Information Processing Letters 76 (2000) 169–174

Information Processing Letters

www.elsevier.com/locate/ipl

# Efficient loopless generation of Gray codes for $k$-ary trees [☆]

Limin Xiang [a,*], Kazuo Ushijima [a], Changjie Tang [b]

[a] *Department of Computer Science and Communication Engineering, Kyushu University, 6-10-1 Hakozaki, Higashi-ku, Fukuoka 812-8581, Japan*

[b] *Department of Computer Science, Sichuan University, 610064 Chengdu, Sichuan, People's Republic of China*

Received 5 August 1999; received in revised form 25 May 2000

Communicated by S. Zaks

**Abstract**

Vajnovszki recently developed a loopless algorithm [Inform. Process. Lett. 68 (1998) 113] to enumerate Gray codes for binary trees, and then Korsh and Lafollette gave a loopless algorithm [Inform. Process. Lett. 70 (1999) 7] to generate Gray codes for $k$-ary trees. In this paper, another loopless algorithm is presented to list Gray codes for $k$-ary trees more efficiently in both space and time than the two former algorithms, and the algorithm is also conceptually simpler than the predecessors. Based on the algorithm, Gray codes for $k$-ary trees with $n$ internal nodes ($n \geqslant 2$ and $k > 3$) can be generated in at least $2^{2(n-1)}$ different ways easily. © 2000 Elsevier Science B.V. All rights reserved.

*Keywords:* Combinatorial problems; Gray codes; $k$-ary trees; Loopless generation

## 1. Introduction

Many algorithms have been developed for generating trees or tree sequences [1–9,11–15]. In [7], the first loopless algorithm was proposed for generating binary tree sequences (codewords [15]). For a loopless algorithm, the number of changes between two successive tree sequences must be bounded by a constant, e.g., the constant in [7] is 2. When the constant is 1, it is well known that the sequences are in Gray code order. In [9], Roelants van Baronaigien and Ruskey used a recursive algorithm to list w-sequences (i.e., weight sequences [5]) for binary trees in Gray code order, and in [6] Vajnovszki enumerated the same sequences using a loopless algorithm obtained by applying Williamson's algorithm [10]. Based on the method in [7], Korsh gave a loopless generating algorithm for $k$-ary tree sequences [1]. The different positions is also bounded by 2 between two successive $k$-ary tree sequences in [1]. Using a shift operation between two $k$-ary trees, Korsh and Lipschutz developed another loopless algorithm to list $k$-ary trees themselves in [3] (a loopless algorithm can be found in the Appendix A of [4] to list binary trees themselves), and inspired by the idea in [6], Korsh and LaFollette presented in [2] a loopless generation of Gray codes for $k$-ary trees.

In this paper, using Zaks' sequences [14] for $k$-ary trees, we present another loopless algorithm to list Gray codes for $k$-ary trees. Compared with the loopless algorithms to generate Gray codes for binary trees in [6] and for $k$-ary trees in [2], ours is more efficient in both space and time, and also conceptually simpler

[☆] This research was supported by the JSPS-RFTF-96P00603.

[*] Corresponding author.

*E-mail addresses:* xiang@csce.kyushu-u.ac.jp (L. Xiang), ushijima@csce.kyushu-u.ac.jp (K. Ushijima), chjtang@scu.edu.cn (C. Tang).

PII: S0020-0190(00)00139-3

than the predecessors. Based on the algorithm, Zaks' sequences for $k$-ary trees with $n$ internal nodes ($n \geqslant 2$ and $k > 3$) can be generated in at least $2^{2(n-1)}$ different Gray code orders easily.

## 2. Zaks' sequences

Given a regular $k$-ary tree $\mathbb{T}$ with $n$ internal nodes, $z = (z_1, z_2, \ldots, z_n)$ is a Zaks' sequence [14], where $z_i$ $(1 \leqslant i \leqslant n)$ is the position of the $i$th node in the preorder traversal of $\mathbb{T}$ for all $n$ nodes and $n(k-1)+1$ leaves. For example, the Zaks' sequence of the tree in Fig. 1 is $z = (1, 2, 4, 12, 16)$. The set of all the Zaks' sequences for $k$-ary trees with $n$ nodes, denoted by $\mathcal{Z}_k(n)$, is (see Theorem 7 in [14])

$$\mathcal{Z}_k(n) = \bigl\{(z_1, z_2, \ldots, z_n) \mid 1 = z_1 < z_2 < \cdots < z_n \text{ and } z_i \leqslant k(i-1)+1 \text{ for } 2 \leqslant i \leqslant n\bigr\}.$$

For simplicity, $k(i-1)+1$ will be denoted by $A_k(i)$ in the following.

Since

$$\bigl|\mathcal{Z}_k(n)\bigr| = \frac{1}{(k-1)n+1}\binom{kn}{n} \quad \text{and}$$

$$\bigl|\mathcal{Z}_k(1)\bigr| = \bigl|\mathcal{Z}_1(n)\bigr| = 1,$$

we will hereafter assume that $n \geqslant 2$ and $k \geqslant 2$. Thus, the following result can be obtained directly from the definition of $\mathcal{Z}_k(n)$, which will be a basis for us to describe the loopless algorithm to list sequences of $\mathcal{Z}_{n,k}$ in Gray code order.

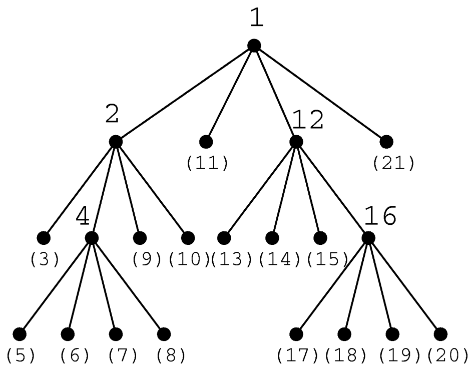


Fig. 1. A 5 node 4-ary tree.

**Theorem 1.** *For $i > 1$, if $(z_1, z_2, \ldots, z_{i-1}) \in \mathcal{Z}_k(i-1)$, then $(z_1, z_2, \ldots, z_{i-1}, z_i) \in \mathcal{Z}_k(i)$ if and only if $z_i \in X_i$, where $X_i = \{x \mid z_{i-1}+1 \leqslant x \leqslant A_k(i)\}$. Furthermore, $|X_i| \geqslant k \geqslant 2$.*

## 3. Algorithm

For a set of sequences with length $n$, if there is only one value at position $i$ $(1 \leqslant i \leqslant n)$, then the position is called to be constant, else variable. For $\mathcal{Z}_k(n)$, e.g., position 1 is constant and the others are variable. The key point for applying Williamson's loopless algorithm [10, p. 112], to generate variations with $n$ positions in Gray code order is to find out two distinct values, called *lower* and *upper* bounds, for each variable position so that values at each variable position may appear from its *lower* bound to its *upper* bound (in the direction *up*) or from its *upper* bound to its *lower* bound (in the direction *down*).

Let array $v[1..n]$ be used for listing sequences in Gray code order and $n$ positions are all variable, then the idea of Williamson loopless algorithm can be described as follows.

(I1) Two arrays $e[1..n+1]$ and $d[1..n]$ are employed. Array $e$ is used for keeping track of positions to be processed, and array $d$ is used for indicating the direction (*up* or *down*) for the values at each position appearing between its bounds.

(I2) The first sequence is when each $v[i]$ $(1 \leqslant i \leqslant n)$ is at its lower bound. Each $e[i]$ $(1 \leqslant i \leqslant n+1)$ and $d[i]$ $(1 \leqslant i \leqslant n)$ are initialized to $i-1$ and *up*, respectively. $e[n+1]$ supplies the position $j$ to be processed.

(I3) The *next* sequence of $v[1..n]$ will be obtained by changing the value at position $j$, i.e., $v[j]$.
   (1) $e[n+1]$ is assigned $n$.
   (2) $v[j]$ is assigned its successor or predecessor according to the direction $d[j]$.
   (3) If the value of $v[j]$ is a bound at position $j$, then $d[j]$, $e[j+1]$ and $e[j]$ are assigned its NOT, $e[j]$ and $j-1$, respectively, where NOT means *down*$(-1)$/*up*$(1)$ if the value is *up*$(1)$/*down*$(-1)$ of $d[j]$.
   (4) $j$ is assigned $e[n+1]$.

(I4) When $j = 0$, the last sequence has been obtained.

```
**void** *next*()
{ $e[n+1] = n$;
  **if** $(d[j] == 1)$
     $v[j] = successor(v[j])$; **else**
     $v[j] = predecessor(v[j])$;
  **if** $is\_a\_bound(v[j])$
     {$d[j] = -d[j]$; $e[j+1] = e[j]$; $e[j] = j-1$;}
  $j = e[n+1]$;
}
```

Fig. 2. Williamson's loopless algorithm.

Thus, Williamson's loopless algorithm in $C$ to generate *next* sequence of $v[1..n]$ can be described as in Fig. 2.

To apply Williamson's loopless algorithm for $\mathcal{Z}_k(n)$, by Theorem 1, noting that both $A_k(i)$ and $A_k(i)-1$ are in $X_i$, we can let $A_k(i)$ and $A_k(i)-1$ be the lower and upper bounds of position $i$ ($2 \leqslant i \leqslant n$), respectively, and let the direction *up* at position $i$ be:

$$A_k(i) \to z_{i-1}+1 \to z_{i-1}+2 \to \cdots \to A_k(i)-1$$

and the direction *down* be the reverse of its *up*. When position $j$ is in direction *up*, we have the rule:

(i) **if** the value of $z_j$ is its lower bound, i.e., $A_k(j)$,
  **then** the next value of $z_j$ is the value of $z_{j-1}+1$,
  **else** the value of $z_j+1$.

When position $j$ is in direction *down*, we have the rule:

(ii) **if** the value of $z_j$ is the value of $z_{j-1}+1$,
  **then** the next value of $z_j$ is its lower bound,
   i.e., $A_k(j)$,
  **else** the value of $z_j-1$.

Thus, by applying Williamson's loopless algorithm, noting that the next position will be $n$ when the value changed just now is not a bound at its position, we can give our algorithm in Fig. 3 to generate Zaks' codes for $k$-ary trees in Gray code order, where array $z[1..n]$ is used to list sequences of $\mathcal{Z}_k(n)$. The algorithms of [6] and [2] are in Pascal and C, respectively, and ours will be in C, too.

By Theorem 1 and the conditions for arrays $e$ and $d$, each $z[i]$ and $A[i]$ ($1 \leqslant i \leqslant n$) are initialized to $k(i-1)+1$, $e[i]$ ($1 \leqslant i \leqslant n+1$) to $i-1$, $d[i]$ ($1 \leqslant i \leqslant n$) to 1, and $j$ to $n$. When $j$ is 1 the last sequence of $\mathcal{Z}_k(n)$ has been obtained, because position 1 is constant. A complete implementation in

```
**void** *nextZ*()
{ **if** $(d[j] == 1)$
     { **if** $(z[j] == A[j])$
        $z[j] = z[j-1]+1$; **else** $z[j] = z[j]+1$;
     } **else**
     { **if** $(z[j] == z[j-1]+1)$
        $z[j] = A[j]$; **else** $z[j] = z[j]-1$;
     }
  **if** $(z[j] >= A[j]-1)$
     { $d[j] = -d[j]$; $e[j+1] = e[j]$; $e[j] = j-1$;
          $j = e[n+1]$; $e[n+1] = n$;
     } **else** $j = n$;
}
```

Fig. 3. Loopless algorithm to generate $\mathcal{Z}_k(n)$.

$C$ can be found at http://www.swlab.csce.kyushu-u.ac.jp/∼xiang/Gen_Z_codes.c.

Fig. 4 shows the 55 Zaks' sequences generated in Gray code order for the 4 node 3-ary trees.

## 4. List Zaks' sequences in different Gray code orders

Lucas et al. proved that four series of binary trees, corresponding respectively to four kinds of integer sequences generated by algorithms of Alg-Z, Alg-P, Alg-M and Alg-RP are isomorphic [4]. For special sequences, by changing the correspondence between the sequences and binary trees simply, non-isomorphic series of binary trees may be obtained [11]. Based on the algorithm *nextZ*() given in Section 3, Zaks' sequences can be generated in many different Gray code orders easily, i.e., non-isomorphic series of $k$-ary trees can be obtained easily without changing the correspondence between Zaks' sequences and $k$-ary trees.

(1) If we let $d_i = 1$ or $d_i = -1$ for $2 \leqslant i \leqslant n$, and the first sequence of $\mathcal{Z}_k(n)$ be $(1, a_2, a_3, \ldots, a_n)$, where $a_i = A_k(i)$ or $a_i = A_k(i)-1$ according to $d_i = 1$ or $d_i = -1$ for $2 \leqslant i \leqslant n$, then *nextZ*() can be used to generate $\mathcal{Z}_k(n)$ in $2^{n-1}$ different Gray code orders.
(2) If we exchange the lower bound with its upper bound for each position $i$ ($2 \leqslant i \leqslant n$) and modify *nextZ*() properly, we can list $\mathcal{Z}_k(n)$ ($k > 3$) in another $2^{n-1}$ different Gray code orders.

| | | | | |
|---|---|---|---|---|
| *1.* (1, 4, 7, 10) | *12.* (1, 4, 6, 9 ) | *23.* (1, 2, 4, 8 ) | *34.* (1, 2, 3, 9 ) | *45.* (1, 3, 4, 5 ) |
| *2.* (1, 4, 7, 8 ) | *13.* (1, 2, 6, 9 ) | *24.* (1, 2, 4, 7 ) | *35.* (1, 2, 7, 9 ) | *46.* (1, 3, 4, 10) |
| *3.* (1, 4, 7, 9 ) | *14.* (1, 2, 6, 8 ) | *25.* (1, 2, 4, 6 ) | *36.* (1, 2, 7, 8 ) | *47.* (1, 3, 5, 10) |
| *4.* (1, 4, 5, 9 ) | *15.* (1, 2, 6, 7 ) | *26.* (1, 2, 4, 5 ) | *37.* (1, 2, 7, 10) | *48.* (1, 3, 5, 6 ) |
| *5.* (1, 4, 5, 8 ) | *16.* (1, 2, 6, 10) | *27.* (1, 2, 4, 10) | *38.* (1, 3, 7, 10) | *49.* (1, 3, 5, 7 ) |
| *6.* (1, 4, 5, 7 ) | *17.* (1, 2, 5, 10) | *28.* (1, 2, 3, 10) | *39.* (1, 3, 7, 8 ) | *50.* (1, 3, 5, 8 ) |
| *7.* (1, 4, 5, 6 ) | *18.* (1, 2, 5, 6 ) | *29.* (1, 2, 3, 4 ) | *40.* (1, 3, 7, 9 ) | *51.* (1, 3, 5, 9 ) |
| *8.* (1, 4, 5, 10) | *19.* (1, 2, 5, 7 ) | *30.* (1, 2, 3, 5 ) | *41.* (1, 3, 4, 9 ) | *52.* (1, 3, 6, 9 ) |
| *9.* (1, 4, 6, 10) | *20.* (1, 2, 5, 8 ) | *31.* (1, 2, 3, 6 ) | *42.* (1, 3, 4, 8 ) | *53.* (1, 3, 6, 8 ) |
| *10.* (1, 4, 6, 7 ) | *21.* (1, 2, 5, 9 ) | *32.* (1, 2, 3, 7 ) | *43.* (1, 3, 4, 7 ) | *54.* (1, 3, 6, 7 ) |
| *11.* (1, 4, 6, 8 ) | *22.* (1, 2, 4, 9 ) | *33.* (1, 2, 3, 8 ) | *44.* (1, 3, 4, 6 ) | *55.* (1, 3, 6, 10) |

Fig. 4. All the Zaks' sequences for the 4 node 3-ary trees.

(3) If we exchange lower bounds with upper bounds at some of variable positions (i.e., there are $2^{n-1}$ choices), we can list $\mathcal{Z}_k(n)$ $(k > 3)$ in $2^{n-1} \times 2^{n-1} = 2^{2(n-1)}$ different Gray code orders.

(4) In the methods above, if we only change the order of elements appearing between $z_{n-1} + 1$ and $A_k(n) - 2$ for $z_n$, i.e., there are $(|X_n| - 2)!$ ways, we can list $\mathcal{Z}_k(n)$ in more different Gray code orders.

Obviously, besides the Gray code orders above, there are many other different Gray code orders for $\mathcal{Z}_k(n)$, but it may be difficult to find out loopless algorithms to generate $\mathcal{Z}_k(n)$ in them.

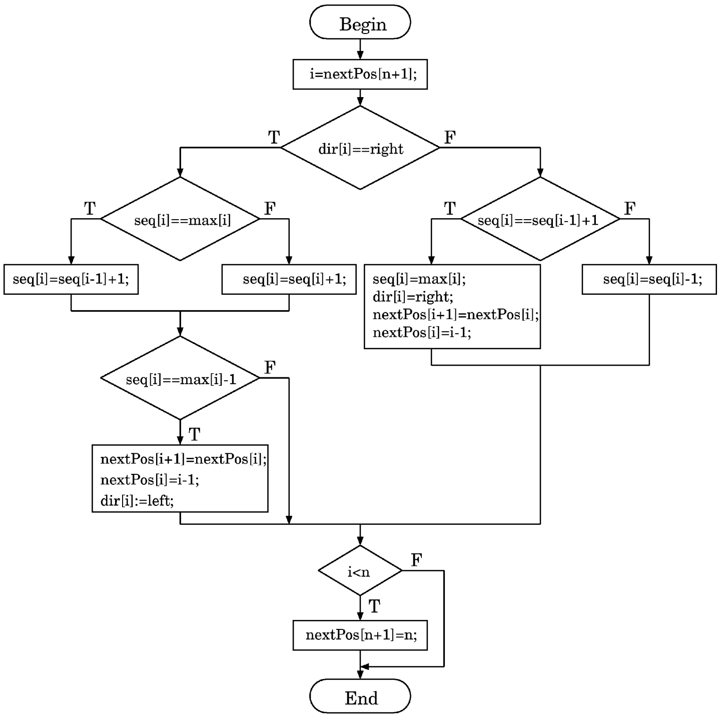
## 5. Conclusion

We have presented a loopless algorithm to list Zaks' sequences $\mathcal{Z}_k(n)$ [14] for $k$-ary trees with $n$ internal nodes in Gray code order, shown that the algorithm can be used to list $\mathcal{Z}_k(n)$ in many different Gray code orders. Our algorithm needs $4n + \mathrm{O}(1)$ space and $2/6$ assignment and 3 (simple) comparison operations in the best/worst case from a sequence to its successor. To enumerate Gray codes for binary trees, the algorithm in [6] needs $8n + \mathrm{O}(1)$ space and executes $9/13$ assignment and $3/5$ comparison operations in its best/worst case from a sequence to its successor. For $n$ (internal) node $k$-ary (regular) trees, the algorithm in [2] needs $(k + 4)n + \mathrm{O}(1)$ space and does $12/19$ assignment and $4/5$ (composite) comparison operations in its best/worst case between two successive sequences. Therefore, our algorithm is more efficient in both space and time, and the algorithm is also conceptually simpler than the predecessors.

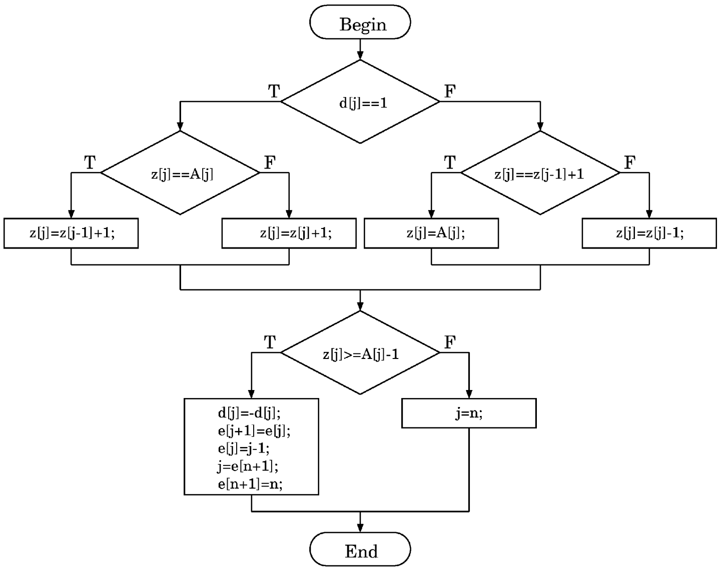
### The latest developments

When we revised an earlier version of this paper, we learned of a similar loopless algorithm *Next*() in [8] to generate Zaks' sequences by Roelants van Baronaigien that was independently discovered. Two flow charts are shown in Fig. 5 for algorithms *Next*() and *nextZ*(). With the same space $4n + \mathrm{O}(1)$, *Next*() needs $2/6$ assignment and $3/4$ comparison operations in the best/worst case from a sequence to its successor. Therefore, *nextZ*() is a little faster than *Next*(), which is also confirmed by the following *actual running times*, where the same computer and the same compiler were used for the two algorithms, and in each case *printing a sequence* was replaced with *adding* 1 *to a counter*. (There is a typo at line 14 on page 105 of [8], i.e., each $Max[i]$ should be initialized to $k(i - 1) + 1$, not $ki + 1$. To generate all the sequences, *Next*() is called one more time than *nextZ*().)

In addition, since the correspondence is simple between $\mathcal{Z}_k(n)$ and $k$-ary trees with $n$ internal nodes,



(a) Roelants van Baronaigien's generation algorithm *Next*( )



(b) Our generation algorithm *nextZ*( )

Fig. 5. Two flow charts.

Table 1
Some actual running times for *Next*() and *nextZ*()

| $(n,k)$ | $(7,8)$ | $(7,9)$ | $(7,10)$ | $(8,8)$ | $(8,9)$ | $(8,10)$ |
|---|---|---|---|---|---|---|
| *Next*() | 2.41″ | 5.05″ | 9.72″ | 40.58″ | 1’35.79″ | 3’25.80″ |
| *nextZ*() | 2.30″ | 4.77″ | 9.22″ | 38.61″ | 1’31.01″ | 3’15.31″ |

$\mathcal{Z}_k(n)$ can be used to help to obtain efficient algorithms for generating $k$-ary trees themselves, which has been discussed in [13] in depth.

## Acknowledgements

The authors are very grateful to anonymous referees whose careful review improved the quality of the paper to a great extent.

## References

[1] J.F. Korsh, Loopless generation of $k$-ary tree sequences, Inform. Process. Lett. 52 (1994) 243–247.
[2] J.F. Korsh, P. LaFollette, Loopless generation of Gray codes for $k$-ary trees, Inform. Process. Lett. 70 (1999) 7–11.
[3] J.F. Korsh, S. Lipschutz, Shifts and loopless generation of $k$-ary trees, Inform. Process. Lett. 65 (1998) 235–240.
[4] J.M. Lucas, D. Roelants van Baronaigien, F. Ruskey, On rotations and the generation of binary trees, J. Algorithms 15 (1993) 343–366.
[5] J.M. Pallo, Enumerating, ranking and unranking binary trees, Comput. J. 29 (1986) 171–175.
[6] V. Vajnovszki, On the loopless generation of binary tree sequences, Inform. Process. Lett. 68 (1998) 113–117.
[7] D. Roelants van Baronaigien, A loopless algorithm for generating binary tree sequences, Inform. Process. Lett. 39 (1991) 189–194.
[8] D. Roelants van Baronaigien, A loopless Gray-code algorithm for listing $k$-ary trees, J. Algorithms 35 (2000) 100–107.
[9] D. Roelants van Baronaigien, F. Ruskey, A Hamiltonian path in the rotation lattice of binary trees, Congr. Numer. 59 (1987) 313–318.
[10] S.G. Williamson, Combinatorics for Computer Science, Computer Science Press, Rockville, MD, 1985.
[11] L. Xiang, C. Tang, K. Ushijima, Grammar-oriented enumeration of binary trees, Comput. J. 40 (1997) 278–291.
[12] L. Xiang, K. Ushijima, Grammar-oriented enumeration of arbitrary trees and arbitrary $k$-ary trees, IEICE Trans. Inform. & Syst. E82-D (1999) 1245–1253.
[13] L. Xiang, K. Ushijima, C. Tang, On generating $k$-ary trees in computer representation, Inform. Process. Lett. 78 (2001) to appear.
[14] S. Zaks, Lexicographic generation of ordered trees, Theoret. Comput. Sci. 10 (1980) 63–82.
[15] D. Zerling, Generating binary trees using rotations, J. ACM 32 (1985) 694–701.