SIAM J. COMPUT.
Vol. 15, No. 2, May 1986

017

CONSTANT TIME GENERATION OF FREE TREES*

ROBERT ALAN WRIGHT†, BRUCE RICHMOND‡, ANDREW ODLYZKO§
AND BRENDAN D. MCKAY¶

Abstract. An algorithm of Beyer and Hedetniemi [SIAM J. Comput., 9 (1980), pp. 706–712] for generating rooted unlabeled trees is extended to generate unlabeled free trees. All the nonisomorphic trees of a given size are generated, without repetition, in time proportional to the number of trees.

Key words. free tree, unrooted tree, nonisomorphic trees, constant time generation, constructive enumeration, lexicographic order, loop-free algorithm

1. Introduction. In [1], Beyer and Hedetniemi exhibit an algorithm for generating all rooted trees of a given size. The method uses a successor function to traverse an ordered set of integer sequences which represents the objects being generated. This method is based on one introduced by Ruskey and Hu [7]. In this paper the technique of Beyer and Hedetniemi is refined to produce only one member of each equivalence class of rooted trees under isomorphism of the underlying free (unrooted) trees.

Previous algorithms for generating these trees have been given by Read [6], Dinits and Zaitsev [2] and Kozina [3]. Our algorithm has an advantage over each of these, in that it only requires $O(n)$ space and constant average time per tree (independently of n). An algorithm for the corresponding random generation problem has been given by Wilf [8].

2. Representing trees by level sequences. The notation (T, z) is used here to denote the rooted tree with underlying free tree T and root vertex z. The *level* of a vertex v in a rooted tree (T, z) is one more than the distance from the vertex to the root. The root z is assigned level value 1. A *level sequence* is defined as a sequence of integers produced by listing the level of each vertex of a rooted tree in preorder. Since a preorder traversal may visit the subtrees at a given vertex in various orders, level sequences for a rooted tree are, in general, not unique. The notation $L(T, z) = [l_1, l_2, \cdots, l_n]$ will be used for any level sequence of a rooted tree (T, z) on n vertices.

In order to have a unique level sequence representation for a given rooted tree, the rules for the preorder traversal must be refined slightly. For a level sequence to be *canonical*, the traversal must visit the roots of adjacent subtrees in nonincreasing lexicographic order of the canonical level sequences of those subtrees. A simpler (and equivalent) formulation of this canonicity criterion is the requirement that the canonical sequence for a given rooted tree be the sequence which is the lexicographically greatest of all level sequences describing that same tree. A more complete discussion of canonicity (with proofs and examples) may be found in [1]. The canonical level sequence of a rooted tree (T, z) will be denoted by $L^*(T, z)$.

3. Generating all canonical level sequences of given length. Beyer and Hedetniemi have described in [1] an algorithm which generates all canonical level sequences of

* Received by the editors March 4, 1983, and in revised form February 6, 1985.

† 1948d Adams Avenue, Costa Mesa, California 92626.

‡ Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

§ AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

¶ Computer Science Department, Australian National University, Canberra ACT 2601, Australia.

given length in lexicographic order. Their method involves an iterative algorithm which has as its basis a *successor function*, which when given any canonical level sequence, will generate the next canonical sequence, with respect to lexicographic order. The precise definition is this: Let $L = L^*(T, z)$ be a canonical level sequence of length n. Let p be the largest integer such that $l_p \neq 2$ and let q be the largest integer such that $q < p$ and $l_q = l_p - 1$. The successor $s(L) = [s_1, s_2, \cdots, s_n]$ of L then is given by:

$$s_i = \begin{cases} l_i, & \text{for } 1 \leqq i < p, \\ s_{i-(p-q)}, & \text{for } p \leqq i \leqq n. \end{cases}$$

In [1] it is proved that this function transforms any canonical level sequence other than $[1, 2, 2, \cdots, 2]$ into the next canonical level sequence, in decreasing lexicographic order.

We now will extend these results by deriving an algorithm which generates a subset of the canonical level sequences corresponding to the set of free trees on a given number of vertices.

4. Extracting a nonisomorphic subset of rooted trees by root selection. The objective here is to place requirements on the root vertex, which may only be satisfied by one rooted tree with a given underlying tree. First we will require that the root be an element of the *center* of the tree, which is defined to be the set of vertices whose maximum distance from the other vertices is least. Since all trees have either one vertex or two adjacent vertices in the center, we now need only refine this rule for the bicentral case. Let T be a bicentral tree, and consider the subtrees T_1 and T_2 remaining when the edge joining the two candidate roots, z_1 and z_2, is deleted. Two rooted trees, (T_1, z_1) and (T_2, z_2) are formed in this way. Either these two trees are isomorphic, in which case the rooted trees (T, z_1) and (T, z_2) are isomorphic, or they can be distinguished by their size, or by the precedence of their canonical level sequences (when they are the same size). The root selected for T will be z_1 if T_1 has fewer vertices than T_2 or if they have the same order and $L^*(T_1, z_1)$ is lexicographically less than $L^*(T_2, z_2)$. Otherwise we select z_2.

We will call the root uniquely selected by the above criteria the *primary root* of the tree T, and denote it by $\bar{z}(T)$, or just $\bar{z}$. Then for any given tree T of size n, there is exactly one member of the set of all rooted trees on n vertices with root meeting the above criteria, namely $(T, \bar{z})$. A level sequence $L(T, \bar{z})$ will be called a *primary level sequence* of T, and $L^*(T, \bar{z})$ will be called the *primary canonical level sequence* of T.

With the above criteria, a maximal set of nonisomorphic free trees may be extracted from the set of all rooted trees of a given size by choosing only trees whose roots are primary. But we need to apply these criteria not to the trees per se, but to their canonical level sequences.

5. Refining the canonicity criteria to obtain only primary level sequences. We will now translate the previously established rules for primary root selection to conditions sufficient for a canonical level sequence to be primary. Let (T, z) be the tree in question, and let $L = L^*(T, z)$. The first requirement was that z be in the center of the tree. We will need to use the fact that z is in the center of T if and only if it is in the center of every path of maximum length in T. The position of z in such a path can be readily checked for a canonical level sequence. To do this, consider the structure of L when viewed as the level number of z (namely 1) concatenated with the level sequences of each of the components remaining when z is deleted from T. These component level sequences, which we will call the *principal subsequences* of L, begin with level 2, instead

of 1, but otherwise are canonical level sequences themselves, due to the recursive definition of canonicity. These subsequences are, moreover, ordered in L by height, so the position of z within one path of maximum length can be determined by looking at the height of the first two principal subsequences, which we will call S_1 and S_2. This is given precisely by the highest level numbers in those respective subsequences. We will identify the position of the first occurrence of the highest level number in the ith principal subsequence by the subscript h_i. The height of S_1 then is $l_{h_1}-2$, and the height of S_2 (if it exists) is $l_{h_2}-2$. We then conclude that the maximum length of a path in T is $l_{h_1}+l_{h_2}-2$, and z is in the center of T if and only if h_2 exists and $l_{h_1}-l_{h_2}$ is either 0 or 1. In addition, if $l_{h_1}-l_{h_2}=0$, then T is unicentral, and hence L is known to be primary without further checking.

In the bicentral case, we must be able to compare $L_1=L(T_1, z_1)$ and $L_2=L(T_2, z_2)$. The two central vertices are those which are represented by level numbers l_1 and l_2, so if we let $z_2=z$, and z_1 be the vertex corresponding to l_2, then L_1 is the same as S_1 (though level numbers will be greater by 1), and L_2 is the same as L with L_1 removed. Size comparison of L_1 and L_2 can be accomplished trivially if we know the position of the start of the second principal subsequence: if the level number at this position is denoted l_m, then the size of L_1 is $m-2$, and that of L_2 is $n-m+2$ (where n is the length of L). Finally, by checking lexicographic precedence of $L_1=[l_2-1, l_3-1, \cdots, l_{m-1}-1]$ and $L_2=[l_1, l_m, l_{m+1}, \cdots, l_n]$, we have that L is primary for T if and only if L_1 is identical to L_2, or L_1 has lower precedence than L_2. This can be summarized as follows:

Let (T, z) be a rooted tree with n vertices and a canonical level sequence $L=[l_1, l_2, \cdots, l_n]$. Then using the definitions of h_1, h_2, m, L_1 and L_2 given above, we have that L is the primary canonical level sequence of T if and only if all of the following hold for L:

(i) h_2 (and hence m) exists,

(ii) $l_{h_2} \geqq l_{h_1}-1$,

(iii) if equality holds in (ii), then $m-2 \leqq n-m+2$,

(iv) if equality holds in (iii), then either $L_1=L_2$, or L_1 is shorter than L_2, or L_1 has the same length as L_2 but precedes L_2 lexicographically.

Now we are ready to derive an algorithm for generating all such primary canonical level sequences for a given value of n.

6. Generating all primary canonical level sequence of a given size. We wish to derive a successor function which, if given an appropriate starting sequence, will efficiently generate all primary canonical level sequences of the same size. It is not obvious at first that such a successor function follows naturally from the one defined by Beyer and Hedetniemi. However, due to certain choices made in the definition of the primary canonical level sequences, the previously defined function will usually yield a primary canonical level sequence when it transforms a sequence which is primary and canonical. Thus our goal becomes to detect the cases where the s function will fail and to take alternate action for those cases. As it turns out, the alternate action is trivial, although it has a significant effect on the length of the algorithm. To determine the "failure cases" for s, we will examine what the input sequence must look like in order to cause s to produce a sequence violating one of the conditions for primary sequences.

The first case we will look at is the case where s transforms a primary canonical sequence L into a sequence $s(L)$ for which the condition (ii) is violated. (Note that condition (i) can never be violated in this way). In this case, the value of l_{h_2} is changed from $l_{h_1}-1$ to $l_{h_1}-2$, which means that the p value for L is h_2, and $l_i=2$ for $p<i \leqq n$. Hence this type of failure will occur if and only if $l_{h_2}=l_{h_1}-1$ and $p=h_2$, with the one

exception being $L=[1, 2, 3, 2, 2, \cdots, 2]$, which is not a problem since in this case $s(L)$ is the last sequence to be generated.

The second case where s will fail is when $s(L)$ violates rule (iii) but not rule (ii). This occurs when L_1 is larger than L_2 in the converted sequence. For this to happen, l_{h_1} must be equal to l_{h_2}, and p must be equal to h_2 (since l_{h_2} must change). In addition, we must have $m-2>n-m+2$ for L_1 to end up larger than L_2. It should also be clear that these conditions are sufficient for a failure to occur as well.

The final kind of failure with which we must deal occurs when condition (iv) alone is violated. But this happens precisely when $L_1=L_2$, since the precedence of L_2 is always diminished when L_1 remains unchanged (and when L_1 does change, this failure cannot occur). Hence this failure condition can be detected by comparison of L_1 and L_2.

We are left with the question of what to do when we encounter one of these failure cases. Once again, the definition of the primary canonical sequences has been so chosen as to make this easy. Very simply, to get the next primary canonical sequence from a sequence L which satisfies one of the failure conditions described above, we first set p to $m-1$, apply s to get $s(L)$, and if $l_{m-1}>3$ (with the old value of m), we replace the final h_1-1 elements of $s(L)$ with $2, 3, \cdots, h_1$. The reason for this is straightforward: l_{m-1} must change (i.e. L_1 must change), since changes beyond $m-1$ will only result in L_2 having lower precedence, which can never result in all of conditions (i)-(iv) being satisfied again. Now, when L_1 does change, there are two cases to consider. Either l_{m-1} was a 3, in which case the action of s on L will result in copies of L_1 being made starting at l_m and repeating through l_n, or $l_{m-1}>3$, in which case no L_2 sequence will occur in $s(L)$. In the former case, $s(L)$ is primary, and so we are done, but in the latter case, we must correct the fact that L_2 has been eliminated. By replacing the last h_1-1 level numbers with $2, 3, \cdots, h_1$, we replace the fewest number of level numbers that we can (L_2 must have the same height as L_1), and the replacement is the highest sequence of its length which retains canonicity. Thus we necessarily have the primary canonical sequence of highest precedence which has lower precedence than the value of L which we started with.

7. Generating trees in lexicographic order. In order to be able to check the failure conditions described in the previous section, and to do so without increasing the complexity of the succession algorithm, the values of h_1, h_2, p, etc., are to be maintained. In addition, an index c to the first element of L_2 which is not the same as the corresponding element of L_1 must be kept, so that it will be apparent when $L_1=L_2$ (to facilitate detection of the third kind of failure condition described above). Lastly, a sequence $W=[w_1, w_2, \cdots, w_n]$ will be kept such that w_i is the subscript of the level number in L corresponding to the parent of the vertex corresponding to l_i in the tree represented by L.

The procedure below will accept any primary canonical level sequence other than $[1, 2, 2, \cdots, 2]$ and produce the next primary canonical level sequence in canonical order. The parameters are as we have defined except for r, which is one less than m. The value of c is occasionally set to ∞ when it will not be needed at the next iteration.

The first primary canonical level sequence is that of a path rooted at its center. To find its parameters (for $n\geqq 4$), let $k=\lfloor n/2\rfloor+1$. Then $L=[1, 2, \cdots, k, 2, 3, \cdots, n-k+1]$, $W=[0, 1, \cdots, k-1, 1, k+1, \cdots, n-1]$, $p=n$ (except that $p=3$ when $n=4$), $q=n-1$, $h_1=k$, $h_2=n$ and $r=k$. Correct operation is assured if c is initialised to ∞ for odd n and $n+1$ for even n. The last tree has been generated when the procedure returns with $q=0$.

```
procedure nexttree (L, W, n, p, q, h1, h2, c, r)
  fixit ← false
  if c = n+1 or p = h2 and (l_{h1} = l_{h2}+1 and n − h2 > r − h1 or
                l_{h1} = l_{h2} and n − h2 + 1 < r − h1) then
    if l_r > 3 then
      p ← r; q ← w_r
      if h1 = r then h1 ← h1 − 1 endif
      fixit ← true
    else
      p ← r; r ← r − 1; q ← 2
    endif
  endif
  needr ← false; needc ← false; needh2 ← false
  if p ≦ h1 then h1 ← p − 1 endif
  if p ≦ r then needr ← true
  elseif p ≦ h2 then needh2 ← true
  elseif l_{h2} = l_{h1} − 1 and n − h2 = r − h1 then
      if p ≦ c then needc ← true endif
  else c ← ∞
  endif
  oldp ← p; δ ← q − p; oldlq ← l_q; oldwq ← w_q; p ← ∞
  for i ← oldp to n do
    l_i ← l_{i+δ}
    if l_i = 2 then w_i ← 1
    else
      p ← i
      if l_i = oldlq then q ← oldwq
      else q ← w_{i+δ} − δ
      endif
      w_i ← q
    endif
    if needr and l_i = 2 then
      needr ← false; needh2 ← true; r ← i − 1
    endif
    if needh2 and l_i ≦ l_{i−1} and i > r+1 then
      needh2 ← false; h2 ← i − 1
      if l_{h2} = l_{h1} − 1 and n − h2 = r − h1 then needc ← true
      else c ← ∞
      endif
    endif
    if needc then
      if l_i ≠ l_{h1−h2+i} − 1 then needc ← false; c ← i
      else c ← i + 1
      endif
    endif
  endfor
  if fixit then
    r ← n − h1 + 1
    for i ← r+1 to n do
      l_i ← i − r + 1; w_i ← i − 1
    endfor
```

```
    w_{r+1} ← 1; h_2 ← n; p ← n; q ← p − 1; c ← ∞
  else
    if p = ∞ then
      if l_{oldp−1} ≠ 2 then p ← oldp − 1
      else p ← oldp − 2
      endif
      q ← w_p
    endif
    if needh2 then
      h_2 ← n
      if l_{h_2} = l_{h_1} − 1 and h_1 = r then c ← n + 1
      else c ← ∞
      endif
    endif
  endif
endprocedure.
```

8. Proof of the constant time property. Let t_n and T_n denote the number of free and rooted unlabeled trees of order n, respectively. A fundamental tool in our analysis is the following result of Pólya [5] and Otter [4].

THEOREM 1. *$T_n \sim C_1 n^{-3/2}\rho^{-n}$ and $t_n \sim C_2 n^{-5/2}\rho^{-n}$ as $n \to \infty$, where $C_1 \approx 0.4399$, $C_2 \approx 0.5349$ and $\rho \approx 0.3383$.*

To begin, we must establish that the total number of steps taken to do all simple successions does not exceed $O(t_n)$. (That is, we are not yet counting the steps used in recovering from any of the three failure conditions which can be encountered.) The number of steps in each of these conversions is of the order of $n-p+1$, i.e. the number of elements of the sequence which change. Hence we must sum all of the $n-p+1$ values for arbitrary n. Since the number $n-p$ is just the number of leaves adjacent to the root, if we denote the number of trees of size n with exactly k root-adjacent leaves by $t_{n,k}$, the desired sum is then $\sum_{k=0}^{n-1}(k+1)t_{n,k}$. Observing that $t_{n,k} \leqq t_{n-k}$, we then have that the above sum is no more than $\sum_{k=0}^{n-1}(k+1)t_{n-k} \leqq 4t_n$.

We now need only count the steps used in correcting failure conditions. To do this, we note that the number of steps needed to correct a single instance of a failure condition is no more than $O(n)$, and show that the number of failure cases becomes so small compared to t_n for large n that they are not significant. We will handle the three kinds of failure conditions separately.

The third kind of failure can be dispensed with easily, since it occurs only when $L_1 = L_2$, which means that the tree is symmetric about a central edge, and thus there are at most $T_{n/2}$ instances of this failure for given n. By Theorem 1, $T_{n/2}/t_n \to 0$ exponentially fast, which finishes this case.

For the first kind of failure condition, the starting sequence must have the form

$$L = [1, 2, 3, \cdots, l_{h_1}, \cdots, l_m = 2, 3, \cdots, l_{h_2} = l_{h_1} - 1, 2, 2, \cdots, 2].$$

In addition, $h_1 < m < a$, where $a = \lfloor (n+1)/2 \rfloor$, and so clearly the sum of all such cases for given n does not exceed T_a, which behaves the same as the sum did for the third kind of failure condition.

The second kind of failure condition occurs with greatest frequency. For this kind of failure to occur, the starting sequence must have the form

$$L = [1, 2, \cdots, l_{h_1}, \cdots, l_m = 2, 3, \cdots, l_{h_2} = l_{h_1}, 2, 2, \cdots, 2].$$

Also, $3 \leqq h_1 \leqq n - m - 2$ for $n/2 \leqq m \leqq n-3$ so if we denote the number of rooted trees

of height h and size n by T_n^h, and the number of rooted trees of height no more than h and size n by S_n^h, the number of sequences of the above form is no more than

$$\sum_{m=n/2}^{n-3} \sum_{h_1=2}^{n-m-1} T_m^{h_1-1} < \sum_{k=1}^{n/2} S_{n-k}^k.$$

The latter sum can be divided into two sums

$$\sum_{k=1}^{n/2} S_{n-k}^k = \sum_{k=1}^{\lfloor 3 \log n \rfloor} S_{n-k}^k + \sum_{k=\lfloor 3 \log n \rfloor + 1}^{n/2} S_{n-k}^k.$$

The second sum is much less than $nT_{n-\lfloor 3 \log n \rfloor}$, which is $o(n^{-1}t_n)$, by Theorem 1. To handle the remaining sum, we just need the following.

THEOREM 2. *There is a constant $\delta > 0$ such that uniformly for $1 \leqq h \leqq n$ we have*

$$S_h^n = O(T_n n^{3/2} \exp(-\delta n/h^2)).$$

The proof of Theorem 2 will follow from several auxiliary results. We first prove the known fact that $T(\rho) = 1$. We define

$$J(x, y) = x \exp\left(y + \sum_{k=2}^{\infty} k^{-1} T(x^k)\right) - y.$$

The functional equation satisfied by $T(x)$ states that $J(x, T(x)) = O$. Since $T(x)$ has a singularity at $x = \rho$, and $J(x, y)$ is analytic in both x and y separately in a neighborhood of $(\rho, T(\rho))$, it follows that we must have

$$\left.\frac{\partial J(x, y)}{\partial y}\right|_{\substack{x=p \\ y=T(\rho)}} = 0,$$

which says that

$$1 = \rho \exp\left(T(\rho) + \sum_{k=2}^{\infty} k^{-1} T(\rho^k)\right) = T(\rho).$$

LEMMA 1. *Define $x_h > 0$ by $S_h(x_h) = 1$. Then there exists a constant C such that for all $h \geqq 1$,*

$$S_h'(x_h) \leqq Ch.$$

Proof. Since $\frac{1}{2} = x_1 > x_2 > \cdots$, we see that $x_h \leqq 3\rho/2$ for all $h \geqq 1$. Let

$$C = \max\left(4, \rho^{-1} + \sum_{k=2}^{\infty} (3\rho/2)^{k-1} T'((3\rho/2)^k)\right).$$

Since $3\rho/2 < 0.55$, the series above converges. Since $S_1'(x_1) = 4$, the lemma clearly holds for $h = 1$. Suppose that the lemma holds for h. Then

$$S_{h+1}'(x) = \left(1 + x \sum_{k=1}^{\infty} S_h'(x^k) x^{k-1}\right) \exp\left(\sum_{k=1}^{\infty} k^{-1} S_h(x^k)\right)$$

$$= \left(\frac{1}{x} + \sum_{k=1}^{\infty} S_h'(x^k) x^{k-1}\right) S_{h+1}(x).$$

Hence, using $S_{h+1}(x_{h+1}) = T(\rho) = 1$, we find

$$S_{h+1}'(x_{h+1}) \leqq \rho^{-1} + \sum_{k=2}^{\infty} (3\rho/2)^{k-1} T'((3\rho/2)^k) + S_h'(x_{h+1}) \leqq C + Ch = C(h+1),$$

which proves the lemma by induction.

LEMMA 2. *There exists a constant K such that for all $h \geqq 1$,*

$$1 - S_h(\rho) \geqq K/h.$$

Proof. Clearly the lemma is true for $1 \leqq h \leqq 2$ and some $K < \frac{1}{2}$. Suppose the lemma holds for $h \leqq H-1$, $H \geqq 3$. Now

$$\begin{aligned} S_H(\rho) &= \rho \exp\left(S_{H-1}(\rho) + \sum_{k=2}^{\infty} k^{-1} S_{H-1}(\rho^k) \right) \\ &= \rho \exp\left(T(\rho) + \sum_{k=2}^{\infty} k^{-1} T(\rho^k) + S_{H-1}(\rho) - T(\rho) + \sum_{k=2}^{\infty} k^{-1}(S_{H-1}(\rho^k) - T(\rho^k)) \right) \\ &= T(\rho) \exp\left(S_{H-1}(\rho) - T(\rho) + \sum_{k=2}^{\infty} k^{-1}(S_{H-1}(\rho^k) - T(\rho^k)) \right) \\ &= \exp\left(S_{H-1}(\rho) - 1 + \sum_{k=2}^{\infty} k^{-1}(S_{H-1}(\rho^k) - T(\rho^k)) \right). \end{aligned}$$

Now, since $S_{H-1}(\rho^k) \leqq T(\rho^k)$, the induction hypothesis yields

$$\begin{aligned} S_H(\rho) &\leqq \exp\left(-\frac{K}{H-1} \right) \leqq 1 - \frac{K}{H-1} + \frac{K^2}{2(H-1)^2} \\ &= 1 - \frac{K}{H} + K\left(\frac{1}{H} - \frac{1}{H-1} \right) + \frac{K^2}{2(H-1)^2} \\ &= 1 - \frac{K}{H} + K\left(\frac{K}{2(H-1)^2} - \frac{1}{H(H-1)} \right) < 1 - \frac{K}{H}. \end{aligned}$$

Thus the lemma follows by induction for all h.

We can now prove Theorem 2. From Lemma 2,

$$S_h'(x_h)(x_h - \rho) \geqq S_h(x_h) - S_h(\rho) = 1 - S_h(\rho) \geqq \frac{K}{h}.$$

Now Lemma 1 gives

$$x_h - \rho \geqq \frac{K}{S_h'(x_h)h} \geqq \frac{K}{Ch^2} \geqq \frac{C_1}{h^2},$$

or

$$x_h \geqq \rho\left(1 + \frac{C_2}{h^2} \right).$$

Finally for any x, $0 < x < 1$,

$$S_h^n \leqq S_h(x) x^{-n},$$

so

$$S_h^n \leqq S_h(x_h) x_h^{-n} \leqq \rho^n (1 + C_2 h^{-2})^{-n} = O(T_n n^{3/2} \exp(-\delta n/h^2)).$$

9. Concluding remarks. The algorithm described in this paper has been implemented in BLIS10 on a DEC-System-10 (KL) at Vanderblit University. The program generates trees at the rate of 15-20 thousand trees per second, irrespective of n.

REFERENCES

[1] T. BEYER AND S. M. HEDETNIEMI, *Constant time generation of rooted trees*, this Journal, 9 (1980), pp. 706–712.

[2] E. A. DINITS AND M. A. ZAITSEV, *Algorithm for the generation of nonisomorphic trees*, Avtomat. Telem., 4 (1977), pp. 121–126; Automatic and Remote Control, 38 (1977), pp. 554–558.

[3] A. V. KOZINA, *Coding and generation of nonisomorphic trees*, Kibernetika, 5 (1979), pp. 38–43; Cybernetics, 15 (1975), pp. 645–651.

[4] R. OTTER, *The number of trees*, Ann. Math., 49 (1948), pp. 583–599.

[5] G. PÓLYA, *Kombinatorische Anzahlbestimmungen für Gruppen, Graphen und chemische Verbindungen*, Acta Math., 68 (1937), pp. 145–254.

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

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

[8] H. S. WILF, *The uniform selection of free trees*, J. Algorithms, 2 (1981), pp. 204–207.