

# Lexicographic Listing and Ranking of $t$ -ary Trees

M. C. ER\*

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

*This paper presents three simple and efficient algorithms for generating, ranking and unranking  $t$ -ary trees in a lexicographic order. The simplest idea of encoding a  $t$ -ary tree with  $n$  nodes as a bit-string of length  $tn$  is exploited to its full advantages. It is proved that the lexicographic order in the set of  $t$ -ary trees with  $n$  nodes is preserved in the set of bit-strings of length  $tn$ , using the above encoding scheme. Thus by generating all bit-strings in the lexicographic order, a simple decoding algorithm can convert them to  $t$ -ary trees in the same order. Finally, the theoretical basis for ranking a lexicographic listing of bit-strings is discussed, and the ranking and the unranking algorithms are derived.*

Received December 1985, revised September 1986

## 1. INTRODUCTION

In recent years the interest in the lexicographic generation of regular trees has been shifted from binary trees to  $t$ -ary trees. Ruskey<sup>1</sup> presented one of the earliest algorithms for generating  $t$ -ary trees in a lexicographic order. He encoded a  $t$ -ary tree as a string of digits, each digit representing the level of a leaf. Thus by manipulating a string of leaf levels, all  $t$ -ary trees could be generated. Trojanowski<sup>2</sup> established a one-to-one mapping between a set of  $t$ -ary trees and a set of stack-generatable permutations. Thus by enumerating a set of stack-generatable permutations, all  $t$ -ary trees could also be generated. Liu<sup>3</sup> used a permutation of the multiset  $\{1, 2, \dots, n\}^{t-1}$  to encode a  $t$ -ary tree with  $n$  nodes. Thus the problem of generating all  $t$ -ary trees is reduced to the problem of generating all permutations of the multiset subject to certain constraints. It is well known that a representation has tremendous influence on the complexity and the efficiency of an algorithm that manipulates it. Due to the utilisation of these complex encoding schemes for representing  $t$ -ary trees, their algorithms for generating  $t$ -ary trees are unnecessarily complex; so are their ranking and unranking algorithms.

In a recent paper, Er<sup>4</sup> proposed a method for encoding a binary tree as a binary string, and demonstrated that simple and efficient algorithms for generating, ranking and unranking binary trees could be constructed. In this paper we extend the results to  $t$ -ary trees. More specifically, we shall show that a  $t$ -ary tree can be encoded as a binary string, and that simple and efficient algorithms for generating, ranking and unranking  $t$ -ary trees can be constructed.

## 2. DEFINITIONS AND NOTATIONS

Throughout this paper, we consider only the class of rooted, ordered and regular  $t$ -ary trees. A tree is said to be *rooted* if it has a single root; it is said to be *regular* if every internal node has  $t$  children; and *ordered* if the subtrees of each internal node can be identified. When we say a  $t$ -ary tree, we mean a rooted, ordered and regular  $t$ -ary tree.

Let  $T(t, n)$  be a set of  $t$ -ary trees with  $n$  nodes. Further, let  $T_i$  denote the  $i$ th subtree of  $T \in T(t, n)$ . To generate members of  $T(t, n)$  systematically, first of all it is necessary to specify the lexicographic order.

\* Now at: Dept. of Maths. and Computing Sciences, St. Francis Xavier University, Antigonish, Nova Scotia, Canada B2G 1C0.

### Definition of lexicographic order

Given two  $t$ -ary trees,  $T$  and  $T'$ , we say that  $T < T'$  if

- (a)  $T$  is empty and  $T'$  is not empty, or
- (b) both  $T$  and  $T'$  are not empty, and for some  $1 \leq i \leq t$ :
  - (i)  $T_j = T'_j$ , for  $j = 1, 2, \dots, i-1$ , and
  - (ii)  $T_i < T'_i$ . □

Let  $|T(t, n)|$  denote the number of distinct  $t$ -ary trees with  $n$  nodes. Further, let  $C_{t, n}$  be the generalized Catalan number.<sup>5</sup> Then the following lemma is immediate.

### Lemma 1

$$|T(t, n)| = C_{t, n} = \frac{1}{(t-1)n+1} \binom{tn}{n}$$

for  $t \geq 2$  and  $n \geq 0$ . □

A  $t$ -ary tree  $T$  with  $n$  nodes can be converted to an *extended*  $t$ -ary tree  $\bar{T}$  with  $n$  internal nodes (see Ref. 6 for an extended binary tree), which is also a *full*  $t$ -ary tree, by replacing all empty nodes of  $T$  by  $n(t-1)+1$  leaves. Let  $\bar{T}(t, n)$  be a set of extended  $t$ -ary trees with  $n$  internal nodes. It is obvious that  $T(t, n)$  and  $\bar{T}(t, n)$  are in one-to-one correspondence.

An extended  $t$ -ary tree  $\bar{T}$  can be encoded as a binary bit-string as follows. An internal node and a leaf are represented as a one and a zero, respectively, by the pre-order traversal of  $\bar{T}$ . The resulting bit-string comprises  $(tn+1)$  bits. Because of the nature of pre-order traversal, the last bit of such a bit-string is always a zero. It can therefore be ignored without affecting the representation. The reason for deleting the last zero is to facilitate the following definitions.

Let  $B = (b_1 b_2 b_3 \dots b_{tn})$  be a bit-string.  $B$  is said to have the  *$t$ -dominating property* if the number of zeros is always less than or equal to  $t$  times the number of ones while scanning from  $b_1$  to  $b_{tn}$ . Furthermore,  $B$  is said to be  *$t$ -feasible* if the number of zeros is equal to  $t$  times the number of ones, and  $B$  satisfies the  *$t$ -dominating property*. Finally, let  $B(t, n)$  be a set of bit-strings of length  $tn$ , such that they are all  *$t$ -feasible*.

## 3. ENCODING AND DECODING ALGORITHMS

Since an extended  $t$ -ary tree can be encoded as a bit-string,  $\bar{T}(t, n)$  and  $B(t, n)$  are related in some ways. Furthermore,  $T(t, n)$  and  $\bar{T}(t, n)$  are in one-to-one correspondence;

hence  $T(t, n)$  and  $B(t, n)$  are also related. Thus if a node of  $T \in T(t, n)$  is encoded as a one and an empty subtree of  $T$  is encoded as a zero, then  $T$  can be encoded as a bit-string directly. The details are presented in the following encoding algorithm.

```

procedure Encode( $T$ : treeptr);
var  $j$ : integer;
begin
   $i := i + 1$ ;
  if  $T = \text{nil}$  then
     $B[i] := '0'$ 
  else begin
     $B[i] := '1'$ ;
    for  $j := 1$  to  $t$  do
      Encode( $T^{\wedge}.son[j]$ );
    end
  end {Encode};

```

Note that  $i$  is initialised to zero to start with. The zero in  $B[t*n + 1]$  may be omitted.

The above algorithm clearly suggests the following theorem.

#### Theorem 1

The encoding of  $T \in T(t, n)$  as a bit-string always yields a  $t$ -feasible bit-string  $B \in B(t, n)$ .

#### Proof

The encoded bit-string of  $T$  is a trace of the pre-order traversal of  $T$ . As a node is always visited first before its children, which may or may not be empty, the resulting bit-string  $B$  satisfies the  $t$ -dominating property and is always  $t$ -feasible if the last empty subtree visited is omitted.  $\square$

Conversely, a  $t$ -ary tree  $T$  can be constructed from a bit-string  $B$  directly. The detailed decoding algorithm is given below.

```

function Decode: treeptr;
var  $j$ : integer;
   $T$ : treeptr;
begin
   $i := i + 1$ ;
  if ( $B[i] = '0'$ ) or ( $i > t*n$ ) then
    Decode := nil
  else begin
    new( $T$ );
    for  $j := 1$  to  $t$  do
       $T^{\wedge}.son[j] := \text{Decode}$ ;
    Decode :=  $T$ ;
  end;
end {Decode};

```

Again,  $i$  is initialised to zero.

The above encoding and decoding algorithms clearly demonstrate the deep connection between  $T(t, n)$  and  $B(t, n)$ . The relationship between them can be formalized in the following theorem.

#### Theorem 2

The mapping between  $T(t, n)$  and  $B(t, n)$  is one to one.

#### Proof

The one-to-one correspondence is implied by the algorithms for encoding a  $T \in T(t, n)$  and for decoding a  $B \in B(t, n)$ . Let  $B, B' \in B(t, n)$  be the encoded  $T, T' \in T(t, n)$ , respectively. As the pre-order traversal of a  $t$ -ary tree is deterministic, each node is visited in a predetermined manner. Hence, if  $B = B'$ , then  $T = T'$ . The converse is also true.  $\square$

Notice that the encoding and the decoding algorithms described above visit/construct each node of  $T \in T(t, n)$  once and only once, therefore the running times of both algorithms are  $O(n)$ .

#### 4. LISTING ALGORITHM

In the previous section, the one-to-one correspondence between  $T(t, n)$  and  $B(t, n)$  is established. Thus, instead of generating all members of  $T(t, n)$  in the lexicographic order, we may simply enumerate all members of  $B(t, n)$  in the same order. A question that naturally arises is whether or not the same lexicographic order can be preserved in  $T(t, n)$  and  $B(t, n)$ , when the one-to-one correspondence between them is established. The result is summarised in the following important theorem.

#### Theorem 3

Let  $T, T' \in T(t, n)$  and  $B, B' \in B(t, n)$ . Suppose  $B$  and  $B'$  are the encoded bit-strings of  $T$  and  $T'$ , respectively. Then  $B < B'$  if and only if  $T < T'$ .

#### Proof

We prove this theorem in two steps.

(i)  $T < T'$  implies  $B < B'$ .

$T < T'$  implies that, by the pre-order traversal, the  $i$ th (say) node of  $T$  is empty and the  $i$ th node of  $T'$  is not empty, while nodes 1 to  $(i-1)$  are the same. By the encoding algorithm, the first  $(i-1)$  bits of  $B$  and  $B'$  are the same. As  $b_i$  of  $B$  is 0 and  $b'_i$  of  $B'$  is 1, thus  $B < B'$ .

(ii)  $B < B'$  implies  $T < T'$ .

The converse can also be proved readily.  $B < B'$  implies that there exists a  $j$ ,  $1 \leq j \leq tn$ , such that  $b_j < b'_j$  and  $b_i = b'_i$  for  $1 \leq i \leq j-1$ . As  $B$  and  $B'$  are binary bit-strings, it follows that  $b_j = 0$  and  $b'_j = 1$ . By the decoding algorithm, the  $j$ th nodes of  $T$  and  $T'$  are empty and non-empty, respectively, while nodes 1 to  $(j-1)$  are the same. Therefore,  $T < T'$ .  $\square$

This theorem explicitly states that the one-to-one mapping between  $T(t, n)$  and  $B(t, n)$  preserves the lexicographic order of  $T(t, n)$  in  $B(t, n)$ . Therefore we need only to generate  $B(t, n)$  in the lexicographic order; the conversion from bit-strings to  $t$ -ary trees can be trivially carried out by the decoding algorithm. The details of the generating algorithm for listing  $B(t, n)$  in the lexicographic order are given below.

```

procedure Listing( $a, z$ : integer);
var  $i$ : integer;
begin
  if ( $a = 0$ ) and ( $z = 0$ ) then PrintBitString
  else begin
     $i := t*(n-a) - z + 1$ ;
    if  $z < 0$  then begin
       $B[i] := '0'$ ;

```

```

Listing( $a, z-1$ );
end;
if  $a < > 0$  then begin
     $B[i] := '1'$ ;
    Listing( $a-1, z+t-1$ );
end
end
end {Listing};
    
```

If the above algorithm is activated as Listing( $n, 0$ ), then  $B(t, n)$  will be generated. This generating algorithm can be proved correct as follows. (i) We show that exactly  $n$  ones and  $n(t-1)$  zeros are generated per bit-string. Note that  $a$  is decremented by 1 when a one is generated. It follows that  $n$  ones will be generated per bit-string. Furthermore, the value of  $z$  is incremented by  $(t-1)$  when a one is subtracted from  $a$ . As  $z$  is decremented by one when a zero is generated, exactly  $n(t-1)$  zeros will be generated per bit-string – since the initial value of  $z$  is zero. (ii) We show that the bit-strings so generated are  $t$ -feasible. It is important to note that  $(t-1)$  zeros cannot be generated before a one is generated. More generally, additional  $(t-1)$  zeros cannot be generated before an additional one is generated. Consequently, the bit-strings so generated are  $t$ -feasible, as the initial values of  $a$  and  $z$  are  $n$  and 0, respectively. (iii) We show that the successive bit-strings so generated are in the lexicographic order. It is apparent from the algorithm that for any  $i$ ,  $1 \leq i \leq tn$ ,  $b_i$  is assigned a zero before a one whenever possible. Thus the successive bit-strings generated are in the lexicographic order. Hence the correctness of the algorithm is established.

## 5. RANKING ALGORITHM

A bit-string  $B \in B(t, n)$  is a string of binary digits; it can also be interpreted as a binary number. This makes the association between a  $t$ -ary tree and a number obvious. However, using this method the set of numbers that  $B(t, n)$  maps to is not consecutive. It is desirable to map  $B(t, n)$  to a set of consecutive natural numbers so that a list of  $t$ -ary trees with  $n$  nodes in the lexicographic order can be ranked accordingly.

Let  $R$  be a ranking function which maps  $B(t, n)$  to  $Z = \{1, 2, \dots, C_{t,n}\}$ , such that  $B < B'$  if and only if  $R(B) < R(B')$ , where  $B, B' \in B(t, n)$ . By Theorem 3,  $T(t, n)$  is also ranked in the same order. So the remaining problem is to derive  $R$ .

First of all, we derive some auxiliary functions to assist the final derivation  $R$ . Let  $f(i)$  be the number of ones in between  $b_i$  and  $b_{tn}$  of  $B$  inclusively. Further, let  $V(i)$  be the number of bit-strings that are  $t$ -feasible, such that they all have the same prefix  $b_1 b_2 \dots b_{i-1}$  as  $B$  has, with  $b_i = 0$ . Then  $V(i)$  is given by the following important theorem.

**Theorem 4**

$$V(i) = \binom{tn-i}{f(i)} - \binom{tn-i}{f(i)-1} (t-1).$$

**Proof**

$V(i)$  is equal to the number of permutations of  $f(i)$  ones and  $(tn-i-f(i))$  zeros in between  $b_{i+1}$  and  $b_{tn}$  inclusively,

such that the resulting bit-strings are  $t$ -feasible. The number of all possible permutations of  $f(i)$  ones and  $(tn-i-f(i))$  zeros is

$$\binom{tn-i}{f(i)}.$$

However, the above number of permutations also includes bit-strings that do not satisfy the  $t$ -dominating property. By the reflection principle,<sup>7</sup> the number of such bit-strings is

$$\binom{tn-i}{f(i)-1} (t-1).$$

Hence the theorem is correct.  $\square$

**Theorem 5**

$$R(B) = 1 + \sum_{b_i=1} \left( \binom{tn-i}{f(i)} - \binom{tn-i}{f(i)-1} (t-1) \right).$$

**Proof**

For each  $b_i$  of  $B$ , such that  $b_i = 1$ ,  $B$  is preceded by other bit-strings having the prefix  $b_1 b_2 \dots b_{i-1} 0$ , where  $b_1 b_2 \dots b_{i-1}$  are identical with the first  $(i-1)$  bits of  $B$ . By Theorem 4, the total number of bit-strings preceding  $B$  is

$$\sum_{b_i=1} \left( \binom{tn-i}{f(i)} - \binom{tn-i}{f(i)-1} (t-1) \right).$$

Hence the position index  $R(B)$  of  $B$  in the lexicographic listing of  $B(t, n)$  is

$$1 + \sum_{b_i=1} \left( \binom{tn-i}{f(i)} - \binom{tn-i}{f(i)-1} (t-1) \right). \quad \square$$

**Corollary 1**

$$V(1) = 0.$$

**Proof**

As  $f(1) = n$ , therefore

$$\begin{aligned} V(1) &= \binom{tn-1}{n} - \binom{tn-1}{n-1} (t-1) \\ &= 0. \end{aligned} \quad \square$$

Equipped with Theorem 5, the ranking algorithm can be implemented easily, and is detailed below.

**function Rank( $B$ : bitstring): integer;**

**var  $i, j$ , index: integer;**

**begin**

$j := n-1$ ;

$i := 2$ ;

$index := 1$ ;

**while  $j > 0$  do begin**

**if  $B[i] = '1'$  then begin**

$index := index + C(t*n-i, j-1)$

$* (t*(n-j) - i + 1) \text{ div } j$ ;

$j := j-1$ ;

**end;**

$i := i+1$ ;

**end;**

$Rank := index$ ;

**end {Rank};**

Note that the equation of Theorem 4 can be simplified as follows:

$$\binom{tn-i}{f(i)} - \binom{tn-i}{f(i)-1} (t-1) = \frac{t(n-f(i))-i+1}{f(i)} \binom{tn-i}{f(i)-1}.$$

Indeed, this simplified form is used in the ranking algorithm. Note that in the ranking algorithm,

$$C(t*n-i, j-1) = \binom{tn-i}{j-1},$$

and the initial values of  $i$ ,  $j$  and  $\text{index}$  are based on Theorem 5 and Corollary 1.

## 6. UNRANKING ALGORITHM

Let  $R^{-1}$  be the unranking function which maps  $Z = \{1, 2, \dots, C_{t,n}\}$  to  $B(t, n)$ . Clearly  $R^{-1}$  is the inverse of  $R$ . So, given a position index, we want to compute its corresponding bit-string in the lexicographic listing of  $B(t, n)$ . The following implementation does just that.

```

procedure Unrank(index: integer);
var  $i, j, x$ : integer;
begin
   $j := n$ ;
   $i := 1$ ;
  while  $j > 0$  do begin
     $x := C(t*n-i, j-1) * (t*(n-j)-i+1) \text{ div } j$ ;
    if  $\text{index} > x$  then begin
       $B[i] := '1'$ ;
       $j := j-1$ ;
    end
  end

```

## REFERENCES

1. F. Ruskey, Generating  $t$ -ary trees lexicographically. *SIAM Journal on Computing* 7, 424–439 (1978).
2. A. E. Trojanowski, Ranking and listing algorithms for  $k$ -ary trees. *SIAM Journal on Computing* 7, 492–509 (1978).
3. C. L. Liu, Generation of  $k$ -ary trees. *Proc. 5th Colloque de Lille—Les Arbres en Algèbre et en Programmation* (1980).
4. M. C. Er, Enumerating ordered trees lexicographically. *The Computer Journal* 28 (5), 538–542 (1985).
5. A. D. Sands, On generalised Catalan numbers. *Discrete Mathematics* 21, 219–221 (1978).
6. D. E. Knuth, *The Art of Computer Programming*, vol. 1. Addison-Wesley, Reading, Massachusetts (1973).
7. W. Feller, *An Introduction to Probability Theory and its Applications*, vol. 1, third edition. Wiley, New York 1968.

```

     $\text{index} := \text{index} - x$ ;
  end
  else  $B[i] := '0'$ ;
   $i := i+1$ ;
end;
for  $i := i$  to  $t*n$  do  $B[i] := '0'$ ;
end {Unrank};

```

The correctness of the unranking algorithm may be proved as follows. First of all, it compares the position index with  $V(i)$ . If  $\text{index} > V(i)$ ,  $b_i$  should be assigned a one in order that the given position index will fall within the range covered by the subset of  $B(t, n)$  having the same prefix  $b_1 b_2 \dots b_{i-1}$  in the lexicographic listing. The total number of ones to be assigned to  $B$  is reduced by 1, and the value of the position index is reduced by  $V(i)$ . Conversely, if  $\text{index} \leq V(i)$ ,  $b_i$  should be assigned a zero, as  $V(i)$  indicates the number of  $t$ -feasible bit-strings having the prefix  $b_1 b_2 \dots b_{i-1} 0$ . By induction, the unranking algorithm is correct.

## 7. CONCLUDING REMARKS

Representation plays an important role in complex problem solving, as well as in designing efficient algorithms. Some representations lend a hand to simple and transparent solutions, but some do not. This paper shows that the simplest idea of representing a binary tree as a bit-string<sup>4</sup> can be extended to a  $t$ -ary tree. The resulting generating, ranking and unranking algorithms for  $t$ -ary trees are also very simple and efficient.