Constant Time Generation of Free Trees

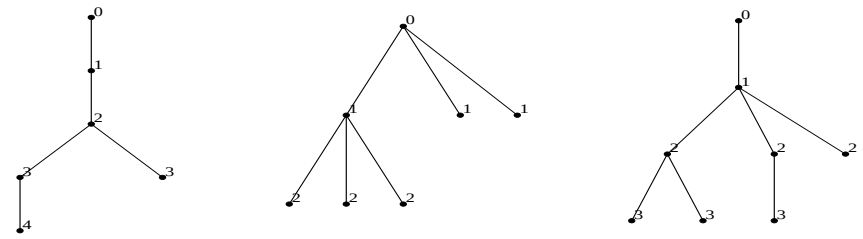
Michael J. Dinneen
Department of Computer Science
University of Auckland

Class: Combinatorial Algorithms – CompSci 720

1 Introduction

This is a combinatorial algorithms report on generating unlabeled free trees in constant amortized time and $O(n)$ space. A CAT (Constant Amortized Time) algorithm given by Wright, Richmond, Odlyzko and McKay [WROM] is presented. (See also the nice presentation given in [Wi].) This algorithm uses a modified successor function that Beyer and Hedetniemi used for constant-time generation of rooted trees [BH]. In both of the these algorithms, the idea of using integer sequences to represent objects being generated was introduced by Ruskey and HU for generating binary trees lexicographically (see [RH] and [Ru]).

Some graph theory definitions are now reviewed. A **tree** or **free tree** is a connected graph $T = (V, E)$ such that the number of edges (size) of T, denoted by $|E|$, is exactly one less then the number of vertices (order) of T, denoted by $|T|$ or $|V|$. If the context is clear we will use n to denote the order. Note that we are excluding multiple edges and loops in our implicit definition of graph. A **rooted tree** (T, z) is free tree T with a designated root z chosen from $V(T)$. We assign an unique **layout** of a rooted tree (T, z) as the lexicographically greatest distance sequence $L(T, z) = [l_1, l_2, l_3, \ldots, l_n]$ resulting over all preorder traversals (depth-first searches) of tree T starting from vertex z. Three such examples are shown below in Figure 1. (The roots in these examples are the ones labeled with distance 0.)



$[0, 1, 2, 3, 4, 3]$ $[0, 1, 2, 2, 2, 1, 1]$ $[0, 1, 2, 3, 3, 2, 3, 2]$

Figure 1: How to assign integer sequences to rooted trees.

The **center** of a free tree $T = (V, E)$ is the set of vertices whose maximum distance from the other vertices is least. The center of a tree will contain either one or two vertices. In the bicentral case the two centers will be adjacent in the graph. (A simple algorithm that deletes all leaves of a tree and repeats on the resulting tree until at most one edge remains can be can be used to find centers.)

The **primary root** of a free tree is defined by two cases. If a tree $T = (V, E)$ has a single center then it is the primary root. Otherwise one of the centers r_1 or r_2 of T will be chosen. Let $T_1 = (V_1, r_1)$ and $T_2 = (V_2, r_2)$ be the resulting two rooted trees when the edge (r_1, r_2) of T is deleted. The primary root of T will be r_2 if and only if $|T_1| < |T_2|$, or $|T_1| = |T_2|$ and $L(T_1, r_1) < L(T_2, r_2)$.

Asside: One can do isomorphism testing for free trees with an isomorphism algorthm for rooted trees by using the primary root as their roots (since they are uniquely defined). Thus, if two root trees have the same lexicographically greatest distance sequences then they are isomorphic.

2 Algorithm

We first present a successor function that generates nonisomorphic rooted trees, [BH]. Let $L = [l_1, l_2, \ldots, l_n]$ be a layout of a rooted tree of order n. Let p be the largest integer (subscript) such that $l_p \neq 1$ and let q be the largest integer such that $q < p$ and $l_q = l_p - 1$. (Note that the vertex indexed by q is the parent of the vertex indexed by p.) The successor $s(L) = [s_1, s_2, \ldots, s_n]$ of L is given by:

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

To generate all rooted trees of order n we start with the first rooted tree with layout $L = [0, 1, 2, \ldots, n-1]$ and repeatedly apply the seccessor function until the last rooted tree $[0, 1, 1, \ldots, 1]$ appears in the enumeration.

We now present an algorithm that generates free trees based on the enumeration procedure for rooted trees. If at least two ones occur in the layout L, then define $L_1 = [l_2 - 1, l_3 - 1, \ldots, l_{m-1} - 1]$ and $L_2 = [l_1, l_m, l_{m+1}, \ldots, l_n]$ where m is the index of the second 1 in L. Wright, Richmond, Odlyzko and McKay use the following necessary and sufficient test to determine when L represents a free tree rooted at its primary root:

(a) index m exists,
(b) $\max(L_2) \geq \max(L_1)$,
(c) if equality holds in (b), then $|L_1| \leq |L_2|$,
(d) if equality holds in (c), then $L_1 \leq L_2$ lexicographically.

The free tree that this algorithm starts with is the integer sequence of an n-path rooted at its primary root. Starting at this tree guarantees that condition (a) never fails. The other cases may fail and the detection can be made before the successor function is applied. Figure 2 shows some examples and the modifications needed to keep the algorithm generating only primary rooted free trees. Note that in the figure, the indicated condition fails for the successor of the trees listed on the left side of the arrows (not the valid free trees displayed).

There is a general procedure for doing these "jumps" down the list of rooted trees when a failure condition is detected. First, we set the parameter p for the successor fuction to the last node p' of L_1 and apply the succesor function on L to get $s(L)$. Second, if $l_{p'}$ was greater than 2 in L then we replace the final elements of $s(L)$ with $1, 2, \ldots, height(L_1') + 1$, where L_1' is the new left subtree of $s(L)$. The intuitive reason for using the successor funcion in this way is to find the next lexicographically greatest sequence that is a primary rooted free tree. The second step guarantees that the tree is rooted at a center vertex.

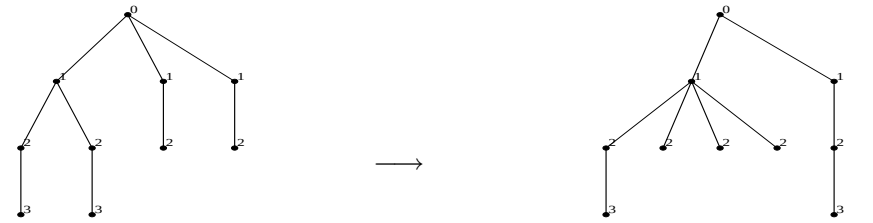
Condition (b) fails: (The height of L_2 is reduced too much.)

$[0, 1, 2, 3, 3, 1, 2, 1, 1, 1]$ $[0, 1, 2, 3, 2, 3, 2, 1, 2, 3]$

Condition (c) but not (b) fails: (Successor causes $|L_2| < |L_1|$ in bicentral case.)

$[0, 1, 2, 2, 2, 2, 2, 1, 2, 1]$ $[0, 1, 2, 2, 2, 2, 1, 2, 2, 2]$

Condition (d) alone is violated: (Only happens when $L_1 \simeq L_2$.)



$[0, 1, 2, 3, 2, 3, 1, 2, 1, 2]$ $[0, 1, 2, 3, 2, 2, 2, 1, 2, 3]$

Figure 2: Corrections when the rooted tree successor function fails.

3 Results

Let t_n and T_n denote the number of free and rooted unlabeled trees of order n, respectively. Wright, Richmond, Odlyzko and McKay use the following result of Pólya [Po] and Otter [Ot] to prove their algorithm runs in constant amortized time.

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

I have implemented the constant time free tree algorithm by Wright, Richmond, Odlyzko and B.D. McKay and did some benchmarks to determine, at least empirically, if it runs in constant amortized time. An affirmative indication resulted! Also I have checked the correctness of the output for free trees with 10 and fewer vertices. This check entailed a comparison of the trees generated by the algorithm in [WROM] and the trees contained in Read's catalog of all non-isomorphic graphs on 10 vertices, [CCRW].

The following table and figure summarizes my experiments. The start-up CPU time was included in the times indicated. So it is reasonable to expect (as shown by my crude `/usr/bin/time` technique) for the amount of CPU per tree to "slightly" decrease when generating larger trees.

Order	No. of Trees	CPU Time	CPU per Tree
1	1		
2	1		
3	1		
4	2		
5	3		
6	6		
7	11		
8	23		
9	47		
10	106		
11	235		
12	551		
13	1301		
14	3159	0.1μ	0.0045μ
15	7741	0.3μ	3.88e-05μ
16	19320	0.7μ	3.62e-05μ
17	48629	1.8μ	3.70e-05μ
18	123867	4.7μ	3.79e-05μ
19	317955	11.7μ	3.71e-05μ
20	823065	30.4μ	3.69e-05μ
21	2144505	78.2μ	3.65e-05μ
22	5623756	203.4μ	3.62e-05μ
23	14828074	532.4μ	3.59e-05μ
24	39299897	1401.8μ	3.57e-05μ
25	104636890	3756.8μ	3.59e-05μ
26	279793450	9136.7μ	3.27e-05μ

Figure 2: CPU time needed to count free trees on a Sparc-2.

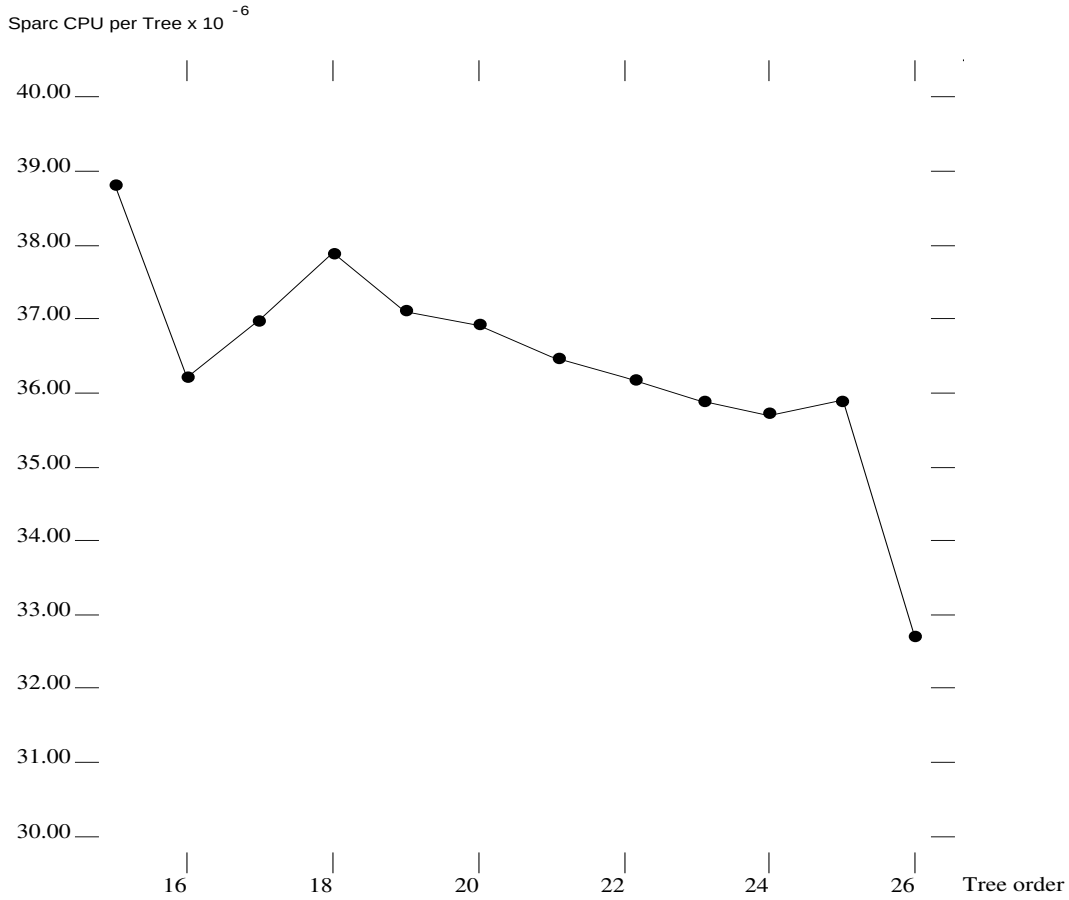


Figure 3: Scaled plot of average time per tree per tree order.

The subsequent pages of this report contain the following items:

1. Listings of all rooted trees of order 5 through 8 generated by the algorithm by Beyer and Hedetniemi.

2. Listings of all free trees of order 5 through 10 generated by the algorithm by Wright, Richmond, Odlyzko and B.D. McKay.

3. Program listings of the above two implementations. (Slightly altered code was used for the CPU timings.)

References

[BH] T. Beyer and S.M. Hedetniemi, *Constant time generation of rooted trees*, SIAM Journal of Computing, **9** (1980), pp. 706–712

[CCRW] R.D. Cameron, C.J. Colbourn, R.C. Read, N.C. Wormald, Cataloguing the Graphs on 10 Vertices, Journal of Graph Theory, 9 (1985) pp. 551–562.

[Tape announcement, Discrete Math. 31 (1980) p. 224.]

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

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

[RH] F. Ruskey and T.C. Hu, *Generating binary trees lexicographically*, SIAM Journal of Computing, **6** (1977), pp. 745–758.

[Ru] F. Ruskey, *Generating t-ary trees lexicographically*, SIAM Journal of Computing, **7** (1978), pp. 424-439.

[WROM] R.A. Wright, B. Richmond, A. Odlyzko and B.D. McKay, *Constant time generation of free trees*, SIAM Journal of Computing, **15** (1986), pp. 540–548.

[Wi] H.S. Wilf, "Listing Free Trees," in *Combinatorial Algorithms (An Update)*, pp. 31-36.