IEICE TRANS. INF. & SYST., VOL.E94–D, NO.2 FEBRUARY 2011

PAPER Special Section on Foundations of Computer Science — Mathematical Foundations and Applications of Algorithms and Computer Science —

Ranking and Unranking of t-Ary Trees Using RD-Sequences

Ro-Yu WU[†a)], ***Member***, **Jou-Ming CHANG**[††], ***and*** **Yue-Li WANG**[†††], ***Nonmembers***

SUMMARY In this paper, we introduce a concise representation, called right-distance sequences (or RD-sequences for short), to describe all t-ary trees with n internal nodes. A result reveals that there exists a close relationship between the representation and the well-formed sequences suggested by Zaks [Lexicographic generation of ordered trees, *Theoretical Computer Science* 10 (1980) 63–82]. Using a coding tree and its concomitant tables, a systematical way can help us to investigate the structural representation of t-ary trees. Consequently, we develop efficient algorithms for determining the rank of a given t-ary tree in lexicographic order (i.e., a ranking algorithm), and for converting a positive integer to its corresponding RD-sequence (i.e., an unranking algorithm). Both the ranking and unranking algorithms can be run in $O(tn)$ time and without computing all the entries of the coefficient table.

key words: *t-ary trees, ranking, unranking, lexicographic order, coding trees*

1. Introduction

Many algorithms have been developed for generating t-ary trees (as well as binary trees) with n internal nodes. In most of these algorithms, t-ary trees are encoded to integer sequences and all different sequences are generated in a particular order, especially in lexicographic order [1], [4], [12], [13], [15], [18], [21], [22] or Gray-code order [2], [6], [11], [19]. Besides, a few algorithms employ the usual pointer in computer representation to generate t-ary trees [5], [7], [8], [20]. Accordingly, we customarily say t-ary trees to mean their representations.

Given a specific order on the set of t-ary trees with n internal nodes, a *ranking algorithm* is a function that determines the position (rank) of a given t-ary tree in that order, and an *unranking algorithm* is one that finds the tree of a given position. To date the best known ranking algorithms [1], [12], [13], [15], [22] and unranking algorithms [1], [13] for diverse representations of t-ary trees require $O(tn)$ time. However, all of them are demanded to build a coefficient table of a size approximating to $(n(t-1)+1)\cdot n$ in advance before ranking or unranking. For this reason, the pre-computation of those coefficient entries takes additional cost in $O(tn^2)$ time. A challenge arisen from this research is to ask whether the pre-computation for building coefficient table is necessary in order to compute the ranking and/or unranking function. Lucas et al. claimed that, in fact, they have been unable to reduce the running time of the ranking algorithm or unranking algorithm to less than $O(n^2)$ time even if they consider for binary trees (see p. 354 in [8]).

In this paper, we deal with the problem of ranking and unranking for t-ary trees with n internal nodes in lexicographic order. We will show that both the ranking and unranking problems for t-ary trees can be solved in $O(tn)$ time without computing all the entries of the coefficient table. An observation shows that only the entries located in a path from the lower-left corner to the upper-right corner of the table are needed to compute. Thus, we establish four formulae that can be used for computing the relative right, left, upper, and lower entries of a given element in the table, respectively. Because each formula can be computed in a constant time and the length of a required path is no more than tn, all subsequent entries can be determined quickly if an initial term is computed. Consequently, we obtain an improvement on the time and space complexities for both the ranking and unranking algorithms.

2. Structural Representations of t-Ary Trees

We begin with a structure representation of t-ary trees. For $t \geqslant 2$, a *t-ary tree* T is an ordered tree defined recursively as follows: either T is empty or it has a distinguished node r called its *root* that is connected to subtrees $T_1, T_2, \ldots, T_t$, each of which is a t-ary tree. Let $\mathscr{T}_{n,t}$ denote the set of t-ary trees with n internal nodes. If $T \in \mathscr{T}_{n,t}$, then we assume that all internal nodes of T are numbered from 1 to n in preorder list (i.e. visit the root and then recursively the subtrees of T from left to right). For each internal node $i \in T$, we define *right distance* d_i as follows. First, $d_i = 0$ for each node i on the right arm of T (i.e., the path from the root to its rightmost descendant). For all other nodes, $d_i = d_{p(i)} + t - k$ $(1 \leqslant k \leqslant t)$, where $p(i)$ stands for the parent of node i and node i is the kth son (from left to right) of node $p(i)$. Thus, $d_i = d_{p(i)}$ when node i is the rightmost child of its parent. Note that $d_i \leqslant d_{i-1} + (t-1)$. Denote that resulting sequence by $rd(T) = (d_1, d_2, \ldots, d_n)$ as the *right distance sequence* (RD-sequence for short) of T.

The concept of RD-sequences comes from a representation introduced by Mäkinen [9], [10] for binary trees.

Manuscript received March 30, 2010.
Manuscript revised July 9, 2010.

[†]The author is with the Department of Industrial Management, Lunghwa University of Science and Technology, Taoyuan, Taiwan, ROC.
[††]The author is with the Institute of Information Science and Management, National Taipei College of Business, Taipei, Taiwan, ROC.
[†††]The author is with the Department of Information Management, National Taiwan University of Science and Technology, Taipei, Taiwan, ROC.
a) E-mail: eric@mail.lhu.edu.tw

DOI: 10.1587/transinf.E94.D.226

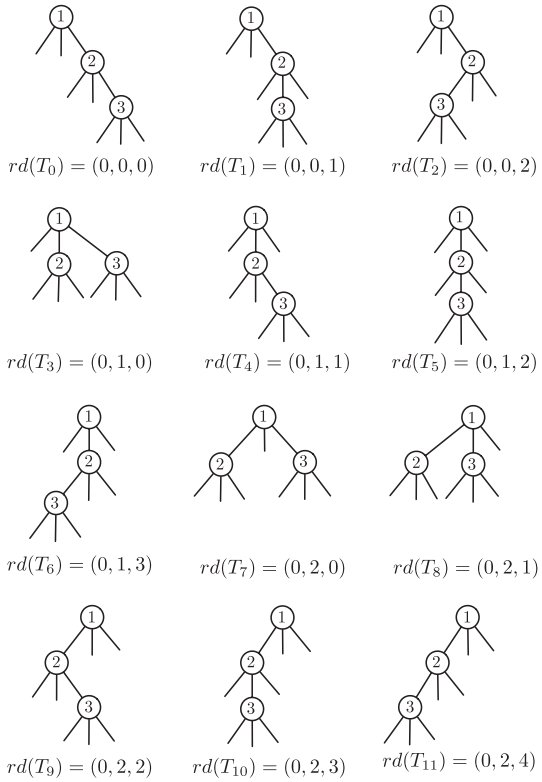


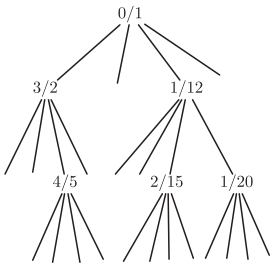
Fig. 1 The 3-ary trees with 3 internal nodes.

However, Mäkinen's distance representation uses the left arm (i.e., the path from the root to its leftmost descendant) of the tree to measure the distance of a node, and thus is referred to as the *left distance method*. Recently, Wu, Chang and Wang [16] extended the usage of RD-sequences to represent a wider class of trees called *non-regular trees*. Recall from [3] that the number of t-ary trees is given by $|\mathscr{T}_{n,t}| = \frac{1}{tn+1}\binom{tn+1}{n} = \frac{1}{(t-1)n+1}\binom{tn}{n}$. Figure 1 depicts twelve 3-ary trees for $n = 3$ and their RD-sequences.

In [21], Zaks defined another representation, called *Z-sequence*, of a t-ary tree T as follows. For each internal node $i \in T$, z_i denotes the visited order of node i in the preorder list of T, where both internal nodes and external nodes have to be traversed and counted in the list. The resulting sequence $z(T) = (z_1, z_2, \ldots, z_n)$ is called the Z-sequence of T. Note that any Z-sequence starts from 1 (i.e., $z_1 = 1$), while the first entry in an RD-sequence is 0 (i.e., $d_1 = 0$). Also, every Z-sequence satisfies $z_{i-1} < z_i \leqslant 1 + t(i-1)$. For example, Fig. 2 shows the RD-sequence and the Z-sequence of a particular 4-ary tree T with 6 internal nodes.

The following theorem shows that there is a one-to-one correspondence between an RD-sequence and a Z-sequence. Thus, the preprocessing time for ranking and unranking with Z-sequence may be also reduced by the same ideas of RD-sequences in the next section. For more complete discussion about the relationship among various representations of binary trees, the reader is referred to [8].

Theorem 1: Let T be a t-ary tree. Then $d_i + z_i = 1 + t(i-1)$ for every internal node $i \in T$.



i	1	2	3	4	5	6
d_i	0	3	4	1	2	1
z_i	1	2	5	12	15	20

Fig. 2 A 4-ary tree T where every internal node $i \in T$ is labeled by d_i/z_i.

Proof : Let K be the set of external nodes visited between two internal nodes i and $i+1$ in the preorder list of T, and let $k = |K|$. By definition, we have the following relevance:

$$z_{i+1} = z_i + (k+1) \tag{1}$$

We now consider the following two cases to show the relevance between d_i and d_{i+1} using the term k. The first case to consider is $0 \leqslant k < t$. In this case, node $i+1$ is the $(k+1)$th child of node i, and thus $d_{i+1} = d_i + t - (k+1)$. Next, we consider the case where $k \geqslant t$. We denote by $c(x) = \ell$ to mean that node x is the ℓth child of its parent. Also, let lca_i denote the least common ancestor of i and $i+1$ in T. Obviously, all the t children of i are external nodes and lca_i is the parent of $i+1$. Suppose that $c(i+1) = s$ and the unique path from lca_i to i is $lca_i = x_0, x_1, \ldots, x_p = i$. Then, for each $j = 1, \ldots, p$, the right distance $d(x_j)$ of node x_j can be computed by $d(x_j) = d(x_{j-1}) + t - c(x_j)$. By iterative substitution, we have

$$d(x_p) = d(x_0) + p \cdot t - \sum_{j=1}^{p} c(x_j)$$

Since $d_{i+1} = d(x_0) + t - s$ and the number of nodes in K which are not the children of node i is $k - t = (s-1) - c(x_1) + \sum_{j=2}^{p}(t - c(x_j))$, this comes into $d_i = d(x_p) = d_{i+1} - t + (k+1)$. Therefore, we obtain the following relevance from the above arguments:

$$d_{i+1} = d_i + t - (k+1). \tag{2}$$

From Eqs. (1) and (2) we obtain

$$d_{i+1} + z_{i+1} = d_i + z_i + t.$$

Since $d_1 = 0$ and $z_1 = 1$, we can solve the recurrence (i.e., an arithmetic sequence $d_i + z_i$ with the given initial term $d_1 + z_1 = 1$) to obtain the desired result. □

According to Theorem 1, it follows that to generate t-ary trees on n nodes using RD-sequences in lexicographic order, it suffices to generate t-ary trees using Z-sequences in the reverse of lexicographic order, and vice versa.

In the following, we apply a systematical way to describe all t-ary trees in $\mathscr{T}_{n,t}$. Given an integer $t \geqslant 2$, the *t-ary coding tree with n levels* is a rooted tree $\mathbb{T}$ constructed by the following rules: (1) Every node in $\mathbb{T}$ is associated with

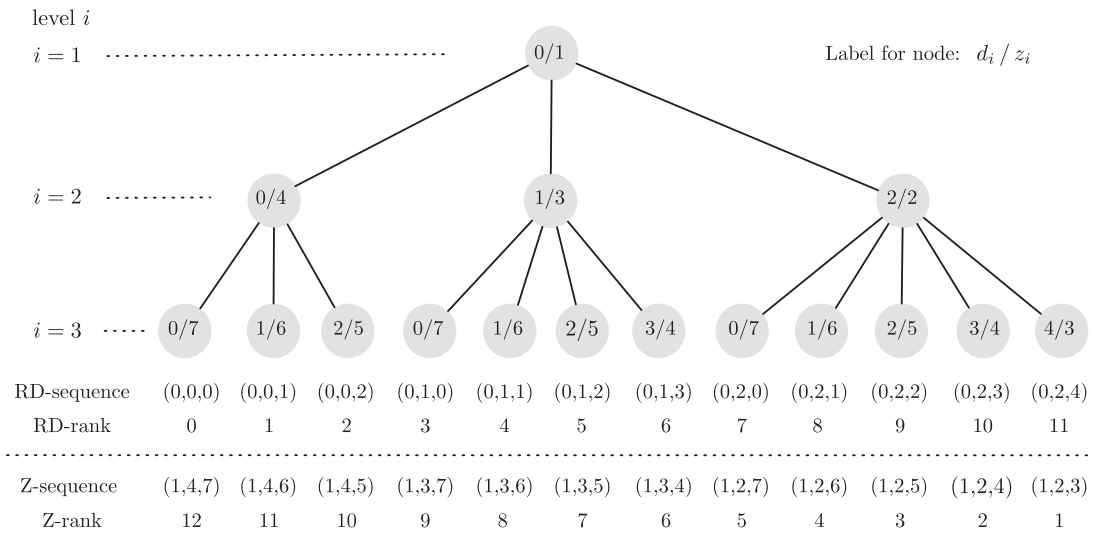


Fig. 3 The coding tree, RD-sequences, Z-sequences, and the rank of 3-ary trees.

a label; (2) The first level contains a single node (i.e., the root) with label 0; (3) Every nonleaf node (i.e., node not in the nth level) with label k has $t + k$ children and these children are labeled by $0, 1, \ldots, t + k - 1$ from left to right in the next level of $\mathbb{T}$. For simplicity, we merely mention *coding tree* if t and n are inessential. A result will be shown in Corollary 3 that the ith level of a coding tree contains $|\mathscr{T}_{i,t}|$ nodes. Consequently, the full labels along the path from the root to a leaf in $\mathbb{T}$ represent the RD-sequence of a t-ary tree with n internal nodes. Due to the relationship of d_i and z_i stated in Theorem 1, there is another way to label the nodes of a coding tree. Figure 3 demonstrates the 3-ary coding tree with three levels. In addition, it also shows the two types of sequences for 3-ary trees and their corresponding ranks.

Similar concepts of coding trees were independently introduced under various names including the backtracking tree [23], the recursion tree [8], and the coding tree [16], [17]. Backtracking trees and recursion trees are used to generate binary tree sequences while coding trees are constructed for t-ary trees and non-regular trees. For throughly studying the structure of t-ary coding trees, we first observe the following facts.

Fact 1: The degree (i.e., the number of children) of a node x in the ith level of $\mathbb{T}$ is at most $(t - 1)i + 1$.

For instance, we consider the coding tree in Fig. 3. The root has $(3 - 1) \cdot 1 + 1 = 3$ children, a node in the second level contains at most $(3 - 1) \cdot 2 + 1 = 5$ children, and so on. Thus, the range of labels for nodes in the ith level of $\mathbb{T}$ is between 0 and $((t - 1)(i - 1) + 1) - 1$.

Fact 2: For every two nodes x and y with the same label in the same level of $\mathbb{T}$, the subtree rooted at x and the subtree rooted at y are isomorphic.

For each level $i = 1, \ldots, n$ and $0 \leqslant k \leqslant (t - 1)(i - 1)$, let $X_{i,k}$ be the set consisting of nodes with label k in the ith level of $\mathbb{T}$. For example, $X_{3,2}$ contains three nodes with label 2 in the third level of the coding tree in Fig. 3. It is clear that every node in $X_{i,k}$ has the same degree. Since the height of the subtree of $\mathbb{T}$ rooted at a node $x \in X_{i,k}$ is $n - i$, we let $A_{i,(t-1)(n-i)+k}$ denote the number of leaves in such a subtree. Obviously, $A_{n,k} = 1$ for $0 \leqslant k \leqslant (t - 1)(n - 1)$ (i.e., the case of $i = n$ and thus every subtree rooted at a leaf of $\mathbb{T}$ contains a singleton). Similarly, we have $A_{n-1,(t-1)+k} = t + k$ for $0 \leqslant k \leqslant (t-1)(n-2)$ (i.e., the case of $i = n-1$). In general, for counting the number of leaves in the subtree of $\mathbb{T}$ rooted at a node x in level $i \leqslant n - 1$, by the decomposition of tree structure we may separately count the number of leaves in the subtree rooted at each child of x, and then sum up these counts for total. Thus we have the following formula:

$$A_{i,(t-1)(n-i)+k} = \sum_{j=(t-1)(n-i-1)}^{(t-1)(n-i)+k} A_{i+1,j}, \tag{3}$$

where $0 \leqslant k \leqslant (t-1)(i-1)$. Using Eq. (3), we can prove the following theorem by induction (see Appendix A).

Theorem 2: Given integers $t \geqslant 2$ and $n \geqslant 1$, for each $i = 1, \ldots, n$ and $0 \leqslant k \leqslant (t - 1)(i - 1)$ we have

$$A_{i,(t-1)(n-i)+k} = \frac{t + k}{mt + k}\binom{mt + k}{m - 1}, \tag{4}$$

where $m = n - i + 1$.

Corollary 3: The ith level of t-ary coding tree $\mathbb{T}$ contains $|\mathscr{T}_{i,t}|$ nodes.

Proof : It can be derived by setting $k = 0$ and substituting i for $n - i + 1$ in Eq. (4). Since $m = i$ in this case, it yields

$$A_{n-i+1,(t-1)(i-1)} = \frac{t}{it}\binom{it}{i - 1} = \frac{1}{it + 1}\binom{it + 1}{i} = |\mathscr{T}_{i,t}|.$$

□

Using Eq. (3), we can set up a table called *t-ary recursion table*. When $t = 2$, the special case is the well-known Catalan's triangle table [14]. Table 1 shows the 3-ary recursion table for $n = 6$. This table can be visualized

Table 1 The 3-ary recursion table for $n = 6$.

A			$(t-1)(n-i)+k$										
n	i	$0 \leqslant k \leqslant (t-1)(i-1)$	0	1	2	3	4	5	6	7	8	9	10
6	1	$k = 0$											1428
	2	$0 \leqslant k \leqslant 2$									273	455	700
	3	$0 \leqslant k \leqslant 4$							55	88	130	182	245
	4	$0 \leqslant k \leqslant 6$					12	18	25	33	42	52	63
	5	$0 \leqslant k \leqslant 8$			3	4	5	6	7	8	9	10	11
	6	$0 \leqslant k \leqslant 10$	1	1	1	1	1	1	1	1	1	1	1

Table 2 The 3-ary accumulation table for $n = 6$.

B			$(t-1)(n-i)+k$										
n	i	$0 \leqslant k \leqslant (t-1)(i-1)$	0	1	2	3	4	5	6	7	8	9	10
6	1	$k = 0$											0
	2	$0 \leqslant k \leqslant 2$									0	273	728
	3	$0 \leqslant k \leqslant 4$							0	55	143	273	455
	4	$0 \leqslant k \leqslant 6$					0	12	30	55	88	130	182
	5	$0 \leqslant k \leqslant 8$			0	3	7	12	18	25	33	42	52
	6	$0 \leqslant k \leqslant 10$	0	1	2	3	4	5	6	7	8	9	10

as a staircase where the corner entry of step i is equal to $A_{i,(t-1)(n-i)} = |\mathscr{T}_{n-i+1,t}|$ by Corollary 3. Also, an observation from Eq. (3) shows that the value of any entry, except those in the bottom row, is equal to the prefix sum of the entries in the next lower row (e.g. $A_{3,6} = 55 = \sum_{j=4}^{6} A_{4,j} = 12 + 18 + 25$ in Table 1). Thus, the value of each entry, except those in the bottom row and corner entries, is equal to the one below plus the one at its left (e.g. $A_{3,7} = 88 = A_{4,7} + A_{3,6} = 33 + 55$ in Table 1). This immediately implies the following.

Corollary 4: Given integers $t \geqslant 2$ and $n \geqslant 2$, for each $i = 1, \ldots, n-1$ and $1 \leqslant k \leqslant (t-1)(i-1)$ we have

$$A_{i,(t-1)(n-i)+k} = A_{i,(t-1)(n-i)+k-1} + A_{i+1,(t-1)(n-i)+k}. \tag{5}$$

For the efficiency of computation, we also define the following formula:

$$B_{i,(t-1)(n-i)+k} = \begin{cases} 0 & \text{if } k = 0 \\ \sum_{j=0}^{k-1} A_{i,(t-1)(n-i)+j} & \text{otherwise} \end{cases}, \tag{6}$$

where $0 \leqslant k \leqslant (t-1)(i-1)$. This means that the term $B_{i,(t-1)(n-i)+k}$ is the prefix sum of the first k nonblank entries in the ith row of a recursion table. A table built from Eq. (6) is thus named as a *t-ary accumulation table*. Table 2 shows the case for $t = 3$ and $n = 6$. Clearly, if $k \neq 0$, we can rewrite $B_{i,(t-1)(n-i)+k}$ as follows:

$$B_{i,(t-1)(n-i)+k} = B_{i,(t-1)(n-i)+k-1} + A_{i,(t-1)(n-i)+k-1}. \tag{7}$$

Using Eqs. (4) and (7), the following theorem can be proved by induction (see Appendix B).

Theorem 5: Given integers $t \geqslant 2$ and $n \geqslant 1$, for each $i = 1, \ldots, n$ and $0 \leqslant k \leqslant (t-1)(i-1)$ we have

$$B_{i,(t-1)(n-i)+k} = \frac{k}{mt+k}\binom{mt+k}{m}, \tag{8}$$

where $m = n - i + 1$.

In addition, from Eq. (8) we can obtain the following formulae which are useful for designing ranking algorithm and unranking algorithm in Sect. 3. In particular, if an entry in the accumulation table is given, we can compute the immediately right entry, the immediately left entry, the immediately upper entry, and the immediately lower entry by using these formulae, respectively.

Corollary 6: Given integers $t \geqslant 2$ and $n \geqslant 2$, for each $i = 2, \ldots, n$ and $1 \leqslant k \leqslant (t-1)(i-1) - 1$ we have

$$B_{i,(t-1)(n-i)+(k+1)} = B_{i,(t-1)(n-i)+k} \cdot \frac{(k+1)(mt+k)}{k(m(t-1)+k+1)}, \tag{9}$$

where $m = n - i + 1$.

Corollary 7: Given integers $t \geqslant 2$ and $n \geqslant 2$, for each $i = 2, \ldots, n$ and $1 \leqslant k \leqslant (t-1)(i-1)$ we have

$$B_{i,(t-1)(n-i)+(k-1)} = B_{i,(t-1)(n-i)+k} \cdot \frac{(k-1)(m(t-1)+k)}{k(mt+k-1)}, \tag{10}$$

where $m = n - i + 1$.

Corollary 8: Given integers $t \geqslant 2$ and $n \geqslant 2$, for each $i = 2, \ldots, n$ and $t - 1 \leqslant k \leqslant (t-1)(i-1)$ we have

$$B_{i-1,(t-1)(n-i)+k} = B_{i,(t-1)(n-i)+k} \cdot \frac{(k-t+1)(mt+k)}{k(m+1)}, \tag{11}$$

where $m = n - i + 1$.

Corollary 9: Given integers $t \geqslant 2$ and $n \geqslant 2$, for each $i = 2, \ldots, n-1$ and $1 \leqslant k \leqslant (t-1)(i-1)$ we have

$$B_{i+1,(t-1)(n-i)+k} = B_{i,(t-1)(n-i)+k} \cdot \frac{m(k+t-1)}{k(mt+k-1)}, \tag{12}$$

where $m = n - i + 1$.

3. Ranking and Unranking Algorithms

In this section, we first determine the rank of a t-ary tree T according to the RD-sequences in lexicographic order. Let $\mathbb{T}$ be the t-ary coding tree. Recall that $A_{i,(t-1)(n-i)+k}$ denotes the number of leaves of the subtree rooted at a node with label k in the ith level of $\mathbb{T}$. Also we know from Eq. (6) that $B_{i,(t-1)(n-i)+k}$ indicates the total number of leaves for those subtrees rooted at nodes with labels from 0 to $k-1$. Since a full label $(d_1, d_2, \ldots, d_n)$ along the path from the root of $\mathbb{T}$ to a leaf represents a certain t-ary tree T, we can define the rank of T by using the order (from left to right) of this appointed leaf in $\mathbb{T}$. It follows that

$$Rank(d_1, d_2, \ldots, d_n) = \sum_{i=1}^{n} B_{i,(t-1)(n-i)+d_i}. \tag{13}$$

Note that $Rank(0, 0, \ldots, 0) = 0$ and $Rank(0, t-1, 2(t-1), \ldots, (n-1)(t-1)) = |\mathscr{T}_{n,t}| - 1$. For example, we consider a 3-ary tree T with $rd(T) = (0, 2, 1, 0, 1, 2)$. From Table 2, we can compute $Rank(0, 2, 1, 0, 1, 2) = B_{1,10} + B_{2,10} + B_{3,7} + B_{4,4} + B_{5,3} + B_{6,2} = 0 + 728 + 55 + 0 + 3 + 2 = 788$.

Based on Eq. (13), we design the following algorithm which takes $rd(T) = (d_1, d_2, \ldots, d_n)$ as the input for computing the rank of a t-ary tree T. For convenience, we assume that there exists at least one nonzero item in the sequence $d_1, d_2, \ldots, d_n$. Indeed, it is unnecessary to build the accumulation table in our algorithm. An alternative way shows that using Eqs. (9) and (11) instead of Eq. (8) can efficiently reduce the complexity for computing the rank.

Algorithm RANKING

Step 1. Let $\ell = \max\{i : d_i \neq 0 \text{ and } 1 \leqslant i \leqslant n\}$ and
$s = \min\{i : d_i \neq 0 \text{ and } 1 \leqslant i \leqslant n\}$;
$B = k = (n-\ell)(t-1) + d_\ell$;
for $m = 1$ to $n - \ell$ **do**
// Refer to Eq. (11), move up
$B = B \cdot \frac{(k-t+1)(mt+k)}{k(m+1)}$;
$k = k - (t-1)$;
endfor
$Rank = B$;

Step 2. $j = \ell$;
while $j > s$ **do**
$\ell = \max\{i : d_i \neq 0 \text{ and } s \leqslant i < j\}$;
$m = n - j + 1$;
$h = (j-\ell)(t-1) + d_\ell - d_j$;
for $k = d_j$ to $d_j + h - 1$ **do**
// Refer to Eq. (9), move right
$B = B \cdot \frac{(k+1)(mt+k)}{k(m(t-1)+k+1)}$;
endfor
$k = d_j + h$;
for $m = n - j + 1$ to $n - \ell$ **do**
// Refer to Eq. (11), move up
$B = B \cdot \frac{(k-t+1)(mt+k)}{k(m+1)}$;
$k = k - (t-1)$;
endfor
$Rank = Rank + B$;
$j = \ell$;
endwhile

Step 3. Output $Rank$;

Algorithm RANKING calculates the rank by a sequence of nonzero terms $B_{i,(t-1)(n-i)+d_i}$ with index i from n downto 1. Step 1 first determines the largest index of nonzero term B. In Step 2, we consider any two consecutive nonzero terms with indices j and ℓ such that $j > \ell$. Since $d_j \leqslant d_\ell + (j-\ell)(t-1)$, we have $(t-1)(n-j) + d_j \leqslant (t-1)(n-\ell) + d_\ell$. Thus, if the given RD-sequence is valid, the term with index ℓ must appear at the upper or the upper-right of the term with index j in the t-ary accumulation table, which depends on the value of h to be zero or not. Then, we can find the next nonzero term by successively traversing the entries of the table through h steps in horizontal, and then through $j - \ell$ steps in vertical. We repeat this procedure until all subsequent nonzero terms are added into the rank. Instead of table search, we obtain all nonzero terms using the calculations of Eqs. (9) and (11), and therefore the correctness of algorithm directly follows. Since each of Eqs. (9) and (11) can be computed in a constant time if the previous nonzero term is provided, and since the total steps traversed horizontally and vertically in the table is no more than $(n-1)(t-1) + (n-1) = t(n-1)$, we conclude the following.

Theorem 10: Algorithm RANKING can correctly determine the rank of a t-ary tree with n internal nodes in $O(tn)$ time.

In the following, we give a reverse procedure, called UNRANKING, that converts a positive integer N to its corresponding RD-sequence. We are given n, t and N, and look for a sequence $d_1, d_2, \ldots, d_n$ such that $Rank(d_1, d_2, \ldots, d_n) = N$. Initially, we assume that $d_i = 0$ for all $i = 1, \ldots, n$.

Algorithm UNRANKING

Step 1. $B = |\mathscr{T}_{n,t}| = \frac{1}{tn+1}\binom{tn+1}{n}$;

Step 2. $i = k = 1$;
while $N > 0$ **do**
$i = i + 1$;
$m = n - i + 1$;
$k = k + (t-1)$;
while $k > 1$ and $B > N$ **do**
// Refer to Eq. (10), move left
$B = B \cdot \frac{(k-1)(m(t-1)+k)}{k(mt+k-1)}$;
$k = k - 1$;
endwhile
if $B \leqslant N$ **then**
$N = N - B$;
$d_i = k$;

endif

// Refer to Eq. (12), move down

$B = B \cdot \frac{m(k+t-1)}{k(mt+k-1)}$;

endwhile

Step 3. Output $(d_1, d_2, \ldots, d_n)$;

Algorithm Unranking decomposes N into a sequence $B_{i,(t-1)(n-i)+d_i}$ for $i = 1, \ldots, n$. For determining which terms are appropriate in the sequence, similar to Ranking, the algorithm performs the search in accumulation table using Eqs. (10) and (12) and without really building the table. Step 1 takes $O(n)$ time to pick out the upper bound of $B_{1,(t-1)(n-1)+d_1}$. In Step 2, for each round of the outer loop, it determines an entry B from a row of the table as an appropriate term, and then updates N by subtracting B from N. The choice of entry B is percolated by the inner loop which can be viewed as a search from right to left in a row of the table and such that B matches a possibly largest value to accommodate the current N. Since each of Eqs. (10) and (12) can be computed in a constant time and the total time is no more than $(n-1)(t-1) + (n-1) = t(n-1)$, we have the following theorem.

Theorem 11: Given a positive integer N, Algorithm Unranking can correctly determine an RD-sequence $d_1, d_2, \ldots, d_n$ of a t-ary tree such that $Rank(d_1, d_2, \ldots, d_n) = N$ in $O(tn)$ time.

4. Conclusion

In this paper, we apply RD-sequences to represent t-ary trees since such a representation is conceptually simple. From the structure of coding trees, we can describe all t-ary trees with n internal nodes in a systematical way by using RD-sequences. As a result, the ranking and unranking algorithms for t-ary trees can be carried out. Four expedient formulae derived from the t-ary accumulation table are used to reduce the complexity and thus both the ranking and unranking algorithms of t-ary trees can be run in O(tn) time and without computing all the entries of the table. As a corollary, we obtain linear time algorithms for solving the ranking and unranking problems on binary trees, respectively.

References

[1] H. Ahrabian and A. Nowzari-Dalini, "Generation of t-ary trees with ballot-sequences," Int. J. Comput. Math., vol.80, pp.1243–1249, 2003.

[2] H. Ahrabian, A. Nowzari-Dalini, and E. Salehi, "Gray code algorithm for listing k-ary trees," Studies in Informatics and Control, vol.13, pp.243–251, 2004.

[3] D.E. Knuth, The Art of Computer Programming, Vol.1: Fundamental Algorithms, Addison-Wesley, Reading, MA, 1968.

[4] J.F. Korsh, "Loopless generation of k-ary tree sequences," Inf. Process. Lett., vol.52, pp.243–247, 1994.

[5] J.F. Korsh, "Generating t-ary trees in linked representation," Comput. J., vol.48, pp.488–497, 2005.

[6] J.F. Korsh and P. LaFollette, "Loopless generation of Gray code for k-ary trees," Inf. Process. Lett., vol.70, pp.7–11, 1999.

[7] J.F. Korsh and S. Lipschutz, "Shifts and loopless generation of k-ary trees," Inf. Process. Lett., vol.65, pp.235–240, 1998.

[8] J.M. Lucas, D. Roelants van Baronaigien, and F. Ruskey, "On rotations and the generation of binary trees," J. Algorithms, vol.15, pp.343–366, 1993.

[9] E. Mäkinen, "Left distance binary tree representations," BIT, vol.27, pp.163–169, 1987.

[10] E. Mäkinen, "Generating random binary trees – A survey," Inf. Sci., vol.115, pp.123–136, 1999.

[11] D. Roelants van Baronaigien, "A loopless Gray-code algorithm for listing k-ary trees," J. Algorithms, vol.35, pp.100–107, 2000.

[12] D. Roelants van Baronaigien and F. Ruskey, "Generating t-ary trees in A-order," Inf. Process. Lett., vol.27, pp.205–213, 1988.

[13] F. Ruskey, "Generating t-ary trees lexicographically," SIAM J. Comput., vol.7, pp.424–439, 1978.

[14] N.J.A. Sloane, "The on-line encyclopedia of integer sequences." Sequence A009766.

[15] A.E. Trojanowaki, "Ranking and listing algorithms for k-ary trees," SIAM J. Comput., vol.7, pp.492–509, 1978.

[16] R.-Y. Wu, J.-M. Chang, and Y.-L. Wang, "Loopless generation of non-regular trees with a prescribed branching sequence," Comput. J., vol.53, pp.661–666, 2010.

[17] L. Xiang, K. Ushijima, and S.G. Akl, "Generating regular k-ary trees efficiently," Comput. J., vol.43, pp.290–300, 2000.

[18] L. Xiang, K. Ushijima, and Y. Asahiro, "Coding k-ary trees for efficient loopless generation in lexicographic order," Proc. International conference on Information Technology: Coding and Computing (ITCC'02), IEEE Computer Society Press, pp.396–401, Las Vegas, April, 2002.

[19] L. Xiang, K. Ushijima, and C. Tang, "Efficient loopless generation of Gray codes for k-ary trees," Inf. Process. Lett., vol.76, pp.169–174, 2000.

[20] L. Xiang, K. Ushijima, and C. Tang, "On generating k-ary trees in computer representation," Inf. Process. Lett., vol.77, pp.231–238, 2001.

[21] S. Zaks, "Lexicographic generation of ordered trees," Theor. Comput. Sci., vol.10, pp.63–82, 1980.

[22] S. Zaks, "Generation and ranking of k-ary trees," Inf. Process. Lett., vol.14, pp.44–48, 1982.

[23] D. Zerling, Generating binary trees using rotations, J. ACM, vol.32, pp.694-701, 1985.

Appendix A: Proof of Theorem 2

The proof is by induction on m. For $m = 1$ (i.e., $i = n$), it is easy to check $A_{n,k} = \frac{(t+k)}{(t+k+1)(t+k)}\binom{t+k+1}{1} = 1$ for $0 \leqslant k \leqslant (t-1)(n-1)$. Suppose that Eq. (4) holds for $m = \ell < n$ and $0 \leqslant k \leqslant (t-1)(n-\ell)$, i.e.,

$$A_{n-\ell+1,(t-1)(\ell-1)+k} = \frac{t+k}{\ell t+k}\binom{\ell t+k}{\ell-1}.$$

We now consider $m = \ell + 1$ (i.e., $i = n - \ell$) and $0 \leqslant k \leqslant (t-1)(n-\ell-1)$. Before the proof, we first note that it holds

$$\frac{p}{\ell}\binom{\ell t+p-1}{\ell-1} + \frac{t+p}{\ell t+p}\binom{\ell t+p}{\ell-1} = \frac{p+1}{\ell}\binom{\ell t+p}{\ell-1}$$

for $1 \leqslant p \leqslant t-1+k$. By Eq. (3), we have

$$
\begin{aligned}
&A_{n-\ell,(t-1)\ell+k} \\
&= \sum_{j=(t-1)(\ell-1)}^{(t-1)\ell+k} A_{n-\ell+1,j} \\
&= A_{n-\ell+1,(t-1)(\ell-1)} + A_{n-\ell+1,(t-1)(\ell-1)+1} + \\
&\quad A_{n-\ell+1,(t-1)(\ell-1)+2} + \cdots + A_{n-\ell+1,(t-1)\ell+k} \\
&= \frac{1}{\ell}\binom{\ell t}{\ell-1} + \frac{t+1}{\ell t+1}\binom{\ell t+1}{\ell-1} + \frac{t+2}{\ell t+2}\binom{\ell t+2}{\ell-1} + \cdots \\
&\quad + \frac{t+(t-1+k)}{\ell t+(t-1+k)}\binom{\ell t+(t-1+k)}{\ell-1} \\
&= \frac{2}{\ell}\binom{\ell t+1}{\ell-1} + \frac{t+2}{\ell t+2}\binom{\ell t+2}{\ell-1} + \frac{t+3}{\ell t+3}\binom{\ell t+3}{\ell-1} + \cdots \\
&\quad + \frac{t+(t-1+k)}{\ell t+(t-1+k)}\binom{\ell t+(t-1+k)}{\ell-1} \\
&\vdots \\
&= \frac{t+k}{\ell}\binom{\ell t+(t-1+k)}{\ell-1} \\
&= \frac{t+k}{(\ell+1)t+k}\binom{(\ell+1)t+k}{\ell}.
\end{aligned}
$$

□

Appendix B: Proof of Theorem 5

The proof is by induction on k. The theorem is trivially true for $k = 0$ because $B_{i,(t-1)(n-i)} = 0$. We now consider $1 \leqslant k \leqslant (t-1)(i-1)$ and suppose that the theorem is true for $k-1$. By Eq. (7), we have $B_{i,(t-1)(n-i)+k} = B_{i,(t-1)(n-i)+k-1} + A_{i,(t-1)(n-i)+k-1}$. Thus, we can use induction hypothesis and Eq. (4) to get

$$
\begin{aligned}
&B_{i,(t-1)(n-i)+k} \\
&= \frac{k-1}{mt+k-1}\binom{mt+k-1}{m} + \frac{t+k-1}{mt+k-1}\binom{mt+k-1}{m-1} \\
&= \frac{k}{mt+k}\binom{mt+k}{m}.
\end{aligned}
$$

□

Ro-Yu Wu received his B.S. degree in industrial engineering from the Tunghai University in 1985, the M.B.A. degree in industrial management from the National Cheng Kung University in 1987, and the Ph.D. degree in information management from the National Taiwan University of Science and Technology in 2007. Currently, he is an associate professor in the Department of Industrial Management of Lunghwa University of Science and Technology. His research interests include reliability, graph theory, and combinatorial algorithms.

Jou-Ming Chang received his B.S. degree in applied mathematics from the Chinese Culture University in 1987, the M.S. degree in information management from the National Chiao Tung University in 1992, and the Ph.D. degree in computer science and information engineering from the National Central University in 2001. Currently, he is a professor and the head of the Institute of Information Science and Management in the National Taipei College of Business (NTCB). His research interests include algorithm design and analysis, graph theory and combinatorics, parallel and distributed computing.

Yue-Li Wang received the B.S. and M.S. degrees from the Information Engineering Department of Tam-Kang University in 1975 and 1979, respectively, and the Ph.D. degree in information engineering from the National Tsing-Hua University in 1988. Now, he is a professor in the Department of Information Management of National Taiwan University of Science and Technology. His research interests include graph theory, algorithm analysis, and parallel computing.