Loopless Generation of Trees with Specified Degrees

JAMES F. KORSH AND PAUL LAFOLLETTE

Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA
Email: korsch@temple.edu, lafollet@joda.cis.temple.edu

An ordered tree with specified degrees and n nodes has a_i nodes of degree i where $a_0 = 1 + \sum_{i=1,h}(i-1)a_i$ and $n = \sum_{i=0,h} a_i$. This paper presents a new and simpler loopless algorithm for generating all ordered trees with specified degrees. When $a_k = N$, $a_0 = (k-1)N + 1$ and all other a_i's are 0, then all N node k-ary trees are generated.

Received 13 June 2000; revised 27 November 2001

1. INTRODUCTION

Ehrlich [1] discussed loopless algorithms for generating combinatorial objects. These algorithms generate each combinatorial object from its predecessor in no more than a constant time and hence can be written with no loops. Many combinatorial objects have now been shown to have such loopless generation algorithms. These include permutations [1, 2], multi-set permutations [3], combinations [1, 3, 4, 5], subsets [6], integer partitions [7, 8], binary and k-ary trees [9, 10, 11, 12, 13], linear extensions of posets (topological sorts) [14], reflected Gray codes [15] and, more recently, trees with specified degrees.

A combinatorial Gray code is a listing of combinatorial objects in which only a small change takes place from one object to its successor. An excellent survey is given in Savage [16]. Combinatorial Gray codes may not be loopless and loopless algorithms may not produce Gray codes. However, each may be helpful in the development of the other. Moreover, loopless algorithms are challenging to develop and can serve as good examples of the application of data structure techniques.

It is important to recognize that generation algorithms for combinatorial objects can depend on the representation used for the objects. Combinatorial Gray codes and loopless algorithms for one representation need not be combinatorial Gray codes or loopless algorithms for another representation.

An ordered tree with specified degrees and n nodes has a_i nodes of degree i where $a_0 = 1 + \sum_{i=1,h}(i-1)a_i$ and $n = \sum_{i=0,h} a_i$. The degree of a node is the number of its subtrees. The number of such ordered trees has been shown by Raney [17] to be $(n-1)!/(\prod_{i=0,h} a_i!)$. Algorithms have appeared which generate ordered trees with specified degrees [18, 19], but these are not loopless. The existence of such an algorithm was stated as an open problem in Roelants van Baronaigien [11]. Recently, the loopless algorithm of [13] was interleaved with a new version of the loopless multi-set generation algorithm of [3] to obtain the first loopless algorithm for this problem [20]. Another loopless algorithm was obtained [21] using the algorithm of [14] as a different loopless generation algorithm for the multi-sets. Both essentially carry out the transformations based on shifts described in [10], but applied to trees whose nodes do not all have the same degree.

The algorithm developed here is not based on shifts. Instead it generates a series of tree representations, each lexicographically. It still requires constant time generation of multi-sets. The algorithm is loopless and uses storage linear in n.

The motivation for the representation used here was to try to generalize the results of [9] and [11]. The purpose of developing this loopless algorithm, based on the representation introduced here, is part of an ongoing attempt to better understand the nature of loopless algorithms and combinatorial Gray codes and of those properties of combinatorial objects (and particular representations of those objects) which facilitate their development.

In this case, for instance, it is interesting that, when all nodes have the same degree, the algorithm appears to be the same as that in [11] for binary trees and as [9] for k-ary trees since it generates exactly the same representations and in the same order. However, it is not at all the same since those representations correspond to different trees and do not extend to trees whose nodes have unequal degrees.

2. REPRESENTATION

A *degree sequence d* and a *chain length sequence c* are used in this paper for the representation of an ordered tree T with specified degrees. The d sequence is of length m, the number of internal nodes in T, $(\sum_{i=1,h} a_i)$.

The sequence $d = d_1 d_2 \ldots d_m$ is obtained by traversing T in preorder and for each internal node writing down its degree. Figure 1 shows a tree T with $a_0 = 12$, $a_1 = 2$, $a_2 = 4$, $a_3 = 2$, $a_4 = 1$ and $m = 9$ whose d sequence is 1 3 2 1 2 3 4 2 2. Terminal nodes are not shown. The node corresponding to d_i will be referred to as *node i*, $1 \leq i \leq m$.

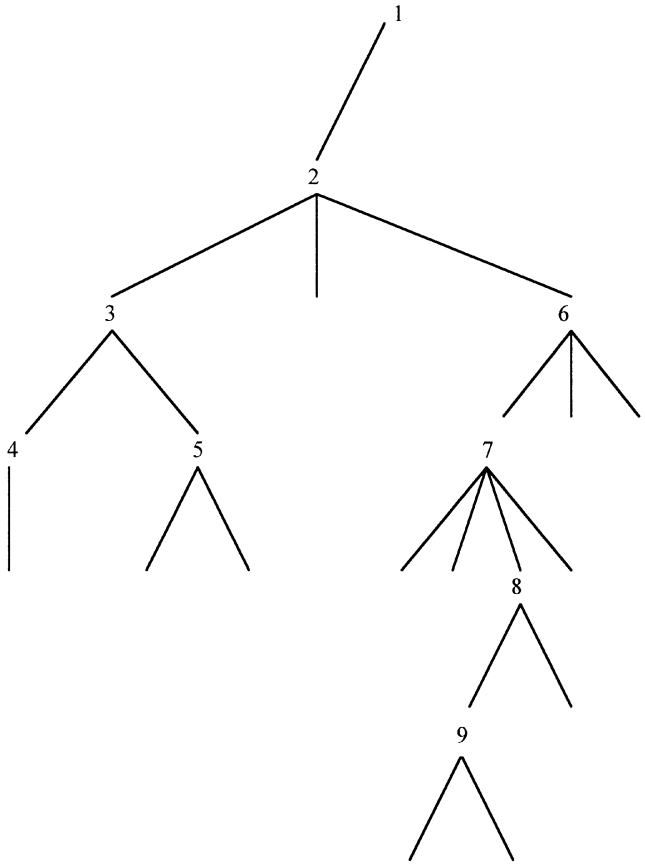

FIGURE 1.

TABLE 1.

Node	1	2	3	4	5	6	7	8	9
d	1	3	2	1	2	3	4	2	2
c		0 2	1		0	0 0	0 2 0	0	0
q		2 2	3		5	6 6	7 7 7	8	9
t		1 1	3		4	5 5	7 7 7	10	11
Position		1 2	3		4	5 6	7 8 9	10	11

In general, d will be a multi-set permutation of a_i i's, $i = 1, h$.

Some terminology is needed before c sequences can be defined.

DEFINITION 2.1. *A* chain *of length k is a sequence of nodes, $n_1, n_2, \ldots, n_{k+1}$, where:*

- *for $i = 1, 2, \ldots, k$, node n_i is an internal node;*
- *for $i = 1, 2, \ldots, k$, node n_{i+1} is the first child of n_i;*
- *n_1 is not the first child of its parent; and*
- *node n_{k+1} is a leaf.*

The top node *of the chain is n_1 and we can associate leaf n_{k+1} uniquely with its chain.*

A node of T whose degree exceeds 1 has more than one chain associated with it, each one corresponding to one of its subtrees, other than the first. The top node of the chain is the root of the subtree. If the chain has length greater than zero, its top node is an internal node; otherwise, the top node is a leaf node. In Figure 1 the chains of length greater than zero are 6–7, 5 and 8–9. We say the chains associated with a node *belong* to the node.

The sequence $c = c_1 c_2 \ldots c_r$ is obtained by first traversing T in preorder and, for each internal node except the last, writing down for each of its subtrees, other than the first, the length of its chain. The last internal node contributes a final 0, so c_r is always 0. This imposes an ordering on the chains: chain j is the chain whose length is c_j. Although each terminal node corresponds to a unique chain, we will only be interested in $r - 1$ of the chains and will exclude those belonging to the last node. Thus r is the number of terminal nodes, a_0, minus $(p_m - 1)$. Nodes of degree one will not contribute to the c sequence. T of Figure 1 has c sequence 0 2 1 0 0 0 0 2 0 0 0 with $r = a_0 - (p_m - 1) = 12 - (2 - 1) = 11$.

Two additional sequences, q and t, will be needed. Each can be derived from d. The sequence $q = q_1 q_2 \ldots q_r$ is obtained by setting q_j to the parent of the top of the jth chain. The following process produces q from p: replace d_j by $d_j - 1$ copies of j. The d sequence of Figure 1 has q sequence 2 2 3 5 6 6 7 7 7 8 9.

Each distinct value of the q sequence corresponds to a distinct node of the d sequence with degree greater than one. The q sequence above for Figure 1 has distinct values 2, 3, 5, 6, 7, 8 and 9 corresponding, respectively, to nodes 2, 3, 5, 6, 7, 8 and 9. The number of such distinct values is $m - a_1$. Also, q_1 is the first node of T whose degree is not one. In Figure 1, q_1 is node 2.

The sequence $t = t_1 t_2 \ldots t_r$ is obtained by setting t_j to the smallest k such that $q_k = q_j$ for $1 \leq j \leq r$. The following process produces t from p: replace d_j by $d_j - 1$ copies of $\sum_{k=1, j-1}(d_k - 1) + 1$. It is always the case that t_1 is 1.

The q sequence of Figure 1 has the t sequence 1 1 3 4 5 5 7 7 7 10 11. Note that t_j specifies where, in q, to find the first occurrence of the parent of the top of the jth chain. It also specifies where, in c, to find the length of the first chain of this parent. This is illustrated in Table 1 for Figure 1. For instance, the first occurrence of node 6, the parent of the top of the 5th (and 6th) chain, is in position 5 of q and the length of 6's first chain is in position 5 of c, as specified by $t_5 = t_6 = 5$.

We define *predj* to be $t_j - 1$, the last chain of q_j's predecessor and j' to be $t_j +$ (degree of $q_j - 2$), the last chain of q_j. Because t_j specifies the first chain of q_j, whenever chain j belongs to $q_j, t_j - 1$ specifies the last chain of q_{predj}, so chain $t_j - 1$ belongs to q_{predj}. In Figure 1, since $j = 7$, 8 and 9 belong to node 7, $t_7 - 1 = t_8 - 1 = t_9 - 1 = 6$ and $q_{predj} = q_6$, node 6. Also, $j' = t_j +$ (degree of $q_j - 2$) $= 7 + (4 - 2) = 9$, the last chain of node 7. If $j = 4$, 4 belongs to node 5 and chain $t_j - 1 = 4 - 1 = 3$, so chain 3 belongs to $q_{predj} = q_3$, node 3.

Define *chain 0 of T* to be the sequence of internal nodes visited by starting at the root and visiting all first children

```
convert(k, length, j, root, lastchain, last)

create a node of degree d_k and set root to it
if (length is 1)
   make root's first child a leaf node
   set last to k. Set i to k
else
   convert(k+1, length-1, j+d_k-1, t, lastchain, last)
   set the first subtree of root to t
   set nextj to lastchain+1 and i to last+1
   for s from 2 to d_k
      if (c_{j+1} > 0)
         convert(i, c_{j+1}, t, nextj, lastchain, last)
         set subtree s of root to t
         set nextj to lastchain+1 and i to last+1
      else
         make child s of t a leaf node
         set j to j+1
      set lastchain to nextj
```

ALGORITHM 2.1.

until a leaf is reached. In Figure 1 chain 0 consists of nodes 1, 2, 3 and 4.

The total number of internal nodes in T, excluding those on its chain 0, is given by $\sum_{i=1}^{r} c_i$. Consequently, the number of nodes on chain 0 must be $m - \sum_{i=1}^{r} c_i$. For T of Figure 1, $\sum_{i=1}^{r} c_i$ is 5 and $m - \sum_{i=1}^{r} c_i = 9 - 5 = 4$, the number of internal nodes on chain 0.

More generally, $\sum_{i=t_j}^{r} c_i$ is the total number of internal nodes appearing in chains that belong to q_j and the nodes that follow q_j in a preorder traversal of T. In Figure 1, for example, the number of internal nodes appearing in chains that belong to q_3 and nodes following q_3 is 3. These are nodes in chains 5 and 8–9 (nodes in chain 6–7 belong to node 2 not node 3).

We now give a recursive O(n) algorithm to convert the d, c representation of T to a standard linked representation of T. The algorithm first creates the root node, and then, starting with the first and ending with the last, creates and appends each of the root's subtrees to T.

The first three parameters are k, length and j. Node k is the node that is to be at the root of the tree, length is the length of the chain whose top node is k and j is the chain number of k's first chain. When the algorithm completes, root will point to node k, last will contain the number of the last node of root and lastchain will contain the number of last's last chain.

To create T, invoke `convert(1, m-`$\sum_{i=1}^{r} c_i$`, 0, T, lastchain, last)`, as in Algorithm 2.1.

It is now possible to characterize chain length sequences.

LEMMA 2.1. *A sequence c of r non-negative integers is a chain length sequence iff $\sum_{i=t_j}^{r} c_i \leq m - q_j$ for $1 \leq j \leq r$.*

Proof. Two things must be shown: (1) if c is a chain length sequence, i.e. it came from a tree, then c must satisfy the constraints of the lemma; and (2) if the conditions are met then c must be a chain length sequence, i.e. it must have come from a tree.

To show (1), we have already noted that $\sum_{i=t_j}^{r} c_i$ is the total number of internal nodes appearing in chains belonging to q_j and nodes that follow q_j in a preorder traversal of T. There are exactly $m - q_j$ nodes following q_j in the traversal, and only they can appear in these chains. Hence $\sum_{i=t_j}^{r} c_i$ is at most $m - q_j$. Thus c must satisfy the constraints $\sum_{i=t_j}^{r} c_i \leq m - q_j$ for $1 \leq j \leq r$.

We accomplish (2) by showing that the convert algorithm above will be able to produce the tree when c satisfies the constraints. The only reason the algorithm might not successfully produce the tree is if it runs out of available nodes as it creates the chains. When a node is processed by the algorithm, a chain is created only if the node has degree greater than one. Consider the situation when node q_j, which must have degree greater than one, is added. The tree then contains only internal nodes q_1 through q_j of degree greater than one and these nodes all appear in chains associated with nodes prior to q_j. Others of degree one may also appear in these chains. At this point, the number needed for all the chains associated with nodes q_j through q_m is $\sum_{i=t_j}^{r} c_i$. However, the constraints again ensure that this does not exceed $m - q_j$, which is exactly the number still available. Adding node q_j does not affect the requirements on chains associated with earlier nodes since it appears on one of those chains. This completes the proof. □

The bound of the lemma is actually achieved for all j by the tree in which node $i + 1$ is the last child of node i, for $i = 1, \ldots, m - 1$.

3. GENERATION OF *C* SEQUENCES FOR A FIXED PERM

In this section the generation of all c sequences that correspond to a particular d sequence is considered. The

goal is to generate them in lexicographical order. The discussion that follows was motivated by [9, 11], although their sequences represent k-ary and binary trees in an entirely different fashion based on rotations and do not extend to trees with specified degrees.

Lemma 2.1 imposes constraints on the partial sum $\sum_{i=t_j}^{r} c_i$. The lexicographically smallest c sequence consists entirely of 0's and the largest is $m - q_1$ followed entirely by 0's.

In order to develop a loopless algorithm, information must be available about the value that certain sums of the c_j's can attain. This will be presented in Lemma 3.1, which requires another sequence, M.

Now $\sum_{i=t_j}^{j} c_i$ is the number of nodes in chains t_j to j belonging to q_j for $t_j \leq j \leq j'$. $\sum_{i=t_x}^{t_j-1} c_i$ is the number of nodes in chains belonging to nodes q_x to q_{predj}. For example, if x is 2 and j is 5, then in Figure 1, the number of nodes in chains belonging to nodes q_2 to q_4 (nodes 2, 3 and 5) is three.

Define M_j to be the largest that $\sum_{i=1}^{j'} c_i$ can be, given the fixed values of c_i for i from 1 to $t_j - 1$, so that $c_1 c_2 \ldots c_j$ can be extended to a chain sequence. Thus M_j is the largest that $\sum_{i=1}^{t_j-1} c_i + \sum_{i=t_j}^{j'} c_i$ can be given $\sum_{i=1}^{t_j-1} c_i$. Now, by Lemma 2.1, $c_1 c_2 \ldots c_j$ can be extended to a chain sequence iff $\sum_{i=t_x}^{t_j-1} c_i + \sum_{i=t_j}^{j'} c_i \leq m - q_x$ for all x, $1 \leq x \leq j$. Hence

$$\sum_{i=t_j}^{j'} c_i \leq \min_{1,j} \left\{ m - q_x - \sum_{i=t_x}^{t_j-1} c_i \right\}$$

where $\min_{1,j}$ denotes the minimum over x from 1 to j. Consequently,

$$\begin{aligned} M_j &= \sum_{i=1}^{t_j-1} c_i + \min_{1,j} \left\{ m - q_x - \sum_{i=t_x}^{t_j-1} c_i \right\} \\ &= \min_{1,j} \left\{ \sum_{i=1}^{t_x-1} c_i + m - q_x \right\} \\ &= \min \left\{ M_{j-1}, \sum_{i=1}^{t_j-1} c_i + m - q_j \right\}. \end{aligned}$$

This gives Lemma 3.1.

LEMMA 3.1.

$$M_j = \min \left\{ M_{j-1}, \sum_{i=1}^{t_j-1} c_i + m - q_j \right\} \quad \text{for } j > 1 \text{ and } M_1 = m - q_1.$$

M_r is always $\sum_{i=1}^{r} c_i$. Clearly, whenever $q_j = q_{j+1}$, $M_j = M_{j+1}$ and $M_1 \geq M_2 \geq \ldots \geq M_r$, so M is a non-increasing sequence.

For c of Figure 1, $\sum_{i=1}^{t_j-1} c_i + m - q_j$ is $9 - 2 = 7$ for $j = 1, 2 + 9 - 3 = 8$ for $j = 3, 3 + 9 - 5 = 7$ for $j = 4, 3 + 9 - 6 = 6$ for $j = 5, 3 + 9 - 7 = 5$ for $j = 7, 5+9-8 = 6$ for $j = 10$ and $5+9-9 = 5$ for $j = 11$. So, $M_1 = M_2 = 7$, M_3 is 7, M_4 is 7, $M_5 = M_6 = 6$, $M_7 = M_8 = M_9 = 5$, M_{10} is 5 and M_{11} is 5 and the M sequence is 7 7 7 7 6 6 5 5 5 5 5.

Given c, define its *active index* $J(c)$ as the largest j for which

$$\sum_{i=1}^{t_j-1} c_i + \sum_{i=t_j}^{j'} c_i = \sum_{i=1}^{j'} c_i$$

has *not* attained its limit, M_j. Then $c_{J(c)+1} \geq 0$. For $i > J(c) + 1$, all the c_i must be 0. Consequently, $\sum_{i=1}^{r} c_i$ must equal $M_{J(c)+1}$.

The lexicographic successor of c, c', is obtained by setting $c'_{J(c)}$ to $c_{J(c)+1}$ and $c'_{J(c)+1}$ to 0. Then $\sum_{i=1}^{r} c'_i = \sum_{i=1}^{r} c_i + 1 - c_{J(c)+1}$ and, if it has reached its limit $M_{J(c)}$, then $J(c')$ will be $J(c)-1$; otherwise $J(c')$ will be $r-1$. Thus, c' can be obtained from c by incrementing one component, $c_{J(c)}$, by 1 and setting its right neighbor to 0; $J(c)$ specifies where to make these changes. Furthermore, $J(c')$ can be determined from $J(c)$. This is summarized in Lemma 3.2.

LEMMA 3.2. *Let c be a chain length sequence and c' its lexicographic successor. Let $j = J(c)$. Then*

$$c'_i = \begin{cases} c_i + 1 & \text{if } i = j, \\ 0 & \text{if } i = j + 1, \\ c_i & \text{otherwise.} \end{cases}$$

Furthermore,

$$J(c') = \begin{cases} j - 1 & \text{if } c'_j = M_{J(c)}, \\ r - 1 & \text{if } c'_j < M_{J(c)}. \end{cases}$$

Proof. All c_i must be 0 for $i > j + 1$. If one of these c_i, say c_k, were not 0, then its limit

$$M_k = \sum_{i=1}^{k'} c_i > M_{j+1} = \sum_{i=1}^{(j+1)'} c_i.$$

However, the M sequence is non-increasing. Since these c_i are 0, $\sum_{i=1}^{r} c_i$ must equal M_{j+1}. Now, suppose that some $k < j$ has reached its limit M_k. Then

$$M_k = \sum_{i=1}^{k'} c_i \leq \sum_{i=1}^{j'} c_i \leq \sum_{i=1}^{(j+1)'} c_i = M_{j+1}.$$

However, since $M_k > M_{j+1}$, it follows that $M_k = M_{j+1}$ and so j must have reached its limit M_j. This contradicts the fact that j is the active index. Consequently, no chains less than j can have reached their limits.

The next smallest value that c_j can take and not exceed M_j is $c_j + 1$, and, for $i \geq j + 1$, the smallest value c_i can be is 0. Since they already are 0 except possibly for c_{j+1}, we ensure that they all are 0 by setting c_{j+1} to 0. This sequence c' is then a chain length sequence and is clearly the lexicographic successor to c.

Now, $\sum_{i=1}^{k'} c_i = \sum_{i=1}^{r} c_i$ for all $k > j+1$ and, since they all reached their limit, $M_k = M_r$ for all $k > j + 1$. Also,

$$\sum_{i=1}^{j'} c'_i = \sum_{i=1}^{k'} c'_i = \sum_{i=1}^{r} c'_i = \sum_{i=1}^{r} c_i + 1 - c_{j+1}$$

```
while (j>0)
   set sum to sum+1-c_{j+1}, c_J to c_j+1 and then c_{j+1} to 0
   determine M_j
   if(sum is equal to M_j)
      set j to j-1
   else
      set M_r to sum and j to r-1
```

ALGORITHM 3.1.

p : 3 2 1 3 2

c	M	$\underline{M}$	c	M	$\underline{M}$	c	M	$\underline{M}$
0 0 0 0 0 0	443110	0	0 2 1 0 1 0	444444	444444	1 1 2 0 0 0	444444	444444
0 0 0 0 1 0	443221	111	0 2 1 1 0 0	444444	444444	1 2 0 0 0 0	444443	444443
0 0 0 1 0 0	443221	111	0 2 2 0 0 0	444444	444444	1 2 0 0 1 0	444444	444444
0 0 1 0 0 0	443221	331	0 3 0 0 0 0	444443	444443	1 2 0 1 0 0	444444	444444
0 0 1 0 1 0	443222	3222	0 3 0 0 1 0	444444	444444	1 2 1 0 0 0	444444	444444
0 0 1 1 0 0	443222	3222	0 3 0 1 0 0	444444	444444	1 3 0 0 0 0	444444	444444
0 0 2 0 0 0	443332	3332	0 3 1 0 0 0	444444	444444	2 0 0 0 0 0	444332	444442
0 0 2 0 1 0	443333	3333	0 4 0 0 0 0	444444	444444	2 0 0 0 1 0	444333	444333
0 0 2 1 0 0	443333	3333	1 0 0 0 0 0	444221	444441	2 0 0 1 0 0	444333	444333
0 0 3 0 0 0	443333	3333	1 0 0 0 1 0	444222	444222	2 0 1 0 0 0	444333	444443
0 1 0 0 0 0	444222	444331	1 0 0 1 0 0	444332	444222	2 0 1 0 1 0	444444	444444
0 1 0 0 1 0	444222	444222	1 0 1 0 0 0	444332	444442	2 0 1 1 0 0	444444	444444
0 1 0 1 0 0	444222	444222	1 0 1 0 1 0	444333	444333	2 0 2 0 0 0	444444	444444
0 1 1 0 0 0	444332	444222	1 0 1 1 0 0	444333	444333	2 1 0 0 0 0	444443	444443
0 1 1 0 1 0	444333	444333	1 0 2 0 0 0	444443	444443	2 1 0 0 1 0	444444	444444
0 1 1 1 0 0	444333	444333	1 0 2 0 1 0	444444	444444	2 1 0 1 0 0	444444	444444
0 1 2 0 0 0	444443	444443	1 0 2 1 0 0	444444	444444	2 1 1 0 0 0	444444	444444
0 1 2 0 1 0	444444	444444	1 0 3 0 0 0	444444	444444	2 2 0 0 0 0	444444	444444
0 1 2 1 0 0	444444	444444	1 1 0 0 0 0	444332	444442	3 0 0 0 0 0	444443	444443
0 1 3 0 0 0	444444	444444	1 1 0 0 1 0	444333	444333	3 0 0 0 1 0	444444	444444
0 2 0 0 0 0	444332	444422	1 1 0 1 0 0	444333	444333	3 0 0 1 0 0	444444	444444
0 2 0 0 1 0	444333	444333	1 1 1 0 0 0	444443	444443	3 0 1 0 0 0	444444	444444
0 2 0 1 0 0	444333	444333	1 1 1 0 1 0	444444	444444	3 1 0 0 0 0	444444	444444
0 2 1 0 0 0	444443	444433	1 1 1 1 0 0	444444	444444	4 0 0 0 0 0	444444	444444

FIGURE 2.

for all $k > j$. Let M' denote the new M sequence for c'. Then $M'_j = M_j$. Now,

$$\sum_{i=1}^{j'} c'_i = \sum_{i=1}^{k'} c'_i = \sum_{i=1}^{r} c'_i = \sum_{i=1}^{r} c_i + 1 - c_{j+1} = M'_r$$

for all $k > j$. So, if $\sum_{i=1}^{j'} c'_i$ has reached its limit M_j, then so has $\sum_{i=1}^{k'} c'_i$ for all $k > j$. Hence, $J(c')$ will be $J(c) - 1$. If $\sum_{i=1}^{j'} c'_i$ has not reached its limit, then chain r has (it always has) and $J(c')$ will be $r - 1$.

For c of Figure 1, $J(c)$ is 7 so the successor of c is 0 2 1 0 0 0 1 0 0 0 0, $J(c')$ is 10 and M' is 7 7 7 7 6 6 5 5 5 5 4.

Lemma 3.2 provides the basic algorithm for generating all the c sequences in lexicographical order that correspond to a given p sequence. Let the sum be $\sum_{i=1}^{r} c_i$ and let j be the active index. The algorithm is as in Algorithm 3.1.

Figure 2 gives the complete listing of the c sequences for the d sequence 3 2 1 3 2 and also shows the M sequences. It contains $\underline{M}$ too, which will be introduced next and reconsidered again in Section 4. The $\underline{M}$ values not shown are -1.

M' cannot be generated from c in constant time although c' can be. Fortunately, M' itself is not required but M_j is, and it must be determined in constant time. Lemma 3.3 provides one way to do this by updating M_j when it is needed and allows this to be done in constant time as long as $\sum_{i=t_j}^{j} c_i$ is available. However, another method will be used. From here on we let $S_j = \sum_{i=t_j}^{j} c_i$ for $t_j \leq j \leq j'$.

Let $\underline{M}_j$ denote the current estimate of M_j for c. Initially, $c_j = 0$ and $\underline{M}_j = m - q_j$ for $1 \leq j \leq r$. The $\underline{M}_j$ are thus initially set to the correct values of the M_j for all j. Define *delta* to be $q_{j+1} - q_j - 1$, the number of nodes of degree one between q_{j+1} and q_j. Whenever j is active, $\underline{M}_j$ and $\underline{M}_{j+1}$ will be updated as follows:

```
if (M_j > M_{j+1}+delta)
   set M_j to M_{j+1}+delta+1
set M_{j+1} to M_j
```

The updated $\underline{M}_j$ is then the current estimate of M'_j for c'. Note that $\underline{M}_j$ and $\underline{M}_{j+1}$ can change only when q_{j+1} is not equal to q_j.

LEMMA 3.3. *The updated $\underline{M}_j$ will always equal the correct value of M'_j for c'.*

Proof. The proof is by induction on the number of updates. Since initially $\underline{M}_j = m - q_j$ for $1 \le j \le r$, the first time each j becomes the active index the updating produces a correct $\underline{M}_j$.

Let j be the active index now and assume that all previous updates have been correct. Thus $\underline{M}_{j+1}$ must have its correct value at this time. Also, all $c_i = 0$ and $\underline{M}_i = \underline{M}_{j+1}$ for $j + 1 < i$.

If $\sum_{i=1}^{(j-1)'} c_i + S_j$ did *not* reach its limit the last time j was the active index, then $\underline{M}_j$ is the same as the last time and is correct, and $\underline{M}_{j+1}$ must be the same as before or increased by 1. Thus, updating will not change $\underline{M}_j$ so it will equal the correct value of M'_j for c'.

If $\sum_{i=1}^{(j-1)'} c_i + S_j$ *did* reach its limit the last time j was the active index, $j - 1$ would have become the active index and $\underline{M}_j$ may have been changed from its correct last value. This can only happen when

$$\sum_{i=1}^{(j-1)'} c_i + S_j = \underline{M}_{j+1}, \quad \sum_{i=1}^{(j-1)'} c_i + 1 = \underline{M}_j$$

and $j - 1$ becomes active. This can cascade with a series of contiguous k's, say $s + 1, \ldots, j - 1$, j becoming the active index one after another. Here $s + 1$ denotes the smallest such k. All S_k must be 1 for $s + 1 < k \le j$ and S_{s+1} may be greater than 1.

After the updating for $s + 1$ takes place, $\underline{M}_{s+1}$ is set correctly and $\underline{M}_{s+2}$ set to it. So at that point

$$\underline{M}_i = \underline{M}_j = \sum_{i=1}^{(j-1)'} c_i + 1, \text{ for } s + 1 \le i \le j$$

and

$$\underline{M}_j = \min\left\{\underline{M}_s, \sum_{i=1}^{(j-1)'} c_i + 1 - S_{s+1} + 1 + m - q_{s+1}\right\}.$$

$\underline{M}_s$ is updated next; it remains correct and unchanged and $\underline{M}_{s+1}$ is set to it. If

$$M_s \le \sum_{i=1}^{(j-1)'} c_i + 1 - S_{s+1} + 1 + m - q_{s+1}$$

then $\underline{M}_s = M_s = \underline{M}_j = \underline{M}_i$ for $s + 1 \le i \le j$; otherwise

$$\underline{M}_s = M_s > \sum_{i=1}^{(j-1)'} c_i + 1 - S_{s+1} + 1 + m - q_{s+1} = \underline{M}_j.$$

Now when updating for j, $\underline{M}_{j+1}$ will be correct. Since all the c_i are 0 for $s + 1 \le i \le j$,

$$\begin{aligned}\underline{M}_{j+1} &= M_{j+1} \\ &= \min\left\{M_s, \sum_{i=1}^{(j-1)'} c_i + 1 - S_{s+1} + 1 + m - q_{j+1}\right\}\end{aligned}$$

and $\underline{M}_s = M_s$, and

$$M_j = \min\left\{M_s, \sum_{i=1}^{(j-1)'} c_i + 1 - S_{s+1} + 1 + m - q_j\right\}.$$

Consequently, if $\underline{M}_{j+1}$ is $\underline{M}_s$ then so is M_j and

$$\begin{aligned}M_s &\le \sum_{i=1}^{(j-1)'} c_i + 1 - S_{s+1} + 1 + m - q_{j+1} \\ &< \sum_{i=1}^{(j-1)'} c_i + 1 - S_{s+1} + 1 + m - q_{s+1}.\end{aligned}$$

So $\underline{M}_j$ is also $\underline{M}_s$ and is correct. Since updating will not change $\underline{M}_j$, it will equal the correct value of M'_j for c'.

If $\underline{M}_{j+1}$ is not $\underline{M}_s$ then it must be

$$\sum_{i=1}^{(j-1)'} c_i + 1 - S_{s+1} + 1 + m - q_{j+1}$$

and so

$$\underline{M}_s = M_s > \sum_{i=1}^{(j-1)'} c_i + 1 - S_{s+1} + 1 + m - q_{j+1}.$$

Also,

$$\sum_{i=1}^{(j-1)'} c_i + 1 - S_{s+1} + 1 + m - q_{s+1} = \underline{M}_j.$$

Now

$$\begin{aligned}&\underline{M}_{j+1} + delta \\ &\quad = \sum_{i=1}^{(j-1)'} c_i + 1 - S_{s+1} + 1 + m - q_{j+1} + delta \\ &\quad = \sum_{i=1}^{(j-1)'} c_i + 1 - S_{s+1} + m - q_j.\end{aligned}$$

If $M_j = M_s$ then

$$M_j \le \sum_{i=1}^{(j-1)'} c_i + 1 - S_{s+1} + 1 + m - q_j$$

which is $\underline{M}_{j+1} + delta + 1$. Since $\underline{M}_j > \underline{M}_{j+1} + delta$, $\underline{M}_j$ will be updated to the correct value of M'_j for c'.

If

$$M_j = \sum_{i=1}^{(j-1)'} c_i + 1 - S_{s+1} + 1 + m - q_j,$$

```
enter the data
initialize d and c
generate all other c for d
while (there is another permutation of d)
   generate the next permutation of d
   reset c
   generate all other c for d
```

ALGORITHM 4.1.

which is $\underline{M}_{j+1} + delta + 1$, then $\underline{M}_j$ exceeds $\underline{M}_{j+1} + delta$. Therefore $\underline{M}_j$ is again updated to $\underline{M}_{j+1} + delta + 1$ and so remains correct. Again, $\underline{M}_j$ will have to be the correct value of M'_j for c'.

Thus in all cases, $\underline{M}_j$ will be updated correctly as long as $\underline{M}_{j+1}$ is correct when j is the active index. Finally, when the active index j is $r-1$, $\underline{M}_{j+1} = \underline{M}_r$ will always have been set to its correct value and when the active j moves left, $\underline{M}_{j+1}$ will always be correct too. This completes the proof. □

4. THE MULTI-SET PERMUTATION IMPLEMENTATION AND GENERATION OF ALL TREES

The initial degree sequence will be sorted by degree, with smallest degrees at the left. Once all trees (i.e. all c sequences) corresponding to the current degree sequence have been generated, generating the rest requires that: (1) the next permutation of the current degree sequence be generated in constant time; and (2) any resetting needed be done in constant time. All c sequences for the new degree sequence can then be generated. When all the degree sequences have been generated, along with their corresponding c sequences, all d and c sequences will have been generated for a_i, $i = 0, h$. This entire process is carried out by Algorithm 4.1. The data entered is i and a_i for each $a_i > 0$.

In [3] a multi-set permutation is represented as a linked list with `perm` pointing to its first item. For our purposes, `perm` represents a current degree sequence d and so it consists of m items. Each item of the `perm` list corresponds to a node of the tree and contains its `degree` as a member, a `left(right)` member that points to its left(right) neighbor, as well as other members needed to generate the next permutation in constant time as described in [3]. We also add `rend` to point to `perm`'s last item. This allows minor changes to the code of [3] but the algorithm used here is the same, except for modifications required to deal with nodes of degree one. The algorithm uses two other lists containing the items of `perm` that can be shifted and those that are finished. To avoid confusion, we use the names `ready` and `f`, respectively, for these lists here.

The algorithm of [3] for generating the next permutation of the current degree sequence is shown in Algorithm 4.2.

All the data structures used in the implementation of this algorithm are detailed in [3]. Here, only the modifications required for its application to generating all c sequences are considered. The first modification, other than adding `rend`, is the addition of three members to each item of `perm`: `first`, `last` and `howmany`. Whenever there is a series of adjacent items corresponding to nodes of degree one, then `last` of the leftmost item of the series will point to the rightmost item of the series. Similarly, `first` of the rightmost item of the series will point to the leftmost item of the series and `howmany` of the rightmost item of the series will contain the number of items in the series.

The information in `first`, `last` and `howmany` is kept current by modifications to the `shift` algorithm which we do not include, as they are straightforward.

In order to generate all other c for d and to reset c, it is necessary to access the degree in an item of `perm`, as well as the information in `first`, `last` and `howmany` of an item. To do this, pointers to the items are kept in an array, `jp`, so `jp[q[j]]` contains a pointer to node `q[j]`'s item. Arrays are also used for the sequences c, q, t and $\underline{M}$. However, `t[q[j]]` contains the number of the first chain of `q[j]` and `M[q[j]]` contains $\underline{M}_j$.

The `initialize` algorithm first creates the initial `perm`, `ready`, `f` and `rend` and other data structures that are needed, including `first` for the leftmost item of a series of degree one nodes and `last` and `howmany` for the rightmost item of the series. It also sets `Last` to `rend`'s degree, the degree of the last node, determines m and `r`, and initializes `c` to all `0`'s.

It would be possible to initialize all the `q[j]`'s, `t[q[j]]`'s and `M[q[j]]`'s for the initial degree sequence, but after the next permutation is generated it may not be possible to reset them *all* in constant time. Consequently, only those needed `initially` are then set by initialize, which sets `q[r]` to `m`, `q[r-1]` to `m-1`, `M[m]` to `0`, `M[m-1]` to `-1`, `jp[m]` to `rend` (so it points to the last node's item), `jp[m-1]` to the item to the left of `rend` (so it points to the next to the last node's item), the active index `j` to `r-1` and `sum` to 0. The `-1` in `M[m-1]` is used as a flag to indicate that `M[m-1]` has not yet been initialized.

As the algorithm executes, the active index `j` moves left from `r-1`, often coming back to `r-1`, but eventually working its way to `0`, when it will be done. *The first time the active index* `j` *belongs to node* `q[j]`, `j` *will be the last chain of node* `q[j]` and `q[j]` and `jp[q[j]]` will have been set correctly, but neither `M[q[j]]` nor `t[q[j]]` will have been set. In fact all entries of `q[k]`, `jp[q[k]]`, `M[q[k+1]]` and `t[q[k+1]]` will have been set for $j \leq k$.

In order to be ready for when the active index is `predj`, the last node of `q[j]`'s predecessor node in `q`, `predj`'s `q` and `jp` entries must be set. Also, `M[q[j]]` must be flagged as not yet initialized and `t[q[j]]` must be set.

If `q[j]` is not 1 then it has a predecessor node in `q` and `predj` is `j+1-jp[q[j]]`'s degree. If `q[j]` is not at the left end of `perm` (is not the root of the tree) and `q[j]`'s left neighbor in `perm` has degree one, then the predecessor of `q[j]` in `q` is the node that is the left neighbor of the

next_perm

```
set p to ready
if (p would be blocked after shifting)
   remove it from ready
if (p has an adjacent multiple)
   add its upward neighbor to ready
if (succ[p->degree] is not NULL)
   transfer items with degree larger than p->degree
   from f to ready
if (p is finished)
   if (p is direction done)
      add p to f, set succ[pred[p->degree] to p
      and set start[p->degree] to True
   change p's direction
shift the item p points to
```

ALGORITHM 4.2.

all_trees

```
output perm and c
while (j>0)
   set sum to sum+1-c_{j+1}, c_j to c_j+1
   if (M[q[j]] was not initialized)
      set M[q[j]] to m-q[j]
      if (q[j]is not node 1)
         set predj to j+1+jp[q[j]]
         if((jp[q[j]]isn't perm)and(jp[q[j]]->degree is 1))
            set q[predj] to q[j]-1-jp[q[j]]->lt->howmany
            set jp[q[predj]]to p[q[j]]->lt->first->lt
         else
            set q[predj]to q[j]-1
            set jp[q[predj]] to jp[q[j]]->lt
         set M[q[predj]] to -1 and t[q[j]] to predj+1
      set delta to q[j+1]-q[j]-1
      if(M[q[j]]>M[q[j+1]]+delta) set M[q[j]] to M[q[j+1]]+delta+1
      set M[q[j+1]] to M[q[j]] and c[j+1] to 0
      if (sum is M[q[j]])
            if (t[q[j]]<j) set q[j-1] to q[j]
            set j to j-1
      else
            set M[q[m]] to sum and j to r-1
      output perm and c
```

ALGORITHM 4.3.

first node in the series of 1's whose last node is `q[j]`'s left neighbor in `perm`. Otherwise, the predecessor of `q[j]` in `q` is `q[j]-1`, the node that is the left neighbor of `q[j]` in `perm`. `M[q[j]]` is flagged by setting it to `-1` and `t[q[j]]` is set to `predj` $+1$.

Finally, we need to set the `q[k]` for `t[q[j]]` $<$ `k` $<$ (`t[q[j]]` $+$ the degree of node `q[j]`), i.e. the `q[k]` for all chains `k` of `q[j]` but the first, which was `t[q[j]]`. This is done whenever the active `j` is about to move left (i.e. when `sum` is `M[q[j]]`) as follows: `if t[q[j]]` $<$ `j` set `q[j-1]` to `q[j]`.

Algorithm 4.3 outputs the initial `perm` and initial `c` and then generates and outputs all other `c`'s for that perm. It also updates `M[q[j]]` and `M[q[j+1]]` as described in Section 3.

After `next_perm` has generated the next permutation of the current degree sequence, `perm` and `rend` must be set again. Also, `Last`, `r`, `q[r]`, `q[r-1]`, `M[q[r]]`, `M[q[r-1]]`, `jp[m]`, `jp[m-1]` and the active `j` must be set again, much as `initialize` set them earlier. At this time `c` is all `0`'s except for `c[1]`, which must be set to `0`, see Algorithm 4.4.

```
reset

set r to r + (Last-rend->degree) and Last to rend->degree
set M[m] to 0, jp[m] to rend, q[r] to m, sum to 0 and c[1] to 0
set j to r-1
if (rend->degree is 1)
   set q[r-1] to q[r]-(rend->howmany)
   set jp[q[r-1]] to rend->first->lt
else if (rend->left->degree is 1)
   set q[r-1] to q[r]-1-rend->lt->howmany
   set jp[q[r-1]] to rend->lt->first->lt
else
   set q[r-1] to q[r]-1
   set jp[q[r-1]] to rend->lt
set M[q[r]] to 0 and M[q[r-1]] to -1
```

ALGORITHM 4.4.

```
enter the data
initialize d and c
all_trees
while (ready != NULL)
   next_perm
   reset
   all_trees
```

ALGORITHM 4.5.

Finally, Algorithm 4.5 is a refinement of Algorithm 4.1.

The complete program for Algorithm 4.5 is available at http://www.cis.temple.edu/~lafollet/specified_d_c_trees.html

ACKNOWLEDGEMENTS

We are deeply appreciative of the detailed comments and suggestions of the reviewers, and feel that they have enabled us to substantially improve the paper.

REFERENCES

[1] Ehrlich, G. (1973) Loopless algorithms for generating permutations, combinations, and other combinatorial objects. *J. ACM*, **20**, 500–513.

[2] Dershowitz, N. (1975) A simplified loop-free algorithm for generating permutations. *BIT*, **15**, 158–164.

[3] Korsh, J. F. and Lipschutz, S. (1997) Generating multiset permutations in constant time. *J. Algorithms*, **25**, 321–335.

[4] Chase, P. J. (1989) Combination generation and Graylex ordering. *Congressum Numerantium*, **69**, 215–242.

[5] Ehrlich, G. (1973) Algorithm 466: Four combinatorial algorithms. *Commun. ACM*, **16**, 690–691.

[6] Bitner, J. R., Ehrlich, G. and Reingold, E. M. (1976) Efficient generation of the binary reflected Gray code and its applications. *Commun. ACM*, **19**, 517–521.

[7] Fenner, T. I. and Loizou, G. (1980) A binary tree representation and related algorithms for generating integer partitions. *Comp. J.*, **23**, 332–337.

[8] Nijenhuis, A. and Wilf, H. S. (1978) *Combinatorial Algorithms* (2nd edn). Academic Press, New York.

[9] Korsh, J. F. (1994) Loopless generation of k-ary tree sequences. *Inform. Process. Lett.*, **52**, 243–247.

[10] Korsh, J. F. and Lipschutz, S. (1998) Shifts and loopless generation of k-ary trees. *Inform. Process. Lett.*, **65**, 235–240.

[11] Roelants van Baronaigien, D. (1991) A loopless algorithm for generating binary tree sequences. *Inform. Process. Lett.*, **39**, 189–194.

[12] Lucas, J. M., Roelants van Baronaigien D. and Ruskey, F. (1993) On rotations and the generation of binary trees. *J. Algorithms*, **15**, 1–24.

[13] Roelants van Baronaigien, D. (1997) *Loopfree Generation of k-ary Trees*. Technical Report DCS-254-IR, Department of Computer Science, University of Victoria.

[14] Canfield, E. R. and Williamson, S. G. (1995) A loop-free algorithm for generating the linear extensions of a poset. *Order*, **12**, 57–75.

[15] Vajnovszki, V. (1996) *A Loopless Algorithm for Generating k-ary Reflected Gray Codes*. Research Report, Department IEM, University of Burgundy.

[16] Savage, C. D. (1997) A survey of combinatorial Gray codes. *SIAM Rev.*, **39**, 605–629.

[17] Raney, G. N. (1960) Functional composition patterns and power series reversion. *Trans. Amer. Math. Soc.*, **94**, 441–451.

[18] Ruskey, F. and Roelants van Baronaigien, D. (1984) Fast recursive algorithms for generating combinatorial objects. *Congressus Numerantium*, **41**, 53–62.

[19] Zaks, S. and Richards, D. (1979) Generating trees and other combinatorial objects lexicographically. *SIAM J. Comput.*, **8**, 73–82.

[20] Korsh, J. F. and LaFollette, P. (2000) Multi-set permutations and loopless generation of ordered trees with specified degree sequence. *J. Algorithms*, **34**, 309–336.

[21] Korsh, J. F. and LaFollette, P. (1999) Towers, beads, and loopless generation of trees with specified degree. *Congressus Numerantium*, **139**, 157–166.