Gray Code Algorithm for Listing k-ary Trees*

H. Ahrabian, A. Nowzari-Dalini

Institute for Studies in Theoretical Physics and Mathematics (I.P.M.),

Teheran, Iran.

ahrabian@ut.ac.ir.

E. Salehi

Department of Mathematics and Computer Science, University of Tehran,

Teheran, Iran.

Abstract: In this paper an efficient algorithm for generating k-ary trees in Gray code order is presented. This algorithm is an extended form of the constant time algorithm developed by Vajnovszki to enumerate Gray codes for binary trees. In our algorithm the k-ary trees are represented by bitstring sequences. The generation algorithm is based on moving in a Hamiltonian path of a graph, which each vertex of it is labeled with a bitstring corresponding to a k-ary tree. Clearly, the Gray code property of the vertices construct the edges of this graph. The algorithm generates two successor codes in constant delay time.

Keywords: k-ary tree, Gray codes, Constant time generation algorithm.

Hayedeh Ahrabian is an associate professor of Department of Mathematics and Computer Science, University of Tehran and is currently the director of Graduate studies and director of computer science group in this department. Her research interests include combinatorial algorithm, parallel algorithms, DNA computing, and genetic algorithms.

Abbas Nowzari-Dalini received his Ph.D. in computer science from University of Tehran in 2004. He is currently an assistant professor in Department of Mathematics and Computer Science, University of Tehran. His research interests include combinatorial algorithm, parallel algorithms, DNA computing, neural networks, and computer networks.

Elham Salehi received her M. Sc. in computer science from University of Tehran in 2003 under supervision of Dr. H. Ahrabian.

1. Introduction

Trees are one of the most important data structures in computer science, so it is natural that their properties be studied in depth. Designing efficient algorithms to generate trees in one method of studying these objects. There have been many algorithms published for generating k-ary (including binary) trees [11, 14, 15, 17, 21, 23]. In most algorithms trees are encoded as integer sequences of which subsequent sequences are generated in a specified order. Few of these known sequences are Zaks' sequences, P-sequences, and Ballot sequences, which are generated in lexicographical order [1, 2, 14, 23]. Therefore, the differences in successively generated trees are not bounded by a constant. When the changes in successive sequences are constant, mostly one, it is well known that the sequences are in Gray code order (strongly Gray code is called for only one change). Korsh [8] gave an efficient generating algorithm for k-ary tree sequences using the rotation operation on trees. This algorithm is the generalization of Roelants algorithm for generating binary trees [16]. Using a shift operation between two k-ary trees, another algorithm is developed by Korsh and Lipshutz [10]. Korsh and LaFollette also developed an algorithm for generating k-ary trees Gray codes [9] and another algorithm discovered by Roelants [17], and Xiang, Ushijuma, and Tang [22] independently.

Vajonvszki presented a new Gray code for the Zaks' sequences for binary trees with n nodes [19]. A constant average time algorithm for generating this Gray code is also given. In this paper, another algorithm is presented for generating of k-ary trees represented by Zaks' sequences in Gray code order, which is based on Vajnovszki's paper [19]. This algorithm also moves on a Hamiltonian path of a graph, which each vertex of it is labeled with a Zaks' sequences corresponding to k-ary trees, and the edges are constructed with regard to the Gray code property of the codes. It should be mentioned that sequences corresponding to k-ary trees are generated in many different Gray code orders [8, 9, 10], but the presented algorithm in this paper generates sequences in a new different Gray code order.

Trees have many applications such as database, decision table programming, analysis algorithms [5], string matching [20], switching theory, and even in the theoretical design of circuits required for VLSI [18]. 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, analyze the complexity, prove the

* This research was in part supported by a grant from I.P.M. (No. CS1382-4-01).

correctness of an algorithm [5], or data compression in data communication [4]. Trees are also widely used structures for maintaining data and are used as auxiliary structures when compressing data. In traditional tree compression for both cases the only operations performed are encoding of a tree to code sequence and decoding the code sequence back to a tree. Using ranking algorithm, the code sequence can be represented as an integer which is a compressed form of it. This code sequence can be returned from its rank by unranking algorithm [13]. Obviously any ranking and unranking algorithm on trees are designed based on a specific generation algorithm. In generating trees a desirable goal is that the running time required for producing the representation of trees. Our generation algorithm presented in this paper produces two consecutive trees in a constant time with Gray code property.

The remaining of this papers is organized a follows. In Section 2, some notations and definitions used further in this paper are presented. Construction of our Gray code is discussed in Section 3. In Section 4, we present our generation algorithm. Finally, conclusion is stated in Section 5.

2. Preliminaries

A regular *k*-ary tree is a rooted, ordered and oriented tree in which every internal node has exactly k children. A *k*-ary tree T can also be defined recursively as being either an external (leaf) or an internal node together with a sequence $T_1, T_2, \mathrm{K}, T_k$ of *k*-ary trees, which T_i is defined as the *i*th subtree of T [5]. An *n*-node *k*-ary tree has $(k-1)n+1$ 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 $C_{n,k} = \frac{1}{(k-1)n+1}\binom{kn}{n}$.

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 [3, 7, 12].

Let $<A_i>_{i\in I}$ denote an *indexed family of sets* $A_i = A$, where $A = \{1, \mathrm{K}, m\}, I = \{1, \mathrm{K}, l\}$, and $l, m \geq 1$. Any mapping f which chooses one element from each set A_1, K, A_l is called a *choice function* of the family $<A_i>_{i\in I}$ [12]. With additional restrictions we can model by choice functions various classes of combinatorial objects [3,6].

If $A_i = \{0,1\}$ and $I = \{1, \mathrm{K}, l\}$, then any choice function $\chi = <x_i>_{i\in I}$, that belong to the indexed family $<A_i>_{i\in I}$, is called *binary choice function* of this family. If $l \leq kn$ for a given *k*, each binary choice function with the number of $x_1 + \mathrm{L} + x_i \geq i/k$, 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 $l = kn$ and the number of $x_1 + \mathrm{L} + x_i = n$, are bitstring representations of all *k*-ary trees of the set A [7]. This bitstring representation is called *x*-sequence and also known as Zaks' sequence [23]. By *k*-dominating definition, in each subsequence $\{x_j\}_1^i (1 \leq i \leq kn)$ the accumulated numbers 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 $\delta = <d_1, \mathrm{K}, d_l>$ and $\gamma = <g_1, \mathrm{K}, g_l>$, we say that δ and γ are in Gray code order, if they differ by a constant number of changes.

The *x*-sequence can be obtained directory 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. For example, the *x*-sequence corresponding to the tree presented in Figure 1 is x = 110000100100.

Now, let $\mathbf{S}_k^{m,n}$ be a set of all *k*-dominating binary choice function with the number of $x_1 + \mathrm{L} + x_i = n$ and $m = l - n$, in other words the sets 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.

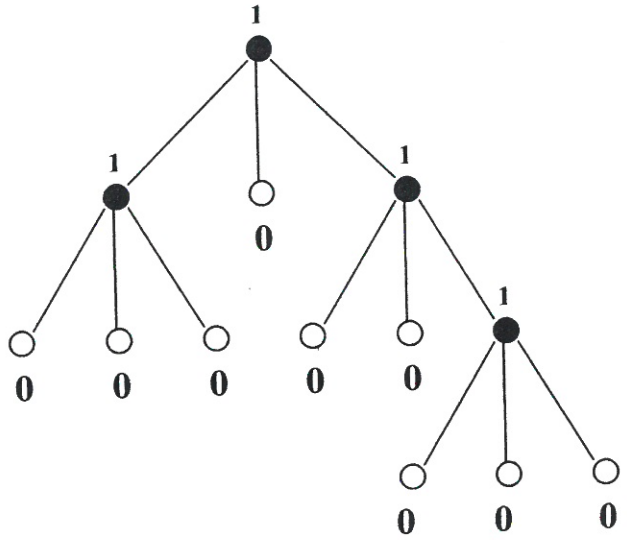

Figure 1: A 3-ary Trees With 4 Internal Nodes.

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, Table1 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 $|\mathbf{S}_k^{m,n}|$ be the number of bitstrings in $\mathbf{S}_k^{m,n}$. According to the above lemma, $|\mathbf{S}_k^{m,n}|$ can be obtained as follows.

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

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

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

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

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

$$|\mathbf{S}_k^{m,n}| = \frac{m+1-(k-1)n}{m+1}\binom{m+n}{n}.$$

With regard to the definition of $\mathbf{S}_k^{m,n}$, it should be 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 $m = (k-1)n$,

$$|\mathbf{S}_k^{(k-1)n,n}| = \frac{1}{(k-1)n+1}\binom{kn}{n}.$$

Obviously $|\mathbf{S}_k^{(k-1)n,n}| = C_{n,k}$ (the number of all k-ary trees with n nodes). This shows that we can use the Lemma 1 for generating k-ary trees.

3. Gray Code Interchange Operations

In this section, we consecutively prove that the set of sequences in $\mathbf{S}_k^{m,n}$ has a Gray code order and this property helps us to design our generation algorithm for generating k-ary trees in the next section.

For the bitstring x in $\mathbf{S}_k^{m,n}$ we define three operations *A, B, C* and their inverse A^{-1}, B^{-1}, C^{-1} which are the extended form of operations given by Vajnovszki for binary trees in [19] as is shown in Table 2. The operation A (therefore A^{-1}) is simply right (left) shift of the rightmost one, or these operations interchange two adjacent positions in the bitstring, and B and C (therefore B^{-1} and C^{-1}) interchange two non-adjacent positions.

Let $\mathbf{G}_k^{m,n}$ be the graph with $|\mathbf{S}_k^{m,n}|$ vertices and each vertex is labeled with one bitstring from the set $\mathbf{S}_k^{m,n}$. Two vertices are connected if and only if corresponding bitstrings can be obtained from each other by applying either operations *A, B, C* and inverse of them. It should be noted that the number of edges in this graph can be more that the number of vertices $|\mathbf{S}_k^{m,n}|$. In the remaining of this text, for any graph **G** and any arbitrary path H in **G** we use notations $x\mathbf{G}$ and xH which means elements $x \in \{0,1\}$ is connected to each bitstring label of **G** and H respectively. The existence of a Hamiltonian path in $\mathbf{G}_k^{m,n}$ shows that we can develop an algorithm which generates all bitstrings of $\mathbf{S}_k^{m,n}$ in a Gray code order. The following lemmas and theorem help us to see the methodology of obtaining the Hamiltonian path H in graph $\mathbf{G}_k^{m,n}$.

Lemma 2. If H is a Hamiltonian path in $\mathbf{G}_k^{m,n+1}$ with end-points x and y, then 0H is always a Hamiltonian path in $0\mathbf{G}_k^{m,n+1}$ with end-points 0x and 0y.

Operation	Source bitstring	Result bitstring	Conditions
A	$\alpha 0^a 10^b$	$\alpha 0^{a+1}10^{b-1}$	$a \geq 0, b \geq k$
A^{-1}	$\alpha 0^a 10^b$	$\alpha 0^{a-1}10^{b+1}$	$a \geq 1, b \geq k$
B	$\alpha 110^{2(k-1)+1}(10^{k-1})^a$	$\alpha 0110^{2(k-1)}(10^{k-1})^a$	$a \geq 0$
B^{-1}	$\alpha 0110^{2(k-1)}(10^{k-1})^a$	$\alpha 110^{2(k-1)+1}(10^{k-1})^a$	$a \geq 0$
C	$\alpha 110^{a+k}(10^{k-1})^b$	$\alpha 010^a(10^{k-1})^{b+1}$	$a \geq k, b \geq 1$
C^{-1}	$\alpha 010^a(10^{k-1})^{b+1}$	$\alpha 110^{a+k}(10^{k-1})^b$	$a \geq k, b \geq 1$

Table 2: The Gray Code Operations.

Lemma 3. *If H' is a Hamiltonian path in $\mathbf{G}_k^{m+1,n}$ with end-points x' and y', then $1H'$ is always a Hamiltonian path in $\mathbf{G}_k^{m+1,n}$ with end-points $1x'$ and $1y'$.*

Lemma 4. *If Lemma 2 and 3 are true and $0y$ is connected with $1y'$ then $0H \cup \{(0y, 1y')\} \cup 1H'$ is a Hamiltonian in $\mathbf{G}_k^{m+1,n+1}$ with end-points $0x$ and $1x'$.*

Lemma 5. If H is a Hamiltonian path in $\mathbf{G}_k^{(k-1)(n+1),n}$ with end-points x and y, then 1H is also a Hamiltonian path in $1\mathbf{G}_k^{(k-1)(n+1),n} = \mathbf{G}_k^{(k-1)(n+1),n+1}$ with end-points 1x and 1y.

Now, consider two special elements of $\mathbf{S}_k^{m,n}$ which are labels of two vertices in graph $\mathbf{G}_k^{m,n}$:

$$u_k^{m,n} = 10^{m-(k-1)(n-1)}(10^{k-1})^{n-1},$$

$$u_k^{m,n} = \begin{cases} 110^{2(k-1)}(10^{k-1})^{n-2} & \text{if } m = (k-1)n, \\ 0^{m-(k-1)n}(10^{k-1})^n & \text{if } m > (k-1)n. \end{cases}$$

With regard to the definitions of $u_k^{m,n}$, and $v_k^{m,n}$ we have the following lemma.

Lemma 6. For any k, m > 0, and $n \geq 0, u_k^{m,n}$ *and* $v_k^{m,n}$ have the following properties.

(i) $0u_k^{(k-1)n,n}$ *is equal to* $v_k^{(k-1)n+1,n}$,

(ii) $0v_k^{m,n+1}$ *is equal to* $v_k^{m+1,n+1}$,

(iii) $1u_k^{(k-1)(n+1),n}$ *is equal to* $v_k^{(k-1)(n+1),n+1}$,

(iv) $1v_k^{m+1,n}$ *is equal to* $u_k^{m+1,n+1}$,

(v) $0u_k^{m,n+1}$ *is obtained from* $1u_k^{m+1,n}$ *by applying operation C,*

(vi) $0v_k^{(k-1)n,n}$ *is obtained from* $1u_k^{(k-1)n+1,n-1}$ *by applying operation B.*

Now, the following theorem shows how to construct the Hamiltonian path $H_k^{m,n}$ with two end-points $u_k^{m,n}$ and $v_k^{m,n}$.

Theorem 2. *Graph $\mathbf{G}_k^{m,n}$ has a Hamiltonian path $H_k^{m,n}$ with two end-points $u_k^{m,n}$ and $v_k^{m,n}$.*

Proof. To prove this theorem we need to show how to construct $H_k^{m,n}$ with two end-points $u_k^{m,n}$ and $v_k^{m,n}$ such that moving from one vertex to the other is done by one of the operations *A, B, C,* or their inverse, and

also the vertices in $\mathbf{G}_k^{m,n}$ are all traversed. This is done by mathematical induction on increasing the values of m and n.

By the definition, $H_k^{m,1}$ for all $m \geq k$ is the path in below with length $|\mathbf{S}_k^{m,1}|$:

$$u_k^{m,1} = 10^m, 010^{m-1}, \text{K}, 0^{m-k+1}10^{k-1} = v_k^{m,1}.$$

This path is obtained by performing $m-k+1$ times operation A on $u_k^{m,1}$, and it is a Hamiltonian path in $\mathbf{G}_k^{m,1}$.

$H_k^{2(k-1),2}$ is also a Hamiltonian path with length $|\mathbf{S}_k^{2(k-1),2}|$:

$$u_k^{2(k-1),2} = 10^{k-1}10^{k-1}, 10^{k-2}10^k, \text{K}, 110^{2(k-1)} = v_k^{2(k-1),2}.$$

This path is also obtained by performing $k-1$ times operation A^{-1} on $u_k^{2(k-1),2}$, and it is also a Hamiltonian path in $\mathbf{G}_k^{1(k-1),2}$.

Now, we assume that $H_k^{m,n+1}$ is a Hamiltonian path for all $m \geq (k-1)(n+1) \geq 0$ in $\mathbf{G}_k^{m,n+1}$ and $H_k^{m+1,n}$ is also a Hamiltonian path for all $m \geq (k-1)n \geq 0$ in $\mathbf{G}_k^{m+1,n}$. We can construct $H_k^{m+1,n+1}$ as follows.

- According to Lemma 3, $1H_k^{m+1,n}$ is a Hamiltonian path in $\mathbf{1G}_k^{m+1,n}$ with end-points $1u_k^{m+1,n}$ and $1v_k^{m+1,n}$.
- According to Lemma 2, $0H_k^{m,n+1}$ is a Hamiltonian path in $0\mathbf{G}_k^{m,n+1}$ with end-points $0u_k^{m,n+1}$ and $0v_k^{m,n+1}$

But, according to case (v) of Lemma 6, $1u_k^{m+1,n}$ and $0u_k^{m,n+1}$ are connected in $H_k^{m+1,n+1}$. Therefore, by applying Lemma 4, we have

$$H_k^{m+1,n+1} = 1H_k^{m+1,n} \cup \{(1u_k^{m+1,n}, 0u_k^{m,n+1})\} \cup 0H_k^{m,n+1},$$

which it is a Hamiltonian path in $\mathbf{G}_k^{m+1,n+1} = 1\mathbf{G}_k^{m+1,n} \cup 0\mathbf{G}_k^{m,n+1}$. The end-points of this path are: $1v_k^{m+1,n} = u_k^{m+1,n+1}$ (by case (iv) of Lemma 6), and $0v_k^{m,n+1} = v_k^{m+1,n+1}$ (by case (ii) of Lemma 6). Therefore, Hamiltonian path $H_k^{m+1,n+1}$ is constructed.

Now, suppose $H_k^{(k-1)n+1,n-1}$ and $H_k^{(k-1)n,n}$ are Hamiltonian paths and we shall see the construction of $H_k^{(k-1)n+1,n}$.

- According to lemma 3, $1H_k^{(k-1)n+1,n-1}$ is a Hamiltonian path in $1\mathbf{G}_k^{(k-1)n+1,n-1}$ with end-points $1v_k^{(k-1)n+1,n-1}$ and $1u_k^{(k-1)n+1,n-1}$.
- According to Lemma 2, $0H_k^{(k-1)n,n}$ is a Hamiltonian path in $0\mathbf{G}_k^{(k-1)n,n}$ with end-points $0u_k^{(k-1)n,n}$ and $0v_k^{(k-1)n,n}$.

According to case (vi) of Lemma 6, $1u_k^{(k-1)n+1,n-1}$ is connected to $0v_k^{(k-1)n,n}$ in $H_k^{(k-1)n+1,n}$. Therefore by applying Lemma 4, we have

$$H_k^{(k-1)n+1,n} = 1H_k^{(k-1)n+1,n-1} \cup \{(1u_k^{(k-1)n+1,n-1}, 0u_k^{(k-1)n,n})\} \cup 0H_k^{(k-1)n,n},$$

which it is a Hamiltonian path in

$$\mathbf{G}_k^{(k-1)n+1,n} = 1\mathbf{G}_k^{(k-1)n+1,n-1} \cup 0\mathbf{G}_k^{(k-1)n,n}.$$

The end-points of this path are: $1v_k^{(k-1)n+1,n-1} = u_k^{(k-1)n+1,n}$ (by case (iv) of Lemma 6), and $0v_k^{(k-1)n,n} = v_k^{(k-1)n+1,n}$ (by case (ii) of Lemma 6). Therefore, Hamiltonian path $H_k^{(k-1)n+1,n}$ is constructed.

If $H_k^{(k-1)(n+1),n}$ is Hamiltonian path then $H_k^{(k-1)(n+1),n+1}$ is also Hamiltonian path. Indeed, by Lemma 5,

we add the prefix 1 to $H_k^{(k-1)(n+1),n}$ to obtain $H_k^{(k-1)(n+1),n+1}$ which is a Hamiltonian path in $\mathbf{G}_k^{(k-1)(n+1),n+1} = 1\mathbf{G}_k^{(k-1)(n+1),n}$ with end-points: $1v_k^{(k-1)(n+1),n} = u_k^{(k-1)(n+1),n+1}$ (by case (iv) of Lemma 6), and $1u_k^{(k-1)(n+1),n} = v_k^{(k-1)(n+1),n+1}$ (by case (iii) of Lemma 6).

According to the above theorem, the generating algorithm of bitstrings in Hamiltonian path $H_k^{m,n}$ is presented in next section.

4. Generating Algorithm

The algorithm *gen*(*m, n*) presented in Figure 2 generates the bitstrings with a given length and *k*-dominating property. This algorithm is based on the recurrence relations given in Lemma 1 and Theorem 2, and generates *x*-sequences in one dimensional array **x** with size *m* + *n*. With regard to the Theorem 2 in each step of the algorithm one of the operations *A, B, C,* and their inversion are employed on the input bitstring to construct the successive bitstring in Gray code order. For *n* = 1, *gen*(*m,* 1) produces $\mathbf{S}_k^{m,1}$ in lexicographical order if **x** is initiated by $0^{m-k+1}10^{k-1}$ or in antilexicographical order if **x** is initiated by 10^m. Initializing $\mathbf{x} = (10^{k-1})^n$ and recalling *gen*(*m, n*), it produces $\mathbf{S}_k^{m,n}$ for $m > (k-1)n > 0$. Finally, for $m = (k-1)n$ and recalling $gen((k-1)n, n)$ with the same initialization $\mathbf{x} = (10^{k-1})^n$, it produces $\mathbf{S}_k^{(k-1)n,n}$. Therefore, it generates all *x*-sequences corresponding to all *k*-ary trees with *n* nodes.

In this algorithm *d* is a global variable and initially is set to *nk* before performing $gen((k-1)n, n)$, and $swap(i, j)$ is a procedure for exchange x_i with x_j. Table 3 shows the *x*-sequences of length 22 corresponding to 4-ary trees with 3 nodes generated by our algorithm.

```
Void gen(int m, int n)
{
      int u, v;
      if (n==1){
            if (x[d-m]==1 {
                  if (m==k-1) return;
                  swap(d-m, d-m+1); print(x) ;
                  gen(m-1, 1);
            }
            else if((x[d-k+1]==1)&& (m!=k)) gen(k, 1);
                  else {
                        swap(d-m, d-m+1); print(x) ;
                        if(x[d-m-1]==) return;
                        gen(m+1, 1);
                  }
}
else  if(m==(k-1)*n) gen(m, n-1);
      else {
            u=0; v=1;
            if(x[d-m-n+1]==0) {
                  u=1; v=0;
            }
            gen(m-u, n-v);
            if(m>(k-1)*n+1) swap(d-m-n+1, d-k*(n-1)+1) :
            else swap(d-m-n+1, d-m-n+3) ;
            print(x); gen(m-v, n-u);
      }
}
```

Figure 2: Generation Algorithm for $\mathbf{S}_k^{m,n}$.

Rank	Bitstring	Operation	Rank	Bitstring	Operation
1	100010001000	A^{-1}	12	101001000000	A
2	100010010000	A^{-1}	13	101000100000	A
3	100010100000	A^{-1}	14	101000010000	A
4	100011000000	B^{-1}	15	101000001000	C^{-1}
5	100110000000	A	16	111000000000	A
6	100101000000	A	17	110100000000	A
7	100100100000	A	18	110010000000	A
8	100100010000	A	19	110001000000	A
9	100100001000	C^{-1}	20	110000100000	A
10	101100000000	A	21	110000010000	A
11	101010000000	A	22	110000001000	A

Table 3: The 22 x-Sequences of Length 12 for $n = 3$ and $k = 4$.

In order to analyze the running time of this algorithm we denote $T(m,n)$ be the number of calls to procedure $gen(m,n)$. Therefore, with regard to the generation algorithm, we can write the following recursive relation.

$$T(m,n)=\begin{cases} m-k+2 & \text{if } n=1, \\ 1+T((k-1)n,n-1) & \text{if } m=(k-1)n, \\ 1+T(m,n-1)+T(m-1,n) & \text{if } m>(k-1)n>0. \end{cases}$$

It can be shown by induction that $T(m,n)<2|\mathbf{S}_k^{m,n}|-1$, so $T(m,n)$ is proportional to $|\mathbf{S}_k^{m,n}|$, therefore the generation algorithm presented in this paper is performed in constant average time.

5. Conclusion

We have devised an algorithm for generating n-node k-ary trees represented by x-sequences. The algorithm is the extended form of algorithm by Vajnovszki for binary trees. The trees are generated by Gray codes in constant average time.

REFERENCES

1. AHRABIAN, H. and NOWZARI-DALINI, A., **Generation of *t*-ary trees with Ballot sequences**, Intern. J. Comput. Math., 80, 2003, pp. 1243-1249.
2. AHRABIAN, H. and NOWZARI-DALINI, A., **On the generation of P-sequences,** Adv. Modeling Optim., 5, 2003, pp. 27-38.
3. KAPRALSKI, A., **New methods for the generation of permutations, combinations and other combinatorial objects in parallel**, J. Parallel Distrib. Comput., 17, 1993, pp. 315-329.
4. KATAJAINEN, J. and MAKINEN, E., **Tree compression and optimization with application**, Inter. J. Found. Comput. Sci.,1 , 1990, pp. 425-447.
5. KNUTH, D.E., **The Art of Computer Programming, Vol. 1: Fundamental Algorithms**, ADDISON WESLY, Reading, 2nd ed., 1973.

6. KOKOSINSKI, Z., **On the generation of permutations through decomposition of symmetric group into cosets**, Bit, 30, 1990, pp. 583-591.

7. KOKOSINSKI, Z., **On parallel generation of *t*-ary trees in an associative model**, Lecture Notes in Computer Science, 2328, 2002, pp. 228-235.

8. KORSH, J.F., **Loopless generation of *k*-ary tree sequences**, Inform. Process. Lett., 52, 1994, pp. 243-247.

9. KORSH, J.F. and LAFOLLETTE, P., **Loopless generation of Gray codes for *k*-ary trees**, Inform. Process. Lett., 70, 1999, pp. 7-11.

10. KORSH, J.F. and LIPSCHUTZ, S., **Shift and loopless generation of *k*-ary trees**, Inform. Process. Lett., 65, 1998, pp. 235-240.

11. LUCAS, J., ROELANTS VAN BARONAIGIEN, D. and RUSKEY, F., **On rotations and the generation of binary trees**, J. Algorithms, 15, 1993, pp. 343-366.

12. MIRSKY, L., **Transversal Theory**, ACADEMIC PRESS, Washington, 1971.

13. OLARIU, S., SCHWING, J.L., WEN, Z. and ZHANG, J., **Optimal parallel encoding and decoding algorithms for trees**, J. ACM, 38, 1991, pp. 1-10.

14. PALLO, J. and RACCA, R., **A note on generating binary tree in A-order and B-order**, Intern. J. Comput. Math., 18, 1985, pp. 27-39.

15. PROSKUROWSKI, A. and RUSKEY, F., **Binary tree Gray codes**, J. Algorithms, 6, 1985, pp. 225-238.

16. ROELANTS VAN BARONAIGIEN, D., **A loopless algorithm for generating binary trees sequences**, Inform. Process. Lett., 39, 1991, pp. 189-194.

17. ROELANTS VAN BARONAIGIEN, D., **A loopless Gray code algorithm for listing *k*-ary trees**, J. Algorithms, 35 (2000), pp. 100-107.

18. UEHARA, T. and CLEEMPUT, W.M., **Optical layout of CMOS functional arrays**, IEEE Trans. Comput., 7, 1981, PP. 305-312.

19. VAJNOVSZKI, V., **Constant time algorithm for generating binary tree Gray codes**, Studies in Informatics and Control, 5, 1996, pp. 15-21.

20. WATERMAN, M.S., **Introduction to Computational Biology**, CRC PRESS, New York, 1995.

21. XIANG, L., USHIJIMA, K. and AKL, S., **Generating regular *k*-ary trees efficiently**, Comput. J., 43, 2000, pp. 290-300.

22. XIANG, L., USHIJIMA, K. and TANG, C., **Efficient loopless generation of Gray codes for *k*-ary trees**, Inform. Process. Lett., 79, 2000, pp. 169-174.

23. ZAKS, S., **Lexicographic generation of ordered tree**, Theoret. Comput. Sci., 10, 1980, pp. 63-82.