A Simple Algorithm for Generating Non-regular Trees in Lexicographic Order

M. C. ER

Department of Computer Science, The University of Western Australia, Nedlands, WA 6009, Australia

A one-to-one correspondence between a set of non-regular trees that have n_i internal nodes each with k_i sons, for $1 \leqslant i \leqslant t$, and $(m+1)$ leaves and a set of feasible codewords that have n_i occurrences of k_i, for $1 \leqslant i \leqslant t$, and m occurrences of 0 is proved to be isotone, where

$$m = \sum_{i=1}^{t} (k_i - 1) n_i.$$

A simple and efficient algorithm for generating a set of non-regular trees in lexicographic order is presented.

Received April 1986, revised July 1986

1. INTRODUCTION

In this paper we are concerned with the generation of rooted, ordered and non-regular trees in a lexicographic order. A tree is said to be *rooted* if there is an internal node which is used as the root. A tree is said to be *ordered* if the sons of each internal node may be distinguished as the first son, the second son and so on. The number of sons a node has is known as the *degree* of that node. A tree is said to be *non-regular* if every internal node of the tree may not have the same number of sons.

The problem of generating rooted, ordered and regular trees has been extensively studied in recent literature.[2, 3] However, the generation of rooted, ordered and non-regular trees has received less attention in the literature. Chorneyko and Mohanty[1] reported one of the earliest attempts in encoding non-regular trees as strings of digits using the breadth-first search method. More recently, Zaks and Richards[4] presented another method for encoding non-regular trees as strings of digits using the depth-first search approach. However, their algorithm for generating non-regular trees is unnecessarily clumsy and computationally expensive.

It turns out that the efficient method[2, 3] for generating regular trees can be extended to the case of non-regular trees. In what follows, we shall derive such an efficient algorithm. This algorithm, of course, runs faster than Zaks and Richards' corresponding algorithm.[4]

2. PRELIMINARIES

Let $K = (k_1, k_2, \ldots, k_t)$ be a t-tuple of non-negative integers, such that $k_1 < k_2 < \ldots < k_t$. Further, let $N = (n_1, n_2, \ldots, n_t)$ be another t-tuple of non-negative integers, such that

$$m = \sum_{i=1}^{t} (k_i - 1) n_i. \tag{1}$$

Note that n_i may be regarded as the frequency of k_i, for $1 \leqslant i \leqslant t$. Let $T(K, N)$ denote a set of non-regular trees, such that each tree has n_i internal nodes each with k_i sons, for $1 \leqslant i \leqslant t$, and $(m+1)$ leaves. An example of $T \in T(K, N)$, where $K = (2, 3, 5)$ and $N = (3, 1, 1)$ is shown in Fig. 1.

For convenience, let T_i denote the ith son of $T \in T(K, N)$, and degree (T) be the degree of the root of

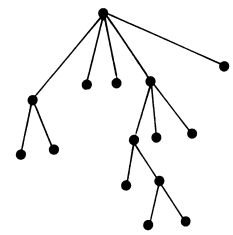

Figure 1. A non-regular tree $t \in T(K, N)$, where $K = (2, 3, 5)$ and $N = (3, 1, 1)$, with codeword $C = 52000032020000$.

T. Let $T, T' \in T(K, N)$. Then we may impose an ordering among them.

Definition 1 (*lexicographic order of non-regular trees*)

We say that $T < T'$ if

(*a*) degree (T) < degree (T'), or

(*b*) degree (T) = degree (T'), and for some i, $1 \leqslant i \leqslant$ degree (T),

(i) $T_j = T'_j$, for $1 \leqslant j < i$, and

(ii) $T_i < T'_i$. □

This lexicographic ordering, of course, is a generalisation of the local ordering used in the case of binary trees.[2, 3] As such, it is easy to construct an efficient algorithm for generating $T(K, N)$ in lexicographic order.

To generate all non-regular trees of $T(K, N)$, it is easier to manipulate linearised representations of non-regular trees rather than the actual trees themselves. Let $C = c_1 c_2 \ldots c_n$ be a codeword, such that it has n_i occurrences of k_i, for $1 \leqslant i \leqslant t$, and m occurrences of 0, where

$$n = m + \sum_{i=1}^{t} n_i. \tag{2}$$

A codeword C is said to possess the *dominating property* if the number of zeros is not greater than

$$\sum_{c_i \neq 0} (c_i - 1)$$

while scanning from c_1 to c_n. A codeword C is said to be *feasible* if it possesses the dominating property and the number of zeros satisfies Equation (1). For example, the codeword of the non-regular tree shown in Fig. 1 is $C = 52000032020000$.

Let $C(K, N)$ be a set of feasible codewords. Let $C = c_1 c_2 \ldots c_n$ and $C' = c'_1 c'_2 \ldots c'_n$ be two feasible codewords. Then we may also impose an ordering among them.

Definition 2 (Lexicographic order of codewords)

We say that $C < C'$ if there exists a i, $1 \leqslant i \leqslant n$, such that
(*a*) $c_j = c'_j$, for $1 \leqslant j < i$; and
(*b*) $c_i < c'_i$. □

Now we prove the following two theorems, which are essential to the subsequent derivation of a generating algorithm.

Theorem 1

The mapping between $T(K, N)$ and $C(K, N)$ is one-to-one.

Proof

Let $T \in T(K, N)$. If T is traversed in pre-order, such that the degree of each node visited is recorded, the sequence of degrees so recorded forms a codeword C of non-negative integers, with the last digit omitted (which is a zero, as the last node visited by pre-order traversal is a leaf). As a property of pre-order traversal, an internal node is visited prior to its sons; therefore its degree comes before the degrees of its sons in C. It it generally true that the number of leaves an internal node has cannot be greater than its degree. Thus the codeword for such a simple tree is feasible. If another simple tree is attached to a simple tree, the codeword for such a resulting tree is also feasible as a zero (a leaf) is replaced by k occurrences of 0 and one occurrence of k, where k is the degree of the attaching simple tree. By induction, C is feasible and therefore is a member of $C(K, N)$.

Let $T, T' \in T(K, N)$, and $C, C' \in C(K, N)$, such that T and T' map to C and C', respectively. If $C = C'$, then $T = T'$ as pre-order traversal visits each node in a deterministic and pre-defined manner. Hence the mapping from $T(K, N)$ to $C(K, N)$ is one to one.

The converse is also easy to prove. □

Theorem 2

The lexicographic ordering of non-regular trees is preserved in the lexicographic ordering of codewords. In other words, the mapping between $T(K, N)$ and $C(K, N)$ is isotone.

Proof

Let $T, T' \in T(K, N)$, such that they both map to $C, C' \in C(K, N)$, respectively. Suppose $T < T'$. Then either (*a*) degree (T) < degree (T'), or (*b*) there exists an i, $1 \leqslant i \leqslant$ degree (T), such that $T_i < T'_i$ and $T_j = T'_j$, for $1 \leqslant j < i$. Corresponding to case (*a*), we have $c_1 < c'_1$. Corresponding to case (*b*), there exists an x such that $c_x < c'_x$ and $c_y = c'_y$, for $1 \leqslant y < x$, where x is the xth nodes in T and T' visited by pre-order traversal such that they are the first nodes that have different degrees. Hence $C < C'$. □

```
procedure Generate(f, z, p: integer);
var i: integer;
begin
  if (f = 0) and (z = 0) then PrintCodeword
  else begin
       if z > 0 then begin
                     c[p] := 0;
                     Generate(f, z-1, p+1)
                     end;
       for i: = 1 to t do
                if n[i] > 0 then begin
                                 c[p] := k[i];
                                 n[i] := n[i]-1;
                                 Generate(f-1, z+k[i]-1,
                                     p+1);
                                 n[i] := n[i]+1
                                 end
  end
end {Generate};
```

Figure 2. An algorithm for generating $C(K, N)$ in lexicographic order.

```
20204000
20240000
20400020
20400200
20402000
20420000
22004000
22040000
22400000
24000020
24000200
24002000
24020000
24200000
40002020
40002200
40020020
40020200
40022000
40200020
40200200
40202000
40220000
42000020
42000200
42002000
42020000
42200000
```

Figure 3. A lexicographic listing of $C(K, N)$ generated by the algorithm shown in Fig. 2, where $K = (2, 4)$ and $N = (2, 1)$.

3. SIMPLE ALGORITHM

From Theorem 2, we see that, to generate $T(K, N)$ in lexicographic order, we need only to enumerate $C(K, N)$ in lexicographic order. Since a codeword can be represented by a one-dimensional array and is easier to manipulate than a tree, generation of $C(K, N)$ is preferred.

To generate all feasible codewords of $C(K, N)$, it is necessary to generate all possible combinations of k_i with the appropriate number of occurrences given by n_i, for $1 \leqslant i \leqslant t$, such that the resulting codewords are feasible. To ensure that the dominating property is satisfied during the process of forming a codeword, two parameters are needed – one controls the number of all remaining k_is

```
function Convert(var i: integer): treeptr;
var j: integer;
  T: treeptr;
begin
  i:= i+1;
  if (c[i] = 0) or (i > f+m) then Convert:= nil
  else begin
    new(T);
    T^.degree:= c[i];
    for j:= 1 to c[i] do T^.son[j]:= Convert(i);
    Convert:= T
    end
end {Convert};
```

Figure 4. An algorithm for converting $C \in C(K, N)$ to $T \in T(K, N)$. It is activated as *Convert(j)*, where $j = 0$.

that can be attached to a head; another one controls the number of zeros that can be added to a head at that point. Both parameters need to be adjusted accordingly as soon as a non-negative integer is appended to a head. An algorithm for generating $C(K, N)$ in lexicographic order is shown in Fig. 2. This algorithm is activated as *Generate* $(f, 0, 1)$, where

$$f = \sum_{i=1}^{t} n_i.$$

The first parameter f of *Generate* controls the number of k_is yet to be used in a codeword; the second parameter z controls the number of zeros that can be added to a codeword such that the resulting codeword still satisfies

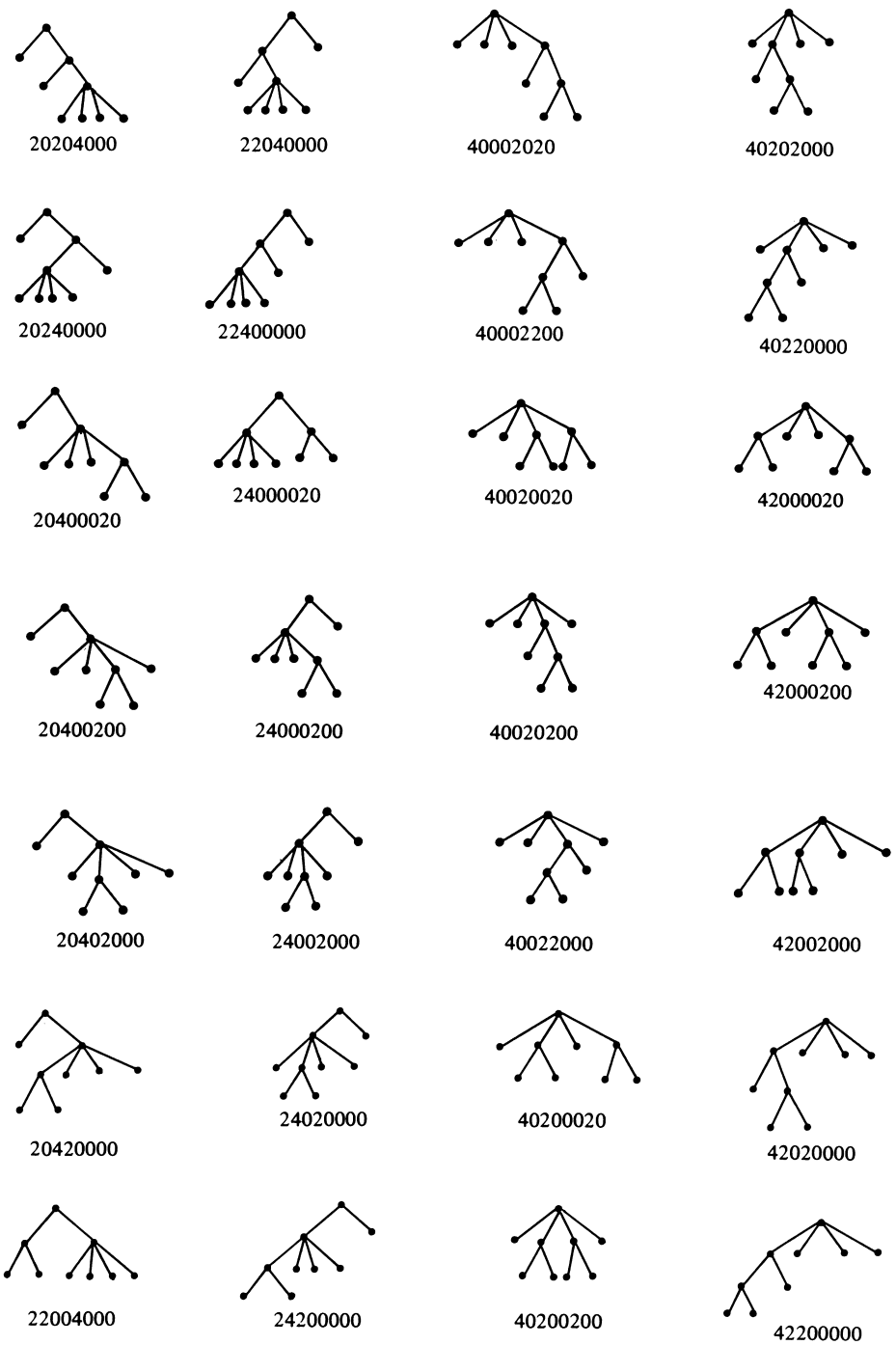

Figure 5. A listing of $T(K, N)$ converted from $C(K, N)$, where $K = (2, 4)$ and $N = (2, 1)$.

the dominating property; the third parameter p points to the next position to be filled. Thus whenever k_i is used in a codeword, f is decremented by 1, and z is incremented by $(k_i - 1)$. Furthermore, whenever a zero is used in a codeword, z is decremented by 1. The combination of f and z ensures that the number of zeros inserted into a codeword is not more than

$$\sum_{c_i \neq 0} (c_i - 1)$$

at any moment. Hence the dominating property is preserved.

Finally, the algorithm *Generate* always tries to assign a zero to the next position first before assigning a k_i, for $i = 1, 2, \ldots, t$, provided $n_i \neq 0$. Therefore the list of codewords so generated is in lexicographic order. A lexicographic listing of $C(K, N)$ generated by the algorithm is shown in Fig. 3, where $K = (2, 4)$ and $N = (2, 1)$.

4. CONCLUDING REMARKS

A codeword $C \in C(K, N)$ may be regarded as a linearised representation of the corresponding non-regular tree $T \in T(k, N)$. In fact, C can be converted to T easily by a conversion algorithm as shown in Fig. 4. A listing of $T(K, N)$ converted from $C(K, N)$, shown in Fig. 3, by this algorithm is illustrated in Fig. 5.

A comparison of our algorithm *Generate* with Zaks and Richards' generating algorithm[4] reveals that ours is very much simpler and shorter. This is, no doubt, due to the explicit use of two parameters to control the dominating property so that the generated codewords are always feasible. An empirical test also reveals that *Generate* consistently runs faster than Zaks and Richards' generating algorithm. Thus our algorithm is preferred.

REFERENCES

1. I. Z. Chorneyko and S. G. Mohanty, On the enumeration of certain sets of planted plane trees. *Journal of Combinatorial Theory* (*B*) **18**, 209–221 (1975).
2. M. C. Er, A note on generating well-formed parenthesis strings lexicographically. *The Computer Journal* **26** (3), 205–207 (1983).
3. M. C. Er, Enumerating ordered trees lexicographically. *The Computer Journal* **28** (5), 538–542 (1985).
4. S. Zaks and D. Richards, Generating trees and other combinatorial objects lexicographically. *SIAM Journal on Computing* **8**, 73–81 (1979).