(0,i)$ for $1 \le i \le n$.

First let us show that $P_{T_b}(i-1,i)=P_{T_\pi}(i-1,i)$ for the case when $i-1$ is to the left of i in Π. According to the construction of T_π, any element between i and $i-1$ which is smaller than $i-1$ will not appear on the path $P_{T_\pi}(i-1,i)$. Therefore ,consider $G=\langle g_1,g_2,\ldots,g_j=i\rangle$, the subsequence of all elements greater than $i-1$ which appear between them in Π. If $j\le 2$, then G is trivially a decreasing subsequence . For $j>2$, suppose that G is non-decreasing , then we can find two elements $g_k<g_\ell$ with $k<\ell$, and this in turn implies that Π contains a subsequence

$$g_\ell<g_k<g_j=i \quad \text{and} \quad k<\ell<j$$

which would contradict the fact that $\Pi\in SS_n$ [3 , pp:239] . Thus G is always decreasing. Therefore $P_{T_\pi}(i-1,i)$ will have the 'shape' shown in Figure 1 , which is also the 'shape' of $P_{T_b}(i-1,i)$, since by definition $j=D[i]$.

The proof now proceeds by induction on i. As a special case $P_{T_\pi}(0,1)=P_{T_b}(0,1)$. Assume that

$$P_{T_\pi}(0,i-1)=P_{T_b}(0,i-1) \ . \qquad (2)$$

When $i-1$ is to the left of i in Π , then $P_{T_\pi}(i-1,i)=P_{T_b}(i-1,i)$. This together with the induction hypothesis (2) gives the desired result. When $i-1$ is to the right of i in Π, we need to show that node i is placed at corresponding points on the paths from 0 to $i-1$ by both constructions. Clearly, in T_π, $i-1$ is the rightmost node in the left subtree of i. Namely , the path $P_{T_\pi}(i-1,i)$ has the shape given in Figure 3. Note that in this case $D[i]=0$. Therefore i is assigned to the node on top of stack, which is the last one generated before $i-1$,and having a left son. This observation combined with (2) completes the proof. ▌

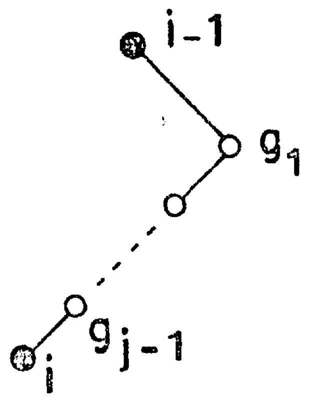

Figure 1

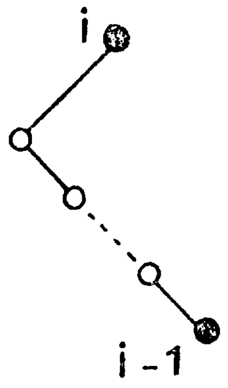

Figure 3

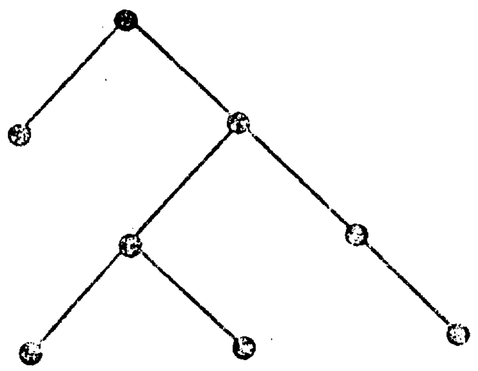

Figure 4.

3. Indexing Binary Trees

To store a binary tree T as an integer, we need to know its index with respect to the generation scheme of procedure TREE. This can be achieved by solving the following two problems:

i) find the ballot-sequence B which is the inversion-table of $\Pi \in SS_n$, such that $T_\pi = T$,

ii) find the number $num_n(B)$, of ballot-sequences of length n which precede B in the lexicographic order of procedure BALLOT.

We shall later see that the difference-sequence D is generated in the solution of both problems as an intermediary step, and therefore B does not need to be explicitly derived. However, to simplify the presentation we shall solve both problems as posed. The solution to the first one is an algorithm which is the inverse of procedure TREE, and will be illustrated by an example. Consider the tree T given in Figure 4. T is traversed in symmetric order, and the number of new nodes which are pushed into the stack before the removal of the i'th node from it is recorded as D[i]. This results in the sequence $D = \langle 2,0,3,0,1,0,1,1 \rangle$ which in fact is the difference-sequence of the ballot-sequence corresponding to T. Hence, we find $B = \langle 6,6,3,3,2,2,1,0 \rangle$, using (1). The justification of this algorithm uses the same arguments as in the proof of Theorem 1.

We now turn to the problem of finding $num_n(B)$ for a given ballot-sequence. Let S_{nk} be the set of all ballot-sequences of order n, with exactly k non-zero digits, and let $|S_{nk}|$ denote the cardinality of this set . A sequence $U_{nk} \in S_{nk}$ will be called a unit-sequence if its k non-zero digits are all equal to 1.

Lemma 1

$$|S_{nk}| = \begin{cases} 1 & \text{for } k=0 \quad (3a) \\ |S_{(n-1)k}| + |S_{n(k-1)}| & \text{for } 0<k\le n-1 \quad (3b) \\ 0 & \text{for } k=n \quad (3c) \end{cases}$$

Proof The relations (3a) and (3c) follow directly from the definitions. To prove (3b) we first show that

$$|S_{nk}| = \sum_{i=1}^{k} |S_{(n-1)i}| \quad , \qquad (4)$$

by constructing a correspondence between S_{nk} and $J=\bigcup_{i=0}^{k} S_{(n-1)i}$. Given a sequence $B=\langle b_1,b_2,\dots,b_n\rangle \in SS_n$, we define $B'=\langle b_1',b_2',\dots,b_{n-1}'\rangle$ as follows ,

$$\begin{aligned} b_i' &= b_i - 1 \qquad \text{for } 1\le i\le k \ , \\ b_i' &= b_i = 0 \qquad \text{for } k<i\le n-1 \ . \end{aligned} \qquad (5)$$

Clearly, every member of S_{nk} is mapped by (5) into a unique member of J. Conversely , given any $B'\in J$, we can find the unique $B\in S_{nk}$ associated with it using ,

$$\begin{aligned} b_i &= b_i'+1 \qquad \text{for } 1\le i\le k \\ b_i &= 0 \qquad \text{for } k<i\le n \ . \end{aligned} \qquad (6)$$

This correspondence implies (4) from which (3b) is readily derived. ▌

In the sequel the subscript of 'num' is omitted when it is applied to unit-sequences.

<u>Lemma 2</u> $\quad num(U_{nk}) = |S_{(n+1)(k-1)}| \quad$ for $0<k<n$. (7)

<u>Proof</u> By definition the sequence U_{nk} is the first ballot-sequence in the lexicographic which has k non-zero digits. Hence , the sequences which precede U_{nk} in this order , are exactly the ones which have fewer than k non-zero digits. The number of such sequences is $\sum_{i=0}^{k-1} |S_{ni}|$. The result now follows from (4). ▌

<u>Theorem 2</u> For a ballot-sequence $B=\langle b_1, b_2, \ldots, b_n \rangle$, let g_i $(1 \leq i \leq b_1)$ be the number of elements in B which are not smaller than i. Then

$$num_n(B) = \sum_{i=1}^{b_1} num(U_{(n+1-i)g_i}). \tag{8}$$

<u>Proof</u> The proof is by induction on the length of B. The result is easily verified for sequences of length 2. Assume that it holds for ballot-sequences of length n-1. Let S(X) denote the set of sequences which are generated before X by procedure BALLOT.

Since B has g_1 non-zero digits , it follows that S(B) can be partitioned into two disjoint sets $S(U_{ng_1})$ and $S(B)-S(U_{ng_1}) = G$. Therefore ,

$$num_n(B) = |S(U_{ng_1})| + |G| = num(U_{ng_1}) + |G| \quad . \tag{9}$$

Consider the ballot-sequence B' of length n-1 , which is obtained according to (5). Using the mappings of (5) and (6) , a 1-1 correspondence between G and S(B') can easily be established. Hence,

$$num_n(B) = num(U_{ng_1}) + num_{n-1}(B'). \tag{10}$$

Let there be g_i' digits which are not smaller than i in B' , then by the construction of B' and the induction hypothesis

$$g_i' = g_{i+1} \tag{11}$$

$$\mathrm{num}_{n-1}(B') = \sum_{i=1}^{b_1'-1} \mathrm{num}(U_{(n-1)g_i'}) . \tag{12}$$

The result follows by substituting (11) and (12) into (10). ▌

The two algorithms for computing $\mathrm{num}_n(B)$ from B and its inverse $\mathrm{num}_n^{-1}(m)$ from an integer $m<C_n$, are both based on Theorem 2. However , they are considerably simplified by observing that in the difference-sequence D[1],...,D[n] of B, an entry D[k] represents the number of times that g_i is equal to k-1. The entry D[1]is irrelevant in both cases. For fast performance , the algorithms make use of a pre-computed table containing the values of $\mathrm{num}(U_{ik})$ $1 \le i \le n$ and $1 \le k \le n-1$, which can be computed directly using the results of Lemmas 1 and 2.

The first procedure, NUM, does not require any explanation since it is a straightforward application of (8). The computation of the inverse in procedure INVNUM, is based on a recursive application of (10). Observe that in row n of TAB, $\mathrm{num}(U_{ng_1})$ is the maximal value smaller than m. Similarly, in row n-1, $\mathrm{num}(U_{n-1,g_2})$ is the maximal element smaller than $m-\mathrm{num}(U_{ng_1})$, and so on. Thus, after $\mathrm{num}(U_{ik})$ is found in row i, and subtracted from the argument, row i-1 is searched for the index of the maximal unit-sequence of order i-1. This process is continued until all the terms of (8) are found. Note that in any row i of TAB, the first i-1 elements constitute a non-decreasing sequence suitable for binary search, and that this search can be restricted to fewer than i-1 elements since the g_i's are non-increasing.

```
procedure NUM ( n,B,numb,TAB ) ;
integer n,numb ; integer array B, TAB ;
begin comment  compute the number of ballot-sequences preceding
B, in the generation of procedure BALLOT.  The difference-se-
quence D is not stored, but each entry in D is computed prior
to its use.  TAB is a precomputed table of dimension [1:n,1:n].
For 1≤i≤n, and 1≤k≤i-1, TAB[i,k]=num(U_ik), and only this lower
triangular portion of TAB is necessary.  ;
integer i,k,j,q  ;
i:=n ; numb:=0 ;
for k:=n-1 step -1 to 1
    do begin j:=B[k]-B[k+1] ;
              if j=0 then go to loop ;
                     else for q:=1 to j  do
                              begin numb:=numb+TAB[i,k] ;
                                    i:=i-1 ;
                              end ;
    loop: end ;
end of procedure NUM  ;
```

```
procedure INVNUM (n,B,m,TAB ) ;
integer n,m ; integer array B, TAB ;
begin comment construct B which is generated as the (m+1)-st
sequence by procedure BALLOT. TAB is as described in procedure
NUM. The difference-sequence D[2],...,D[n], is computed in
B[1],...,B[n-1], which are then used to construct the ballot-
sequence. Procedure BSEARCH(TAB,i,k,j,unum) performs a binary
search among elements TAB[i,1],...,TAB[i,k] for the largest
entry not greater than m. It returns the value of this entry
in unum and its column index in j. min is a library function
whose value is the minimum of its arguments. ;
integer i,j,k,unum ;
i:=n ; k:=n-1 ;
for j:=1 to k do B[i]:=0 ;
loop : if m=0 then go to construct ;
              else BSEARCH(TAB,i,k,j,unum) ;
                   B[j]:=B[j +1];
                   k:=min(j,k-1) ;
                   m:=m-unum ;
                   i:=i-1 ;
                   go to loop ;
construct : B[n]:=0 ;
            for k:=n-1 step -1 to 1
                do B[k]:=B[k]+B[k+1] ;
end procedure INVNUM ;
```

Example

Given $B=\langle 5,3,1,1,0,0\rangle$ find $\text{num}_6(B)$. We first derive $D=\langle 1,2,2,0,1,0\rangle$. Hence , by procedure NUM

$$\text{num}_6(B)= \text{num}(U_{64})+\text{num}(U_{52})+\text{num}(U_{42})+\text{num}(U_{31})+\text{num}(U_{21})$$

$$= 48+5+4+1+1= 59.$$

Conversely , consider the construction of the ballot-sequence B of length 6 whose index is 86. In this case , five successive rows of the table (given below) containing the values of $\text{num}(U_{ik})$, are searched . The contents of array D after each search are listed below.

i) The maximal element not exceeding 86 in row 6 is 48 , found in column 4 , $D=\langle 0,0,0,0,1,0\rangle$.

ii) $m=86-48=38$, the required number in row 5 is $\text{num}(U_{54})=28$, $D=\langle 0,0,0,0,2,0\rangle$.

iii) $m=38-28=10$, $D=\langle 0,0,0,1,2,0\rangle$.

iv) $m=10-9=1$, $D=\langle 0,1,0,1,2,0\rangle$.

v) $m=1-1=0$ and no more searches are needed.

The required sequence B is constructed from D (in iv) as

$$B=\langle 4,3,3,2,0,0\rangle \quad .$$

Table of the $\text{num}(U_{ik})$ values

$i \backslash k$	1	2	3	4	5
1	-	-	-	-	-
2	1	-	-	-	-
3	1	3	-	-	-
4	1	4	9	-	-
5	1	5	14	28	-
6	1	6	20	48	90

4. Comparative evaluation

In this article , the approach taken for generating and indexing of all 'shapes' of binary trees , is based on a relation between such trees and ballot-sequences of the same order.

A different approach , based directly on stack-sortable permutations and Construction-T , has been adopted by Knott [2] to solve the same problems. It may be argued that the indexing we propose is not a natural one , while Knott uses the natural indexing [3 , pp: 331] for binary trees. However , the suggested algorithms are considerably more efficient than their counterparts in similar works known to the authors.

Let us first consider (a) Construction-T as opposed to (b) procedure TREE. The actual creation of the nodes of the tree is the same in both cases . In creating the 'shape' , it is well known that (a) requires $O(n^2)$ comparisons for worst case , and $O(n\log_2 n)$ for best case. The amount of assignment statements have similar bounds. In (b) , the extra memory space for n-1 pointers which may have to be stacked is insignificant. Comparisons are applied to the value j=D[i] , and since $\sum_{i=1}^{n} D[i] = n$, it is clear that exactly n comparisons are performed , while the number of assignment statements is only O(n). The superiority of (b) is highlighted especially for those applications where all C_n 'shapes' need to be generated.

If the 'shape' of the tree is given , generating its corresponding permutation Π or ballot-sequence B is straightforward and of equivalent complexity. However , if only the index of the tree is given , as in the case of random tree generation , the cost of procedure INVNUM is governed by the binary search which may require

$$\sum_{i=1}^{n} (\lfloor \log_2(i-1) \rfloor + 1) \approx O(n\log_2 n)$$

comparisons between entries of a two dimensional array TAB and the argument m of the procedure. In addition there is a fixed initial cost of $O(n^2)$ additions in computing all the values in TAB , but this is insignificant since in many applications it is required to generate random trees in large numbers.

Similar algorithms for generating Π from a given index rely on the definition of natural order among binary trees and the well known recursion relation

$$G_{n+1} = \sum_{j=0}^{n} G_j G_{n-j} \quad ,$$

where G_j is the number of distinct binary trees of order j. The relation between the complexities of the recursive procedure given by Knott [2] for this purpose, and INVNUM seems to be similar to the relation between Construction-T and procedure TREE.

Finally, let us consider the case where k trees having consecutive index numbers q,q+1,...,q+k-1, are to be generated. Using INVNUM we find the sequence $B=\text{num}_n^{-1}(q-1)$, and then call procedure BALLOT k times, with variable 'first' set to false. Note that we compute a ballot-sequence from a given index only once, and each consecutive sequence is generated from its predecessor at minimal cost. On the other hand, to generate k trees corresponding to consecutive permutations would require the transformation of k indices, since there is no simple way of deriving stack-sortable permutations in their order corresponding to the natural order of trees.

References

1. Knott, G.C. Deletion in binary storage trees. Ph.D. Dissertation, Computer Science Department, Stanford University 1975.

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

3. Knuth, D.E. The Art of Computer Programming, Vol.1, Fundamental Algorithms. Addison-Wesley, Reading, Mass., 1968.

4. Knuth, D.E. The Art of Computer Programming Vol.3, Sorting and Searching. Addison-Wesley, Reading, Mass., 1973.

5. Rotem, D. On a correspondence between binary trees and a certain type of permutation. Inf. Processing Letters 4(1975), 58-61.

6. Wells, M.B. Elements of Combinatorial Computing. Pergamon Press, New York, 1971.