International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:6, No:8, 2012

Ranking and Unranking Algorithms for k-ary Trees in Gray Code Order

Fateme Ashari-Ghomi, Najme Khorasani, and Abbas Nowzari-Dalini

Abstract—In this paper, we present two new ranking and unranking algorithms for k-ary trees represented by x-sequences in Gray code order. These algorithms are based on a gray code generation algorithm developed by Ahrabian et al.. In mentioned paper, a recursive backtracking generation algorithm for x-sequences corresponding to k-ary trees in Gray code was presented. This generation algorithm is based on Vajnovszki's algorithm for generating binary trees in Gray code ordering. Up to our knowledge no ranking and unranking algorithms were given for x-sequences in this ordering. we present ranking and unranking algorithms with $O(kn^2)$ time complexity for x-sequences in this Gray code ordering .

Keywords—k-ary Tree Generation, Ranking, Unranking, Gray Code.

I. Introduction

TRees are one of the most important structures in computer science. Trees have many applications such as database generation, decision table programming, analysis algorithms, string matching [11], switching theory, and even in the theoretical VLSI circuit design [26]. Trees are also widely used data structures for maintaining data [19] and as auxiliary structures for compressing data [10]. In addition, the exhaustive generation of all trees of a certain type is often useful; for example, a list of all trees with a given number of nodes n, may be used to test and analyze of algorithm complexity, and prove the correctness of an algorithm [11]. As such, the problem of generating binary trees [1], [4], [17], [21], [22], [30], k-ary trees [2], [5], [6], [7], [8], [13], [18], [23], [28], and other types of trees[20], [25], has been thoroughly investigated and many papers have been published in the literature for generating trees.

In most of these algorithms, trees are encoded as integer sequences and then these sequences are generated with a certain order, and consequently their corresponding trees are also generated in a specific order. The most well-known orderings on trees are A-order and B-order [22], [33], and the orderings on the sequences are lexicographical [33] and Gray code [15], [17]. Typically, an important issue for algorithms to enumerate various classes of objects is that they must be run in a constant time for each generation. Therefore so many algorithms have been proposed to generate k-ary trees in constant time in the worst case. These algorithms generate the sequences corresponding to trees in Gray code order [14], [15], [16], [24], [32].

Fateme Ashari-Ghomi, Najme Khorasani and Abbas Nowzari-Dalini are with the Department of computer Science, School of Mathematics, Statistics and Computer Science, University of Tehran, Tehran, Iran. e-mail addresses: (ftm.ashari@ut.ac.ir, n.khorasani@ut.ac.ir, nowzari@ut.ac.ir)

Beside the generation algorithm for k-ary trees, ranking and unranking algorithms are also important in the concept of generation [23], [28], [31], [33]. Given a specific order on the set of k-ary trees, the rank of a tree (or corresponding sequence) is its position in the exhaustive generated list, and the ranking algorithm computes the rank of a given tree (or corresponding sequence) in this order. The reverse operation of the ranking is called unranking, which generates the tree (or sequence) corresponding to a given rank. Usually tree generation papers present ranking and unranking algorithms for tree generation algorithm. Ranking and unranking algorithms also have many applications. For example, in traditional tree compression algorithm for encoding the tree to code sequence and decoding the code sequence back to a tree, the ranking and unranking algorithms can be used.

As mentioned, in tree generation algorithms, trees are represented as sequences and then these sequences are generated. Two well-known representations of k-ary trees with n nodes are x-sequences (or 0-1 sequences) and z-sequences presented by Zaks [33]. He presented a generation algorithm for z-sequences encoding in lexicographical ordering such that the corresponding trees are generated in B-order, and generates one sequence from a given sequence in $O(n)$ (n is the length of z-sequence). He also presented two ranking and unranking algorithms for this generation in $O(kn)$. Later, some other k-ary tree generation algorithms were presented on this encoding [7], [24], [32].

Vajonvszki presented a new Gray code for the x-sequence corresponding to binary trees with n nodes [27]. A constant average time algorithm for generating this Gray code is also given. Ahrabian et al. introduced another algorithm for generating k-ary trees represented by x-sequence in Gray code order [3], which is based on Vajnovszki's paper [27]. This algorithm is a backtracking generation algorithm. Up to now, no ranking and unranking algorithms are presented for x-sequences in Gray code ordering given by Vajonvszki [27] and Ahrabian et al. [3].

In this paper, we present ranking and unranking algorithms for x-sequences corresponding to k-ary trees with n nodes generated in Gray code based on the Ahrabian et al. [3] generation algorithm. The time complexity of ranking algorithm is $O(kn^2)$ and unranking algorithm is also $O(kn^2)$.

The rest of this paper is organized as follows. Section II introduces the definitions and notations that are used subsequently. In Section III, ranking and unranking algorithms are discussed. Finally, some concluding remarks are offered in Section IV.

II. Preliminaries

In graph theory, a tree is a connected and undirected graph without any cycle. A rooted tree is a tree with a distinct node which is called root. Rooted trees are directed trees and the direction is from the root to the offspring. A rooted tree where the children of each node have a designated order is called ordered tree. A k-regular ordered tree is an ordered tree in which each node has either k children (internal node) or no children (external node or leaf), and is called k-ary tree. An n-node k-ary tree has n internal nodes and $(k-1)n+1$ external nodes or leaves, and the total number of k-ary trees with n internal nodes is denoted by $C_{n,k}$ and is known to have the value

$$C_{n,k} = \frac{1}{(k-1)n+1}\binom{kn}{n}.$$

A k-ary tree T can also be defined recursively as being either an external (leaf) or an internal node together with a sequence $T_1, T_2, \cdots, T_k$ of k-ary trees, which T_i is defined as the ith subtree of T [11].

Let us introduce basic notations used through this paper and a definition of k-ary trees by means of choice functions of indexed families of sets (from [9], [12], [13]).

Let $\Gamma = \{1, \cdots, m\}$ and $I = \{1, \cdots, \ell\}$ be two sets, and $\ell, m \geq 1$, then $< \Gamma_i >_{i\in I}$ where $\Gamma_i = \Gamma$, show an *family of sets indexed by* I. Any mapping function f which chooses one element from each set $\Gamma_1, \cdots, \Gamma_\ell$ is called a *choice function* of the indexed family $< \Gamma_i >_{i\in I}$. With supplementary restrictions, various classes of combinatorial objects can be modeled by choice functions [9], [12], [13].

If $I = \{1, \cdots, \ell\}$ and $\Gamma_i = \{0, 1\}$, then any choice function $\chi =< x_i >_{i\in I}$, that belongs to the indexed family $< \Gamma_i >_{i\in I}$, is called *binary choice function* of this family [12]. If $\ell \leq kn$ for a given k and n, each binary choice function which $x_1 + \cdots + x_i \geq i/k$, for $1 \leq i \leq kn$, is called binary choice function with k-dominating properties. There exist bijections between set of choice functions χ and sets of k-ary trees with n internal nodes in widely used representations. All k-dominating binary choice functions, with $\ell = kn$ and the number of $x_1 + \cdots + x_i = n$, are bitstring representations of all k-ary trees of the set Γ. This bitstring representation is called x-sequence [33]. By k-dominating definition, in each subsequence $\{x_j\}_1^i (1 \leq i \leq kn)$ the accumulated number of 1's is at least $\lceil i/k \rceil$. Clearly, each sequence has n 1's and $(k-1)n$ 0's.

For any given choice functions $\delta =< d_1, \cdots, d_l >$ and $\gamma =< g_1, \cdots, g_l >$, we say that δ and γ are in Gray code order, if they differ by a constant number of changes [29].

The x-sequence can be obtained directly from k-ary trees. Given a regular k-ary tree with n internal nodes, we label each internal node with 1 and each external node with 0. Reading tree labels in pre-order (recursively, visit first root and then all the subtrees from left to right), we get a bitstring with n 1's and $(k-1)n+1$ 0's. As the last visited node is an external node, we omit the corresponding 0 [33]. For example, the x-sequence corresponding to the tree presented in Fig. 1 is $\mathbf{x} = 110000100100$.

Now, let $\mathbf{S}_k^{m,n}$ be a set of all k-dominating binary choice functions with the number of $x_1 + \cdots + x_i = n$ and $m = \ell - n$, in other words the set of all bitstrings with a k-dominating property with m 0's and n 1's such that $m \geq (k-1)n \geq 0$. In particular case where $m = (k-1)n$, $\mathbf{S}_k^{m,n}$ is the set of all x-sequences of length kn. As it is mentioned, we call a list of bitstrings in $\mathbf{S}_k^{m,n}$ Gray code list if every two successive bitstrings in the list differ by an interchange of two bits. Also, let α and β be two bitstrings then we denote $\alpha\beta$ the concatenation of α and β. By these definitions, the following lemma shows the recursion property of the set $\mathbf{S}_k^{m,n}$, which helps us in the generation schema [3].

Lemma 1. *For any $k > 0$, and $m, n \geq 0$, we have*

$$\mathbf{S}_k^{m,n} = \begin{cases} 0^m & \text{if } n = 0, \\ 1\mathbf{S}_k^{m,n-1} & \text{if } m = (k-1)n, \\ 0\mathbf{S}_k^{m-1,n} \cup 1\mathbf{S}_k^{m,n-1} & \text{if } m > (k-1)n > 0. \end{cases}$$

For example, Table I shows all bitstrings $\mathbf{S}_3^{6,2} = 0\mathbf{S}_3^{5,2} \cup 1\mathbf{S}_3^{6,1}$. With regard to the recursion property of the set $\mathbf{S}_k^{m,n}$, we can easily count the number of elements in this set. Let $S_k^{m,n}$ be the number of bitstrings in $\mathbf{S}_k^{m,n}$. According to the above lemma, $S_k^{m,n}$ can be obtained as follows.

Corollary 1. *For any $k > 0$, and $m, n \geq 0$, we have*

$$S_k^{m,n} = \begin{cases} 1 & \text{if } n = 0, \\ S_k^{m,n-1} & \text{if } m = (k-1)n, \\ S_k^{m-1,n} + S_k^{m,n-1} & \text{if } m > (k-1)n > 0. \end{cases}$$

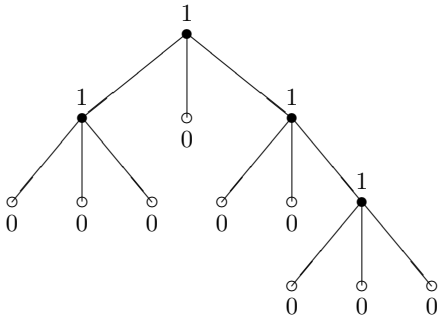

Fig. 1. A 3-ary tree with 4 internal nodes.

TABLE I
All bitstrings in $\mathbf{S}_3^{6,2} = 0\mathbf{S}_3^{5,2} \cup 1\mathbf{S}_3^{6,1}$.

$0\mathbf{S}_3^{5,2}$	$1\mathbf{S}_3^{6,1}$
00100100	11000000
00101000	10100000
00110000	10010000
01100000	10001000
01010000	10000100
01001000	
01000100	

Corollary 2. *For any $k > 0$, $n \geq 0$, and $m \geq (k-1)n \geq 0$, we have*

$$S_k^{m,n} = \frac{m+1-(k-1)n}{m+1}\binom{m+n}{n}.$$

With regard to the definition of $\mathbf{S}_k^{m,n}$, it should noted that $\mathbf{S}_k^{(k-1)n,n}$ is the set of bitstrings corresponding to k-ary trees with n internal nodes. Therefore, we can have the following theorem.

Theorem 1. *In particular case of Corollary 2 where $m = (k-1)n$,*

$$S_k^{(k-1)n,n} = \frac{1}{(k-1)n+1}\binom{kn}{n}.$$

Obviously $S_k^{(k-1)n,n} = C_{n,k}$ (the number of all k-ary trees with n nodes). With regard to the above discussion and using Lemma 1 the generation k-ary trees algorithm in Gray code order was presented in [3]. As an example, a list of 22 x-sequences corresponding to 4-ary trees with $n = 3$ nodes in this Gray code order is shown in Table II.

III. Ranking and Unranking Algorithm

The rank of a k-ary tree with respect to some ordering is the number of previously generated trees in that ordering. The reverse operation of the ranking is unranking. As mentioned, for the x-sequences in Gray code, no ranking and unranking algorithms are presented in the literature in Gray code ordering given by Vajonvszki [27] and Ahrabian et al. [3]. In this section, the ranking and unranking algorithms for these generation and ordering are presented.

For ranking and unranking algorithms we need $S_k^{m,n}$ for given k, m, and n, that are defined and computed earlier. We assume that these values are computed and stored in arrays $S[m][n]$ for a fixed k. Also, for computing the rank of a given bitstring or unranking a given rank, we need three other variables *prePos*, *curPos*, and *dir*. In the ranking and unranking algorithm, when the ith 1 of a bitstring is investigated, the variable *curPos* holds the position of this 1 (ith 1) and the variable *prePos* holds the position of previous 1 ($(i-1)$th 1).

The variable *dir* holds the direction of the ith 1 in a bitstring. Direction of the ith 1 in a bitstring is *down* (represented by 0) if the ith 1 has been shifted one place to the left in the next bitstring in Gray code order, and if the ith 1 has been shifted one place to the right, the direction is *up* (represented by 1). For example, consider two bitstrings '00100100' and '00101000', for the first bitstring the direction of the second 1 is *down*, because in the next bitstring, the second 1 has been shifted to left. But for two bitstrings '01100000' and '01010000', the direction of second 1 in the first bitstring is *up*, because this 1 has been shifted to right. Since the first bit in the input bistring is always equal to 1 and does not shift, we can assume that the direction of the first bit is equal to 0.

To determine the direction of the ith 1 for $2 \leq i \leq n$, we can use the following relation:

If direction of the $(i-1)$th 1 is equal to 0 then
direction of the ith 1 is equal to

$$\begin{cases} 0 & \text{if } (i-1)\text{th 1 is in its last position,} \\ 1 & \text{if } (i-1)\text{th 1 is not in its last position.} \end{cases}$$

If direction of the $(i-1)$th 1 is equal to 1 then
direction of the ith 1 is equal to

$$\begin{cases} 1 & \text{if } (i-1)\text{th 1 is in its last position,} \\ 0 & \text{if } (i-1)\text{th 1 is not in its last position.} \end{cases}$$

In this relation, if the position of the ith 1 is equal to $(i-1)k+1$, then it is in the last position.

As mentioned, a x-sequence corresponding to a k-ary tree with n nodes, has n 1's and $m = (k-1)n$ 0's. In the Gray code order, for generating a bitsring from the previous bitsring, each 1 should be shifted from its position to next position. Therefore, each 1 has a shift direction and can be computed using the above relation. If the position of the ith 1 in the jth bitsring is p and its direction is *up* (dir is equal to 1) and this 1 is a candidate for shifting, then its position in the $(j+1)$th bitsring is $p+1$. Otherwise, if the direction of the ith 1 in the jth bitsring is *down* then its position in the $(j+1)$th bitsring is $p-1$. For a given bitsring, ith 1 have a last and a first position. The last position of the ith 1 is shown by p_{last} and is equal to $(i-1)k+1$ and the first position of it is shown by p_{first}, and is equal to the position of the $(i-1)$th 1 plus 1. A 1 is shiftable if it is not in its last or first position, in other words, a 1 is shiftable if its shift direction is *up* and it is not in its last position, or if its shift direction is *down* and it is not in its first position.

In order to compute the rank of a bitsring, we count the number of generated bitsring before this bitsring in the Gray code order. This can be done, using by the shiftable 1's. For this purpose, the bitsring should be scaned from left to right and a shiftable 1 should be found. As mentioned, the first 1 is never shifted and is always in position 1, and shift direction of ith 1 is determined by shift direction of $(i-1)$th 1. Assume that we investigate the ith 1 in the jth bitsring, we have two cases which are explained as follows.

In the first case, the shift direction of the ith 1 in the jth bitsring is equal to *down* ($dir = 0$). So, the first possible position is p_{last} for this 1, where is equal to $(i-1)k+1$, and the last possible position is p_{first} for this 1, where is equal to the position of the $(i-1)$th 1 plus 1. In all the consecutive bitsrings that are generated by shifting of the ith 1, the possible positions for this 1 can be the positions p_{last}, $p_{last}-1$, $p_{last}-2$, $\cdots$, p_{first}. Therefore, if the position of the ith 1 is p then the positions of this 1 in previous bitsrings are p_{last}, $p_{last}-1$, $\cdots$, $p+2$, $p+1$. The number of such bitsrings should be obtained for ranking. At first, we count the number of bitsrings which the ith 1 of them is in position p_{last}. The positions of the first 1 to $(i-1)$th 1 are fixed and we have to count the number of 0's and 1's which their positions are after the position of ith 1. The number of 1's that their positions are after the ith 1 in the jth bitsring is equal to $n'_j = n - i$. Now

TABLE II
THE 22 x-SEQUENCES OF LENGTH 12 FOR $n = 3$ AND $k = 4$.

$rank$	$bitstring$	$rank$	$bitstring$
1	100010001000	12	101001000000
2	100010010000	13	101000100000
3	100010100000	14	101000010000
4	100011000000	15	101000001000
5	100110000000	16	111000000000
6	100101000000	17	110100000000
7	100100100000	18	110010000000
8	100100010000	19	110001000000
9	100100001000	28	110000100000
10	101100000000	21	110000010000
11	101010000000	22	110000001000

let q be the number of 0's which their positions are before the ith 1 in the jth bitsring, so the number of 0's which their positions are after p is equal to $m'_j = m - q$ (it is obvious that $q = p - i$). Therefore, the number of bitsrings that the ith 1 of them is in position p_{last} is equal to $S_k^{m'_j,n'_j}$. With the same way, we can count the number of bitsrings which the ith 1 of them are in the positions $p_{last} - 1, \cdots, p + 2, p + 1$. Finally, the number of previous bitsrings before bitsring j that the ith 1 of them are in positions $p_{last}, p_{last} - 1, \cdots, p + 2, p + 1$ is equal to:

$$\sum_{j=p+1}^{p_{last}} S_k^{m'_j,n'_j}.$$

In the second case, the shift direction of the ith 1 in the jth bitsring is equal to *up* ($dir = 1$). So the first possible position for this 1 is p_{first}. In all the consecutive bitsrings which are generated by shifting of the ith 1, the possible positions for this 1 can be $p_{first}, p_{first}+1, p_{first}+2, \cdots, p_{last}$. Therefore, if the position of the ith 1 is p then the positions of this 1 in previous bitsrings are $p_{first}, p_{first} + 1, \cdots, p - 2, p - 1$. The number of such bitsrings should be obtained for ranking. This number can be computed similar to the first case and we have:

$$\sum_{j=p_{first}}^{p-1} S_k^{m'_j,n'_j},$$

where the values of m'_j and n'_j are equal to $m - q$ and $n - i$, respectively (similar to the first case).

Now, the rank of a given bitsring can be computed as:

$$\sum_{i=2}^{n} \sum_{j=p_\ell}^{p_u} S_k^{m'_j,n'_j},$$

where p_ℓ and p_u are lower and upper bound for positions of the ith 1.

With regards to the above discussion, the ranking algorithm is given in Algorithm 1. This algorithm takes n, k, and bitstring x corresponding to the k-ary tree with n nodes as input parameters, and returns the rank of x in variable r. Considering the above discussion, the time complexity of the ranking algorithm is $O(kn^2)$.

The unranking algorithm essentially reverses the steps carried out in computing the rank. The unranking algorithm given in Algorithm 2 takes r, k, and n as input, and returns the x-sequence corresponding to the k-ary tree with n nodes that has rank r.

For a given r as a rank, the corresponding bitstring $x = x_1 \cdots x_{kn}$ is computed by the following operations. Initially x_1 is set to 1 and all x_i's ($2 \leq i \leq kn$) are set to zero, the position of the first 1 is set to 1, and the direction of this 1 is set to 0. The bitstring x has to have n 1's, therefore the correct position of the ith 1 ($2 \leq i \leq n$) is computed in a loop. In the iteration i ($2 \leq i \leq n$) of this loop, at first, the direction of the ith 1 (dir) is computed using by the direction and position of the $(i-1)$th 1, after that, the position of the ith 1 ($curPos$) is computed using $s[n*(k-1)-curPos+i][n-i]$. The values of $s[n*(k-1)-curPos+i][n-i]$ determine the possible number of shifts which ith 1 can have. Finally, the value of this position is set to 1 in bitstring x. Considering the above discussion, the time complexity of the unranking algorithm is $O(kn^2)$.

IV. CONCLUSION

we have introduced two new ranking and unranking algorithms for n-node k-ary trees represented by x-sequences. In these algorithms, it is assumed that the order of x-sequences corresponding to the list of all k-ary trees with n nodes is Gray code order. The time complexity of both ranking and unranking algorithms is $O(kn^2)$.

REFERENCES

[1] H. Ahrabian and A. Nowzari-Dalini, On the generation of binary trees from (0-1) codes, *Intern. J. Comput. Math.* **69** (1998), 243–251.
[2] H. Ahrabian and A. Nowzari-Dalini, Generation of t-ary trees with Ballot sequences, *Intern. J. Comput. Math.* **80** (2003), 1243–1249.
[3] H. Ahrabian and A. Nowzari-Dalini, Gray code algorithm for listing k-ary trees, *Studies in Information and Control* **13** (2004), 243–251.
[4] H. Ahrabian and A. Nowzari-Dalini, Parallel generation of binary trees in A-order, *Parallel Comput.* **31** (2005), 948–955.
[5] H. Ahrabian and A. Nowzari-Dalini, Parallel Generation of t-ary trees in A-order, *Comput. J.* **50** (2007), 581–588.
[6] A. Ahmadi-Adl, A. Nowzari-Dalini, and H. Ahrabian, Ranking and unranking algorithms for loopless generation of t-ary trees, *Logic Journal of IGPL* **19** (2011), 33–43.
[7] M.C. Er, Efficient generation of k-ary trees in natural order, *Comput. J.* **35** (1992), 306–308.

Algorithm 1 The ranking algorithm.

```
Procedure Ranking(x : bitstring; n, k : integer)
begin
  r = 1 ;
  curPos = 1 ; dir = 0 ;
  for i = 2 to n do begin
    prePos = curPos;
    curPos = curPos + 1;
    while (x[curPos]! = 1)
      curPos = curPos + 1;
    if ( ( (dir == 0) && (prePos == (i − 2) ∗ k + 1) ) ||
         ( (dir == 1) && (prePos! = (i − 2) ∗ k + 1) ) ) then begin
       dir = 0;
       for j = (i − 1) ∗ k + 1 downto curPos + 1 do
         r = r + s[(k − 1) ∗ n − j + i][n − i];
    end
    else begin
         dir = 1;
         for j = prePos + 1 to curPos − 1 do
           r = r + s[(k − 1) ∗ n − j + i][n − i];
    end
  end
  return r ;
end
```

Algorithm 2 The unranking algorithm.

```
Procedure UnRanking(r, n, k : integer)
begin
  x[1] = 1;
  for i = 2 to kn do
    x[i] = 0;
  prePos = 1; dir = 0;
  for i = 2 to n do begin
     if ( ( (dir == 0) && (prePos == (i − 2) ∗ k + 1) ) ||
          ( (dir == 1) && (prePos! = (i − 2) ∗ k + 1) ) ) then begin
          dir = 0;
          curPos = (i − 1) ∗ k + 1;
          while r > s[n ∗ (k − 1) − curPos + i][n − i] do begin
            r = r − s[n ∗ (k − 1) − curPos + i][n − i];
            curPos = curPos − 1;
          end
     end
     else begin
          dir = 1;
          curPos = prePos + 1;
          while r > s[n ∗ (k − 1) − curPos + i][n − i] do begin
            r = r − s[n ∗ (k − 1) − curPos + i][n − i];
            curPos = curPos + 1;
          end
     end
     x[curPos] = 1;
     prePos = curPos;
  end
  return x ;
end
```

[8] S. Heubach, N. Li, and T. Mansour, Staircase tilings and k-Catalan structures, *Discrete Math.* **308** (2008), 5954–5964.
[9] A. Kapralski, New methods for the generation of permutations, combinations and other combinatorial objects in parallel, *J. Parallel Distrib. Comput.* **17** (1993), 315–329.
[10] J. Katajainen and E. Makinen, Tree compression and optimization with application, *Inter. J. Found. Comput. Sci.* **1** (1990), 425–447.
[11] D.E. Knuth, *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*, Addision Wesly, Reading, 1973.
[12] Z. Kokosinski, On the generation of permutations through decomposition of symmetric group into cosets, *Bit* **30** (1990), 583–591.
[13] Z. Kokosinski, Parallel enumeration of $t$ary trees in ASC SIMD model, *International Journal of Computer Science and Network Security* **11** (2011), 38–49.
[14] J.F. Korsh, Loopless generation of k-ary tree sequences, *Inform. Process. Lett.* **52** (1994), 243-247.
[15] J.F. Korsh and P. LaFollette, Loopless generation of Gray codes for k-ary trees, *Inform. Process. Lett.* **70** (1999), 7–11.
[16] J. F. Korsh and P. LaFollette, A loopless Gray code for rooted trees, *ACM Transactions on Algorithms* **2** (2006), 135-152.
[17] J. Lucas, D. Roelants van Baronaigien, and F. Ruskey, On rotations and the generation of binary trees, *J. of Algorithms* **15** (1993), 343–366.
[18] K. Manes, A. Sapounakis, I Tasoulas, and P. Tsikouras, Recursive generation of k-ary trees, *Journal of Integer Sequences* **12** (2009), 1–18.
[19] O. O. Olugbenga, E. F. Adebiyi, S. Fatumo and A. Dawodu, PQ trees, consecutive ones problem and applications, *International Journal of Natural and Applied Sciences* **4** (2008), 262–277.
[20] J. M. Pallo, A simple algorithm for generating neuronal dendritic trees, *Comput. Meth. Prog. Bio.* **33** (1990), 165–169.
[21] J. M. Pallo, Rotational tree structures on binary trees and triangulations, *Acta Cybernetica* **17** (2006), 799–810.
[22] J. M. Pallo and R. Racca, A note on generating binary trees in A-order and B-order, *Intern. J. Comput. Math.* **18** (1985), 27–39.
[23] F. Ruskey, Generating t-ary trees lexicographically, *SIAM J. Comput.* **7** (1978), 424–439.
[24] D. Roelants Van Baronaigien, A loopless Gray code algorithm for listing k-ary trees, *J. Algorithms* **35** (2000), 100–107.
[25] E. Seyedi-Tabari, H. Ahrabian, and A. Nowzari-Dalini, A new algorithm for generation of different types of RNA, *Intern. J. Comput. Math.* **87** (2010), 1197–1207.
[26] T. Uehara and W.M. Cleemput, Optical layout of cmos functional arrays, *IEEE Trans. Comput.* **7** (1981), 305–312.
[27] V. Vajnovszki, Constant time algorithm for generating binary tree Gray codes, *Studies in Informatics and Control* **5** (1996), 15–21.
[28] V. Vajnovszki and J. M. Pallo, Ranking and unranking of k-ary trees with $4k$-4 letter alphabet, *Journal of Information and Optimization Science* **18** (1997), 271–279.
[29] V. Vajnovszki and R. Vernay, Restricted compositions and permutations: From old to new Gray codes, *Inform. Process. Lett.* **111** (2011), 650-655.
[30] R. Wu, J. Chang, and Y. Wang, A linear time algorithm for binary tree sequences transformation using left-arm and right-arm rotations, *Theor. Comput. Sci.* **335** (2006), .303–314.
[31] R. Wu, J. Chang, and C. Chang, Ranking and unranking of non-regular trees with a prescribed branching sequence, *Math. Comput. Model.*, **53** (2011), 1331–1335.
[32] L. Xiang, K. Ushijima, and C. Tang, Efficient loopless generation of Gray codes for k-ary trees, *Inform. Process. Lett.* **76** (2000), 169-174.
[33] S. Zaks, Lexicographic generation of ordered trees, *Theor. Comput. Sci.* **10** (1980), 63–82.