SIAM J. COMPUT.
Vol. 9, No. 4, November 1980

CONSTANT TIME GENERATION OF ROOTED TREES*

TERRY BEYER† AND SANDRA MITCHELL HEDETNIEMI‡

Abstract. This paper generalizes a result of Ruskey [SIAM J. Comput., 7(1978), pp. 424–439] for generating k-ary trees lexicographically to generating all rooted trees with n vertices. An algorithm is presented which generates canonical representations of these trees in a well-defined order. As in other works, the average number of steps per tree is constant.

Key words. rooted tree generation, lexicographic order, algorithm

1. Introduction.

Interest has arisen recently in generating random trees, all binary trees, and all k-ary trees. The present work has centered on rooted trees, that is, those trees with a designated vertex as the root. The algorithm of Nijenhuis and Wilf [3] provides a method for generating random trees, but it does not provide for a systematic generation of all trees. Read [5] has formulated an algorithm to generate all trees on n vertices. This algorithm unfortunately must process trees on $n-1$ vertices.

The generation of all binary trees by Zaks [10] and Ruskey and Hu [7] uses "feasible" sequences which are altered so as to remain "feasible" and create the "next" tree in lexicographic order. Both Zaks [11] and Ruskey [6] have generalized this technique to k-ary trees. The feasible sequences which Zaks manipulates are related to the degrees of the vertices in the tree ordered by a preorder traversal of the tree. The sequences which Ruskey employs in his algorithm contain the level numbers of the endvertices in the tree; where an endvertex is a vertex of degree 1 and the level of a vertex is the number of vertices on the path from the root to an endvertex, counting both the root and the endvertex in the sum. The algorithm for generating k-ary trees with n leaves in lexicographic order is $O(k)$ per sequence generated.

Trojanowski [9] has presented another algorithm for generating all k-ary trees which manipulate tree permutations.

Zaks [10] has extended his result to the case of a general tree. His feasible sequences contain n entries for a tree of n vertices, and the work in generating the next sequence is $O(1)$.

In this paper we generalize the work of Ruskey to the generation of all rooted trees on n vertices. The feasible sequences used to generate the trees contain an entry of the level number for all vertices in the tree. The average number of computational steps per tree generated is shown to be uniformly bounded by a constant, independent of n.

2. Level representation of rooted, ordered trees.

A tree T is *rooted* if there exists a distinct vertex v designated the *root*. The remaining vertices are partitioned into disjoint sets $T_{u_1}, T_{u_2}, \cdots, T_{u_k}$ which are subtrees of v rooted at $u_1, u_2, \cdots, u_k$, respectively. A rooted tree is said to be *ordered* if there exists a linear order imposed upon the subtrees.

Define the *level* of a vertex v, l_v, to be 1 if v is the root; or one greater than the level of its parent, where v's *parent* is the adjacent vertex on the path from v to the root. A *level sequence* $L(T)=[l_1 l_2 \cdots l_n]$ for a given rooted, ordered tree with n vertices is obtained by traversing T in preorder (cf. Knuth [2]), and recording the level of each vertex as it is visited. T_i, the subtree of T rooted at vertex i, is described by a contiguous

* Received by the editors January 24, 1979, and in final revised form January 31, 1980.

† Computing Center and Department of Computer and Information Science, University of Oregon, Eugene, Oregon, 97403.

‡ Department of Computer and Information Science, Universtity of Oregon, Eugene, Oregon, 97403.

sequence of $L(T)$, denoted $L(T_i)$, which begins with l_i and ends just before the first element, if any, which is less than or equal to l_i (cf. Fig. 1).

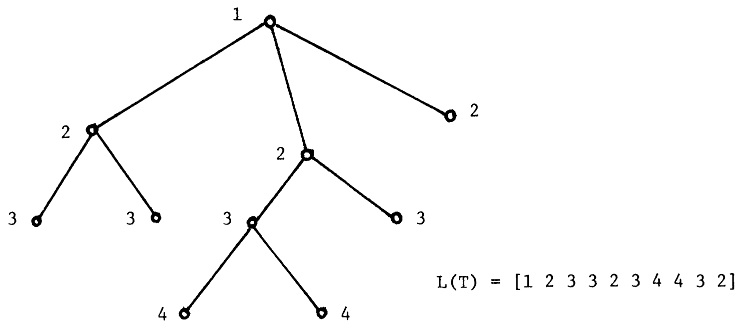

FIG. 1. *An ordered, rooted tree T, with vertex levels shown, and its level sequence.*

The level sequence provides a linear order of the subtrees (cf. Scoins [7]). Given two ordered trees T_1 and T_2, T_1 *dominates* T_2, denoted $T_1 > T_2$, if $L(T_1) > L(T_2)$ in the lexicographic ordering of integer sequences. It follows from the definition that any two trees having the same level sequence are isomorphic.

Two subtrees T_i and T_j are said to be *adjacent subtrees* of T if vertices i and j are consecutive children of the same parent. (Note: $L(T_i)$ and $L(T_j)$ are consecutive subsequences of $L(T)$.) If vertex i is a child of the root of T, T_i is a *principal subtree of T*.

3. Canonical representation of rooted trees. Given a rooted tree T, there exist many non-isomorphic ordered trees corresponding to T (and hence many level sequences). We use the following to ensure a canonical ordering of any rooted tree (i.e. a specific ordering of the subtrees).

The canonical ordering of T will be that ordered tree T^* which dominates all other ordered trees T' corresponding to T, i.e. $L(T^*) > L(T')$ for all T' corresponding to T. We denote the canonical level sequence by $L(T)^*$. Fig. 2 illustrates two trees T_1 and T_2

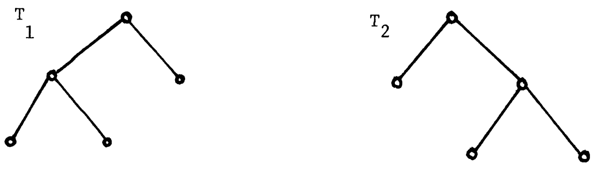

FIG. 2. *Two ordered trees with T_1 dominating T_2.*

which correspond to the same underlying rooted tree T. Since these are the only distinct trees corresponding to T, and since $L(T_1) > L(T_2)$, $L(T)^* = [1\ 2\ 3\ 3\ 2]$, and $T_1 = T^*$.

A level sequence $L(T)$ of a rooted, ordered tree T is *regular* if every pair of adjacent subtrees T_i and T_j, $i < j$, appear in decreasing order of dominance, i.e. $L(T_i) \geqq L(T_j)$.

LEMMA 1. *An ordered tree T is the canonical ordering of its underlying rooted tree T_R, if and only if, $L(T)$ is regular.*

Proof. Assume $L(T_R)^* = L(T)$ but $L(T)$ is not regular. Then there exist adjacent subtrees T_i and T_j such that $i < j$ and $L(T_i) < L(T_j)$. Let T' be the ordered tree obtained by exchanging T_i and T_j. Then $L(T') > L(T)$ which contradicts $L(T_R)^* = L(T)$.

By induction we show that any rooted tree has at most one corresponding regular sequence. For a tree with one vertex this follows immediately. Assume that all trees with less than n vertices have at most one regular sequence. Let $L(T)$ be a regular sequence for a rooted tree with n vertices. $L(T)$ consists of a 1 followed by subsequences corresponding to the principal subtrees of T. Let T_i be a principal subtree. Subtracting 1 from each element in $L(T_i)$ results in a level sequence $L'(T_i)$ for the rooted tree T_i. Since $L(T)$ is regular, $L'(T_i)$ is also regular. But by the inductive hypothesis, $L'(T_i)$ is the unique regular sequence for T_i. Furthermore, the relative order of the subsequences is unique up to isomorphism since $L(T)$ is regular. Therefore, $L(T)$ is the only regular sequence representing the rooted tree T.

LEMMA 2. *If $L(T)$ is regular, and two consecutive elements have a value* 2, *then all the remaining elements have a value* 2.

Proof. Let $l_i = l_{i+1} = 2$ in the regular sequence $L(T)$. Then l_i must represent a principal subtree of T consisting of a single vertex. Since the principal subtrees appear in lexicographic order, and since 2 is the least possible sequence for a subtree, all following subtrees must be represented by 2's.

4. Successor function.

One conceivable method for generating all rooted trees on n vertices would be to generate, in lexicographically decreasing order, the level sequences for all ordered trees on n vertices and filter out those which are not regular. We will now show, however, that it is possible to define a simple successor function which allows us to pass directly from one regular sequence to the next.

Let $L(T) = [l_1 l_2 \cdots l_n]$ be a level sequence containing an element greater than 2. Let p be the position of the rightmost such element. Let q be the rightmost position preceding p such that $l_q = l_p - 1$. Note that the vertex corresponding to l_q is the parent of the vertex corresponding to l_p. Define the successor of $L(T)$ to be the sequence $s(L(T)) = [s_1, s_2 \cdots s_n]$, where

(i) $s_i = l_i$ for $i = 1, 2, \cdots, p-1$,

(ii) $s_i = s_{i-(p-q)}$ for $i = p, \cdots, n$.

The relationships between $L(T)$ and $s(L(T))$ can be seen from the following lines:

$$L(T) = [l_1 \cdots l_q \cdots l_{p-1}\ l_p\ 2\ 2\ 2 \cdots]$$

$$s(L(T)) = [l_1 \cdots \underbrace{l_q \cdots l_{p-1}}_{T_q}\ \underbrace{l_q \cdots l_{p-1}}_{T_q}\ \underbrace{l_q \cdots}_{T_q}].$$

The subsequence labeled T_q represents a subtree of $s(L(T))$ which is repeated as many times as possible, concluding with a partial repetition if necessary to reach a total of n vertices.

Here are some examples of canonical representations and their successors.

$L(T)$	$s(L(T))$
[1 2 3 2 2 2]	[1 2 2 2 2 2]
[1 2 3 4 2 2]	[1 2 3 3 3 3]
[1 2 3 4 3 2 2 2 2 2 2]	[1 2 3 4 2 3 4 2 3 4 2 3]
[1 2 3 4 5 5 2 2 2 2]	[1 2 3 4 5 4 5 4 5 4]

LEMMA 3. *If $L(T)$ is regular and contains an element greater than* 2, *then $s(L(T))$ is also regular.*

Proof. Let $i<j$ be any two indices corresponding to adjacent subtrees T_i and T_j in $s(L(T))$. Let p denote the index of the rightmost value which is not 2, and let q denote the index of the corresponding vertex's parent. If $q \leqq i$, then T_i and T_j are adjacent subtrees in the sequence $T_q, T_q, T_q, \cdots$, from which it follows that $T_i \geqq T_j$. If $i<q$ and $L(T_j)$ does not overlap position p, then T_i and T_j are also adjacent subtrees in the regular sequence $L(T)$, and hence $T_i \geqq T_j$. In the remaining case, $i<q$ and T_j does overlap position p. In this case, the subsequences representing T_i and T_j are the same in $L(T)$ as in $s(L(T))$ up to position p. In that position the value for vertex p is one less in $s(L(T))$ than it was in $L(T)$. Hence we still have $T_i \geqq T_j$.

LEMMA 4. *Let $L(T)$ be a regular sequence of length $n \geqq 2$. If $L(T)$ is of the form $[1\ 2\ 2 \cdots 2]$, then it is the lexicographically least sequence of length n. Otherwise, $s(L(T))$ is defined and is the first regular sequence following $L(T)$ in the lexicographic ordering.*

Proof. The fact that $[1\ 2\ 2 \cdots 2]$ is the least sequence is immediate. By Lemma 3, $s(L(T))$ is regular. It remains to show that no regular sequence is lexicographically between $L(T)$ and $s(L(T))$. Let $L(T)=[l_1 \cdots l_n]$, $s(L(T))=[s_1 \cdots s_n]$, $L(T')=[m_1 \cdots m_n]$, and assume that $L(T)>L(T')>s(L(T))$. Let p, q, and T_q be as previously defined. Since $l_i=s_i$ for all $1 \leqq i \leqq p$, all three sequences are identical in the first $p-1$ positions. If $l_p=m_p$, then $L(T')=L(T)$, since $l_i=2$ for $i \geqq p+1$, and since the only 1 in $L(T')$ appears in position 1. But since $L(T') \neq L(T)$, $l_p \neq m_p$. But $s_p=l_p-1$. Hence, since $m_i=s_i$ for all $i<p$ and $L(T)>s(L(T))$, we have $m_p=s_p$. Thus the first position, r, in which $L(T')$ and $s(L(T))$ differ, is such that $r>p$. But with respect to $s(L(T))$, this must occur in one of the copies of T_q with another copy of T_q to its left. Thus in the sequence $[m_1 \cdots m_n]$ there are two adjacent subtrees not in lexicographic order. That is, $L(T')$ is not regular.

5. Generating algorithm.

5. Generating algorithm. An algorithm to generate all rooted trees on N vertices begins with the lexicographically greatest possible regular sequence, then passes from one regular sequence to the next using the successor function until finally the lexicographically least sequence has been generated.

We present in Fig. 3 a program to implement this algorithm written in FLECS [1], a structured extension of Fortran. Variables L, N, and P correspond to the level sequence, the number of vertices, and the position p in § 4, respectively. The array PREV is used to quickly find the values of q as given in § 4. If value i appears in array L to the left of position P, then PREV(i) is the index of the rightmost such appearance; otherwise, PREV(i) = 0. SAVE is used to efficiently update PREV when adjusting P.

6. Analysis of complexity. The average computational effort per tree generated is bounded by a constant which is independent of N. That is, there is a constant K such that given $N \geqq 1$, the effort expended in generating all $A(N)$ rooted trees on N vertices is less than $K * A(N)$.

For $N \geqq 4$, the number of rooted trees is greater than N. Hence the cost of generating the first tree, which is linear in N, can be ignored. It remains to analyze the total cost of generating the next tree over all $A(N)$ invocations.

Since initially $P=N$ and finally $P=1$, P undergoes a total change in value of $N-1$. But P is increased once for each execution of Line 31, and is decreased once for each execution of Line 36, and is not altered elsewhere. Hence, it remains to show that the total number of decrements is $\leqq K * A(N)$ for some constant K.

On entry to GENERATE-NEXT-TREE, L contains a regular sequence and P indicates its rightmost non-2 element. By Lemma 2, there cannot be two consecutive 2's to the left of position P. Thus, if the condition in Line 26 is false, execution passes

```
 1   SUBROUTINE GENRT (N, L, PREV, SAVE)
 2
 3   INTEGER N, L(N), PREV(N), SAVE(N)
 4   INTEGER P, DIFF, I
 5
 6   GENERATE-FIRST-TREE
 7   CALL PROCESS(N,L)
 8   WHILE (P .GT. 1)
 9   .  GENERATE-NEXT-TREE
10   .  CALL PROCESS(N,L)
11   ...FIN
12   RETURN
13
14   TO GENERATE-FIRST-TREE
15   .  DO (I+1,N) L(I) = I
16   .  WHEN (N .LE. 2) P = I
17   .  ELSE P = N
18   .  IF (P .GT. 1)
19   .  .  DO (I=1,P-1)  PREV(I) = I
20   .  .  DO (I=1,P-1)  SAVE(I) = 0
21   .  ...FIN
22   ...FIN
23
24   TO GENERATE-NEXT-TREE
25   .  L(P) = L(P) - 1
26   .  IF (P .LT. N  .AND.  (L(P) .NE. 2  .OR.  L(P-1) .NE. 2) )
27   .  .  DIFF = P - PREV( L(P) )
28   .  .  WHILE (P .LT. N)
29   .  .  .  SAVE(P) = PREV( L(P) )
30   .  .  .  PREV( L(P) ) = P
31   .  .  .  P = P + 1
32   .  .  .  L(P) = L(P-DIFF)
33   .  .  ...FIN
34   .  ...FIN
35   .  WHILE (L(P) .EQ. 2)
36   .  .  P = P - 1
37   .  .  PREV( L(P) ) = SAVE (P)
38   .  ...FIN
39   ...FIN
40
41   END
```

FIG. 3. *A program to generate rooted trees.*

directly to Line 35 and no more than two iterations of the loop occur. If the condition in Line 26 is true, then repeatedly a copy is made to the right of the subsequence from the left of the subsequence of a subsequence which does not have two consecutive 2's in it. In this case, the loop in Lines 35–38 will be executed at most once. In either case, the loop is executed at most two times per invocation. Thus, the number of decrements of P is $\leqq K * A(N)$.

Although the proof shows that on the average no more than two times is the loop executed per tree generated, a more complicated proof could be given which lowers this bound. Empirically, we have found that the average number of executions decreases as N increases. For $N=7$ the loops are executed .6 times per tree generated, and by $N=11$ this number is down to .5.

A variant of Subroutine GENRT has been executed on a DEC-System-10 (KA-processor), and found to generate roughly $\frac{1}{2}$ million trees per minute.

7. Computation of $A(N)$. It is significant to enquire exactly how many rooted trees exist and therefore will be generated upon execution of Subroutine GENRT for a given input value of N; i.e. what is $A(N)$?

In Knuth [2], the following recursive function of Otter [4] is presented for calculating this number:

$$N * A(N+1) = A(1) * S(N,1) + 2 * A(2) * S(N,2) + \cdots + N * A(N) * S(N,N),$$

where:

$$S(N, K) = \sum_{1 \leqq j \leqq N/K} [A(N+1-J+K)].$$

The $S(N, K)$ terms can be calculated very simply using iterations where the step size is dependent upon J and K. Table 1 presents the values $A(N)$ for the first 10 values of N. It also displays the number of trees of various height that are generated; Subroutine GENRT, in fact, generates the trees according to height.

Although it is common in the literature (cf. [6]–[11]) to provide ranking and unranking algorithms based on the lexicographic order, Table 1 illustrates that this might be quite difficult. Unranking algorithms are often used to generate a random tree. Since the process of generating trees is quite efficient, a suitable alternative would be to simply stop the generative process at a randomly determined time.

TABLE 1.
A catalog of rooted unordered trees having fewer than ten vertices.

Number of Vertices	$A(N)$	Number of trees of DEPTH									
		1	2	3	4	5	6	7	8	9	10
1	1	1									
2	1	0	1								
3	2	0	1	1							
4	4	0	1	2	1						
5	9	0	1	4	3	1					
6	20	0	1	6	8	4	1				
7	48	0	1	10	18	13	5	1			
8	115	0	1	14	38	36	19	6	1		
9	286	0	1	21	46	113	61	26	7	1	
10	719	0	1	29	147	225	180	94	34	8	1

7. Concluding remarks. The basic level sequence generation algorithm presented in this paper has also been adapted to the generation of unrooted (free) trees. The resulting algorithm also appears to have the constant time per tree property. An implementation of this algorithm generates roughly $\frac{1}{4}$ million trees per minute on the DEC-System-10 (KA-processor).

Read [5] has proposed a very general mechanism for generating classes of combinatorial objects. In the case of rooted trees, the method presented in this paper appears to be superior because of both its actual speed and its constant time per tree property. Read's method would not appear to have the constant time per tree property, since it involves processing representations on $n-1$ vertices in order to get representations on n vertices.

Acknowledgments. We wish to thank Professors Steve Hedetniemi and Andrzej Proskurowski for encouragement and stimulation.

REFERENCES

[1] T. BEYER, FLECS: *User Manual*, Computing Center, University of Oregon, Eugene OR, 1975.
[2] D. KNUTH, *Fundamental Algorithms*, Addison-Wesley, Reading MA, 1968.
[3] A. NIJENHUIS AND H. WILF, *Combinatorial Algorithms*, Academic Press, New York, 1975.
[4] R. OTTER, *The number of trees*, Ann. Math., 49 (1948), pp. 583–599.

[5] R. READ, *How to grow trees*, in Combinatorial Structures and their Applications, Gordon and Breach, New York, 1970.

[6] F. RUSKEY, *Generating t-ary trees lexicographically*, this Journal, 7 (1978), pp. 424–439.

[7] F. RUSKEY AND T. C. HU, *Generating binary trees lexicographically*, this Journal, 6 (1977), pp. 745–758.

[8] H. SCOINS, *Placing trees in lexicographic order*, Machine Intelligence, 3 (1969), pp. 43–60.

[9] A. TROJANOWSKI, *Ranking and listing algorithms for k-ary trees*, this Journal, 7 (1978), pp. 492–509.

[10] S. ZAKS, *Lexicographic generation of ordered trees*, Theoret. Comput. Sci., 10 (1980), pp. 63–82.

[11] ———, *Generating k-ary Trees Lexicographically*, University of Illinois Tech. Rept. UICSCS-R 77-901, Urbana IL, 1977.

[12] S. ZAKS AND D. RICHARDS, *Generating trees and other combinatorial objects lexicographically*, this Journal, 8 (1979), pp. 73–81.