An Improved Algorithm for Ranking Zaks’ Sequences in Gray-code Order

Ro-Yu Wu, Chir-Ho Chang, and Jou-Ming Chang

Abstract—A k-ary tree is a rooted and ordered tree such that every internal node has exactly k children. A natural representation of k-ary trees is the use of z-sequences introduced by Zaks in 1980. Under such representations, algorithms for generating z-sequences of k-ary trees in Gray-code order was independently developed by van Baronaigien (2000) and Xiang et al. (2000). Based on such a Gray-code order, Ahmadi-Adl et al. (2011) recently proposed ranking and unranking algorithms and showed that each algorithm can be run in $O(kn^2)$ time, where $n$ is the number of internal nodes in a k-ary tree. In this paper, we make an improvement of the ranking algorithm. The time complexity and space requirement in our algorithm are $O(n^2 \log kn)$ and $O(kn)$, respectively.

Index Terms—loopless algorithms; ranking algorithms; lexicographic order; Gray-code order;

I. INTRODUCTION

Exhaustively generating a class of combinatorial objects has attracted a lot of research due to the importance for many applications, e.g., searching for the desired object with a certain feature among all candidates, looking for a counterexample to some conjecture, or analyzing the average performance of an algorithm over all possible inputs. As usual, combinatorial objects are encoded as integer sequences so that these sequences can be generated in a particular order such as the lexicographic order [3, 4, 12, 20] or a Gray-code order [2, 5, 10, 15, 17, 19]. For efficiency of generation, algorithms to generate objects in the lexicographic order are demanded to run in a constant amortized time. By contrast, algorithms to generate objects in a Gray-code are demanded to run in constant time for each generation. Accordingly, a loopless algorithm for generating objects in a Gray-code is implemented non-recursively by using, after the initialization of the first object, only assignment statements and “if-then-else” statements. The reader is referred to [13] for an excellent survey of generating combinatorial objects in Gray-code orders.

Given a specific order on the class of combinatorial objects, a ranking algorithm is a function that determines the rank of a given object in the exhaustive generated list, and an unranking algorithm is one that generates the object (or its representation) corresponding to a given rank. Trees are one of the most important and fundamental combinatorial objects in computer science. For $k > 2$, a k-ary tree is a rooted tree such that every internal node has exactly $k$ disjoint subtrees. Many algorithms have been developed for generating k-ary trees (as well as binary trees) with $n$ internal nodes. Moreover, many ranking algorithms [1, 2, 4, 11, 12, 14, 16, 18, 21] and unranking algorithms [1, 2, 4, 12, 16, 18] for diverse representations of k-ary trees with n internal nodes have been proposed. Among all representations of k-ary trees, a well-formed representation called z-sequences was first introduced by Zaks [20]. Roelants van Baronaigien [10] and Xiang et al. [19] independently presented a loopless algorithm for generating k-ary trees encoded by z-sequences in a Gray-code order. Based on such a Gray-code order, Ahmadi-Adl et al. [2] recently presented $O(kn^2)$-time ranking and unranking algorithms, where $n$ is the number of internal nodes in a k-ary tree.

In this paper, based on the Gray-code order presented in [10] and [19], we propose an improved ranking algorithm for dealing with k-ary trees encoded by z-sequences, We show that the ranking algorithm can be done in $O(n^2 \log kn)$ time and $O(kn)$ space.

II. PROCEDURE FOR PAPER SUBMISSION

Obviously, a k-ary tree with $n$ internal nodes has $(k-1)n+1$ leaves. Let $R_k$ denote the set of all k-ary trees with $n$ internal nodes. Then, the total number of k-ary trees with $n$ internal nodes is known to have the value $|R_k| = \frac{1}{kn+1} \binom{(kn)^2}{n} = \frac{1}{kn+1} \binom{kn}{n}$. Among all representations of k-ary trees, a frequently used representation is the so-called z-sequences defined by Zaks [20]. Given a k-ary tree $T \in R_k$, we traverse $T$ in preorder (i.e., visit the root and then recursively the subtrees of $T$ from left to right). Then, the z-sequence of $T$, denoted by $z(T) = z_1 z_2 \ldots z_n$, is an integer sequence such that $z_i$ denotes the position of the $i$-th internal node appeared in the visited order. For example, Fig. 1 shows a 3-ary tree $T$ with $z(T) = (1, 2, 5, 6, 13, 16)$.

The following theorem establishes the relation between k-ary trees and z-sequences.

Theorem 1: [20] The following two sets are in one-to-one correspondence:

- All the trees in $R_{k,n}$
- All the integer sequences $(z_i)_{i=1}^n$, such that $z_1 = 1$ and $z_i < z_{i+1}$ for all $i = 2, 3, \ldots, n$.

Manuscript received March 1, 2015. This work was supported in part by Ministry of Science and Technology of Taiwan under Grants MOST103-2221-E-262-023 and NSC102-2221-E-141-001-MY3.

Ro-Yu Wu and Chir-Ho Chang are with the Department of Information Management, Linliao University of Science and Technology, Taoyuan, Taiwan, ROC. (e-mail: eric@mail.lhu.edu.tw)

Jou-Ming Chang is with the Institute of Informatics and Decision Sciences, National Taipei University of Business, Taipei, Taiwan, ROC. (e-mail: spade@ntub.edu.tw).
A flip-flap tree, denoted by $T_{k,n}$, is a rooted labeled tree with $n$ levels and satisfies the following properties:

- The first level contains the only one node (i.e., the root) with label 1.
- For each level $i$ with $i < n$, a node with label $z$ has $k \cdot i - z + 1$ sons labeled by $z + 1, z + 2, \ldots, k \cdot i + 1$ in the next level, where the sons are arranged in either
  \[ k \times i + 1, z + 1, z + 2, \ldots, k \times i + 1 \]  
  (which is called an up fragment)
  or
  \[ k \times i, k \times i - 1, \ldots, z + 2, z + 1, k \times i + 1 \]  
  (which is called a down fragment).

If the sons of a node are arranged in an up fragment, then the sons of its adjacency siblings, if they exist, are arranged in a down fragment, and vice versa. (That is, up fragments and down fragments alternately appear in each level.)

We now easily see that nodes on the boundary of a fragment in the level $i$ are labeled by either $k \times (i - 1) + 1$ (the largest label) or $k \times i + 1$ (the second largest label). Furthermore, the full labels along a path from the root to a leaf in $T_{k,n}$ indicate the $z$-sequence of a $k$-ary tree. Note that every two adjacent $z$-sequences in a flip-flap tree differ in exactly one digit. This provides the base for the design of a loopless algorithm. For example, Fig. 2 shows a flip-flap tree $T_{3,4}$, where the first fragment in each level is an up fragment. Thus, the first $z$-sequence appeared in $T_{3,4}$ is (1, 4, 7, 10). Also, a path with bold lines from the root to a leaf in $T_{3,4}$ represents the 3-ary trees with $z$-sequence (1, 2, 3, 7).

### III. RANKING ALGORITHM

We observe that, for $i = 1, 2, \ldots, n$, the smallest label and the largest label for nodes in the $i$th level of $T_{k,n}$ are $i$ and $k(i - 1) + 1$, respectively. Let $A_{i,z}$ denote the number of leaves in the subtree rooted at a node with label $z$ in the $i$th level of $T_{k,n}$. For instance, if we consider the flip-flap tree $T_{3,4}$ shown in Figure 2, then $A_{1,1}=55$, $A_{1,2}=25$, $A_{3,3}=7$ and $A_{1,7}=1$. Otherwise, for any $n \leq z \leq (n - 1) + 1$, $A_{n,z}=1$ holds. By Theorem 1, we have

$$A_{i,z} = \sum_{j=z+1}^{k \times i + 1} A_{i+1,j} \quad \text{(1)}$$

where $1 \leq i \leq n - 1$ and $i \leq z \leq k(i - 1) + 1$. For the efficiency of computation, we define the following formula

$$B_{i,z} = \sum_{j=z}^{k \times (i-1) + 1} A_{i,j} \quad \text{(2)}$$

where $1 \leq i \leq n$ and $i \leq z \leq k(i - 1) + 1$. The table built from Eq. (2) is named as the accumulation table with respect to $T_{k,n}$. For instance, if we consider the flip-flap tree $T_{3,4}$, we can build the following tables for calculating $A_{i,z}$ and $B_{i,z}$.

Let $b(i, z) = (k(i - 1) + 1) - z + 1$ denote the entry of $i$th row and $z$th column from right to left in the accumulation table, where $1 \leq i \leq n$ and $i \leq z \leq k(i - 1) + 1$. We can prove the following formula by induction:

$$B_{i,z} = \frac{b(i, z)}{mk + b(i, z)} \left( \frac{m^k + b(i, z)}{m} \right) \quad \text{(3)}$$

where $m = n - i + 1$.

For a given $k$-ary tree $T$ associated with the $z$-sequence $z(T) = (z_1, z_2, \ldots, z_n)$, let $x_n$ denote the node with label $z_n$ in the flip-flap tree $T_{k,n}$ such that the full labels along the path $x_n, x_{n-1}, \ldots, x_1$ represent the given $z$-sequence. For each $i = 1, 2, \ldots, n$, let $R(i)$ be the rank of $x_i$ in the $i$th level of $T_{k,n}$, where the rank is ordered from left to right in a level of the tree and a ranking always starts with 0. For instance, if we consider a $z$-sequence (1, 2, 3, 7, 10) in $T_{3,4}$, we have $R(1) = 0$, $R(2) = 1$, $R(3) = 6$ and $R(4) = 31$. Note that $R(1) = 0$ is always true and our ranking algorithm is to obtain $R(n)$.

Since each level of $T_{k,n}$ begins with a up fragment and the two types of fragments appear alternately, it is easy to check the following property.

**Proposition 1.** For $1 \leq i < n$, if the rank of a node in the $i$th level of $T_{k,n}$ is even (respectively, odd), then its sons are arranged in an up fragment (respectively, a down fragment).

We are now in a position to determine the rank of a $k$-ary tree in the Gray-code order of [10] and [19]. We first set $R(i) = 0$ for $i = 1, 2, \ldots, n$. For each $i = 1, \ldots, n - 1$, let $f = k \times i + 1$ denote the largest label in the $(i + 1)$th level of $T_{k,n}$. Then, the computation of $R(i)$ requires $i - 1$ updates by means of Eqs. (4) and (5). If $i \neq n$ and $x_n$ is not the first node in a fragment, we let $y_n$ be the node with rank $R(i) + 1$ in the $(i + 1)$th level of $T_{k,n}$. For each $j = i + 1, \ldots, n$, we update $R(j)$ to represent the rank of the rightmost descendant of $y_n$ in the $j$th level of $T_{k,n}$ (including $y_n$ itself if $j = i + 1$). Let $h = n - i + 1$. By Proposition 1, there exist two cases to compute $R(j)$. If $R(i)$ is even, the sons of $x_i$ are arranged in an up fragment. In this case, we have

$$R(j) = R(i) + (B_{h,(n-j)} \times k + z_i + 1) - B_{h,(n-j)} \times k + z_i + B_{h,(k-1)} \times k + 1) \quad \text{(4)}$$

On the other hand, if $R(i)$ is odd, the sons of $x_i$ are arranged in a down fragment. In this case, we have
Fig. 2. A flip-flop tree $T_{b,\ell}$ encoded by $x$-sequences.

<table>
<thead>
<tr>
<th>$A_{i,\ell}$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>$A_{i,\ell}$</td>
<td>55</td>
<td>25</td>
<td>18</td>
<td>13</td>
<td>6</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$B_{i,\ell}$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>$B_{i,\ell}$</td>
<td>55</td>
<td>30</td>
<td>12</td>
<td>16</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Fig. 4. Illustration of Eq. (4).

Fig. 5. Illustration of Eq. (5).
Finally, the algorithm outputs \( R(4) = 31 \).

The last terms in the right hand side of Eqs. (4) and (5) are called the incremental change of \( R(j) \) in the update. The meanings of incremental changes are illustrated in Figure 4 and Figure 5, respectively. According to Eqs. (4) and (5), we design the following algorithm.

Function Ranking(\( z_1, z_2, \ldots, z_n \))

begin
  for \( i = 1 \) to \( n \) do \( R(i) = 0 \);
  for \( i = 1 \) to \( n \) do \( f = k \times i + 1; \)
    if \( R(i) = 0 \) mod 2 then // an up fragment
      for \( j = i + 1 \) to \( n \) do
        \( h = n - j + i + 1; \)
        if \( z_{i+1} \neq \ell \) then
          \( R(j) = R(j) + (B_{h-(n-j)xk+z_{i+1}+1} - B_{h-(n-j)xk+z_{i+1}}); \)
        else
          \( R(j) = R(j) + (B_{h-(n-j)xk+z_{i+1}+1}); \)
    else // a down fragment
      for \( j = i + 1 \) to \( n \) do
        \( h = n - j + i + 1; \)
        if \( z_{i+1} \neq \ell \) then
          \( R(j) = R(j) + (B_{h-(n-j)xk+z_{i+1}+1} - B_{h-(n-j)xk+z_{i+1}}); \)
        else
          \( R(j) = R(j) + (B_{h-(n-j)xk+z_{i+1}}); \)
  return \( R(n); \)

Example 1. We consider a 3-ary tree \( T \) with \( z(T) = (1, 2, 3, 7) \) and perform Ranking(1, 2, 3, 7). Initially, \( R(1) = R(2) = R(3) = R(4) = 0 \).

When \( i = 1 \), since \( R(1) = 0 \) is even, the sons of \( x \), are arranged in an up fragment. Since \( x \) is not the largest element in the fragment, we have

\[
R(2) = R(2) + B_{4,8} + B_{4,10} + 0 + 3 + 3 + 1 = 1,
\]
\[
R(3) = R(3) + B_{3,5} + B_{3,7} = 0 + 1 + 12 + 2 + 3 + 3 = 25,
\]
\[
R(4) = R(4) + B_{4,2} + B_{4,4} + 0 + 55 + 55 + 12 = 122.
\]

When \( i = 2 \), since \( R(2) = 1 \) is odd, the sons of \( x \), are arranged in a down fragment. Since \( x \) is not the second largest element in the fragment, we have the following updates:

\[
R(3) = R(3) + B_{3,7} = 3 + 4 + 1 = 6;
\]
\[
R(4) = R(4) + B_{4,3} + B_{4,7} = 12 + 18 + 3 = 27.
\]

When \( i = 3 \), since \( R(3) = 6 \) is even, the sons of \( x \), are arranged in an up fragment. Since \( x \) is not the largest element in the fragment, we obtain

\[
R(4) = R(4) + B_{4,5} + B_{4,7} + B_{4,9} + 27 + 7 + 4 + 1 = 31;
\]

Finally, the algorithm outputs \( R(4) = 31 \).

Obviously, building the accumulation table using Eq. (3) requires \( O(kn) \) time and space. In what follows, we will show that every element of the accumulation table in the ranking algorithm can be accessed in a constant time provided we make a preprocessing in advance. The basic idea is that all accessed elements in the accumulation table always occur in a certain relation when every time we update \( R(j) \) in the ranking algorithm, e.g. three elements for an up fragment and two elements for a down fragment. Thus, we can easily calculate these elements by using Eq. (3) and obtain the following formula:

\[
B_{i-1,z-k} = \frac{B_{i,z}}{m + 1} \cdot \prod_{j=0}^{k-1} m(k - 1) + b(i, z) + j
\]

where \( m = n - i + 1 \) and \( b(i, z) = (k(i - 1) + 1) - z + 1 \).

Eq. (6) states the relation between two elements \( B_{i-1,k} \) and \( B_{i,z} \) in the table. We observe that the numerator (respectively, the denominator) of a fractional number in the right hand side includes the product of \( k \) (respectively, \( k - 1 \)) consecutive integers. Let \( x = mk + b(i, z) \) and \( y = m(k - 1) + b(i, z) + 1 \). Also, define \( f(x) = x(x+1)(x+2) \ldots (x+k-1) \) and \( g(y) = y(y+1)(y+2) \ldots (y+k-2) \). Then, Eq. (6) can be reformulated as follows:

\[
B_{i-1,z-k} = B_{i,z} \cdot \frac{f(x)}{g(y)}
\]

where \( m = n - i + 1 \). To obtain some requisite terms in the accumulation table from Eq. (7), we now build a table of size \( 2x[n-2]+k+1 \) to store the terms \( f(t) \) and \( g(t) \) for \( k+1 \leq t \leq (n-1)k+1 \), where the first term \( z = k+1 \) is due to the initial conditions \( m = 1 \) (i.e., \( n = m \)), \( b(n, k(n-1)+1) = 1 \) and \( j = 0 \).

Clearly, \( f(t+1) = f(t) \cdot \frac{t+k}{t} \) and \( g(t+1) = g(t) \cdot \frac{t+k-1}{t} \). Thus, the table can be constructed in \( O(k^2) \) time. Since every element in the last row of the accumulation table is easy to obtain (i.e., \( b(n, z) = (k(n-1)+1)z - 1 \)), we can compute all requisite terms by using this preprocessing table.

For example, for \( k = 3 \) and \( n = 4 \), the desired preprocessing table is shown below:

<table>
<thead>
<tr>
<th>( z )</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(z) )</td>
<td>120</td>
<td>210</td>
<td>336</td>
<td>504</td>
<td>720</td>
<td>990</td>
<td>1320</td>
</tr>
<tr>
<td>( g(z) )</td>
<td>20</td>
<td>30</td>
<td>42</td>
<td>56</td>
<td>72</td>
<td>90</td>
<td>110</td>
</tr>
</tbody>
</table>

We can check that \( \prod_{j=0}^{k-1} m(k + b(i, z) + j) + 4 \cdot 5 \cdot 6 = 120 \) and \( \prod_{j=0}^{k-1} m(k + 1) + b(i, z) + j = 4 \cdot 5 \cdot 6 = 20 \) when \( m = 1 \) and \( b(i, z) = 1 \).

In the following, we show how to use the preprocessing table to compute the rank of a 3-ary tree \( T \) with \( z(T) = (1, 2, 3, 7) \). We have known \( B_{3,5} = B_{4,8} = 4 \cdot 8 \cdot 3 = 336 \) which is an element in the last row. Since \( i = 4 \) (i.e., \( m = 1 \) and \( z = 8 \) in this stage, we obtain \( x = mk + b(i, z) = 3 + 3 + 6 = 12 \) and \( y = m(k - 1) + b(i, z) + 1 = 2 + 3 + 1 = 6 \). Thus, using the terms as indices to search for the preprocessing table, we can compute \( B_{3,5} \) as follows:

\[
B_{3,5} = \frac{B_{4,8}}{m + 1} \cdot \frac{f(6)}{g(6)} = \frac{3}{2} \cdot \frac{336}{42} = 12.
\]

Next, since the current stage has changed \( i \) to \( 3 \) (i.e., \( m = 2 \)), the terms \( x \) and \( y \) can be obtained from their previous values by adding an extra offset \( k \) and \( k-1 \), respectively. Thus, the two indices are \( x = 6 + k = 9 \) and \( y = 6 + (k - 1) = 8 \), respectively. We now compute \( B_{2,2} \) as follows:

\[
B_{2,2} = \frac{B_{3,2}}{m + 1} \cdot \frac{f(9)}{g(8)} = \frac{12}{3} \cdot \frac{990}{72} = 55.
\]
We repeat such a process until all required elements are calculated. Note that, using Eq. (7), every required element can be computed in constant time. Since building the preprocessing table needs $O(kn)$ time and space and the number of accessed elements in the accumulation table is at most $O(n)$, we conclude the following.

**Theorem 2:** Determining the rank of a $k$-ary tree encoded by $z$-sequence with $n$ internal nodes in a Gray-code order can be done in $O(\max\{n, kn\})$ time and $O(kn)$ space.

### IV. CONCLUSION

In this paper, based on the Gray-code order of $k$-ary trees encoded by $z$-sequences [10], [19], we present an efficient ranking algorithm that improves the one introduced by Ahmadi-Adl et al. [2]. The time complexity and space requirement in our algorithm are $O(\max\{n, kn\})$ and $O(kn)$, respectively. As a future work, we will design an efficient unranking algorithm.

### REFERENCES


