19th International Computer Science and Engineering Conference (ICSEC)
Chiang Mai, Thailand, 23-26 November, 2015

Ranking Left-weight Sequences of Binary Trees in Gray-code Order

Ro-Yu Wu*, Jou-Ming Chang†, Sheng-Lung Peng‡ and Shun-Chieh Chang§

*Department of Industrial Management, Lunghwa University of Science and Technology, Taoyuan, Taiwan

†Institute of Information and Decision Sciences, National Taipei University of Business, Taipei, Taiwan

‡Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan

§Department of Information Management, National Taiwan University of Science and Technology, Taipei, Taiwan

***Abstract*—LW-sequences are in common currency for encoding binary trees. Wu et al. [13] proposed an algorithm associated with tree rotations for listing all binary trees in diverse representations including LW-sequences. In particular, such a list of LW-sequences is generated in Gray-code order. Based on this ordering, we propose an efficient algorithm for ranking binary trees with n internal nodes. Our algorithm can be run in $\mathcal{O}(n^2)$ time and requires $\mathcal{O}(n)$ space.**

***Keywords*—Ranking algorithms; Unranking algorithms; Gray-code order; Left-weight sequences; Left-distance sequence**

I. Introduction

There are many applications in computer science to exhaustively generate a class of combinatorial objects (objects for short), e.g. combinatorial group testing, counterexample searching, and algorithm performance analyzing. It is the practice to encode objects into integer sequences and generate these sequences in a particular order. Due to enormous sequences need to be generated, it is necessary to design an efficient generating scheme. In particular, Gray-code order is a very common use in recent development [7], [8], [11], [13], [16]–[18] because the change between two successive sequences is restricted to only one position. In addition, it is important to efficiently rank and unrank objects in a list of Gray-code order [1], [14], [15]. For a set of objects generated in a particular order, a *ranking algorithm* can be used to determine the ranking of an appointed object. By contrast, an *unranking algorithm* can be used to produce the corresponding sequence of object with respect to the specific ranking.

Binary trees are the most fundamental objects used in data structure and so that there are many algorithms for generating these objects [4], [6], [7], [9]–[11], [13], [17]. Pallo [6] adopted the so-called *left-weight sequences* (LW-sequences for short) to encode binary trees. Moreover, for generating LW-sequences in lexicographic order, Pallo proposed a constant amortized time algorithm (CAT algorithm for short). See also [12] for another recursive CAT algorithm. At a later time, more algorithms that generates LW-sequences in Gray-code order were provided in [7], [10]. Recently, a loopless algorithm associated with tree rotations for simultaneously generating four types of binary tree representations in Gray-code order (including LW-sequences) was proposed by Wu et al. [13].

Based on the Gray-code order of LW-sequences introduced in [13], we will propose an efficient ranking algorithm in this paper. The technique used in this paper relies on a set of corresponding sequences of binary trees called *left-distance sequences* (LD-sequences for short) that was originated by Mäkinen in [5]. The correspondence between LW-sequences and LD-sequences was found out by Lucas et al. [4]. In addition, a flip-flap tree structure introduced in [13] will facilitate our discussion about the Gray-code order of LW-sequences.

We organize the remaining part of this paper as follows. In Section 2, we give the definitions of LW-sequences and LD-sequences formally and show the correspondence between them. In Section 3, we introduce the structure of flip-flap trees to generate the Gray-code order of LW-sequences. In Section 4, we present our ranking algorithm and show the correctness. Finally, we provide a concluding remark in the last section.

II. Preliminaries

A rooted and ordered tree is called an *extended binary tree* if every internal node has exactly two children. i.e., the *left child* and the *right child* [2]. We suppose that an extended binary tree T with n internal nodes are numbered from 1 to n in symmetric order (i.e., we visit the tree T in the left subtree, the root, and the right subtree recursively). For a node $i \in T$, we denote T_i as the subtree rooted at i. Moreover, the subtree rooted at the left child of i, denoted by L_i, is called the *left subtree* of i, and the subtree rooted at the right child of i, denoted by R_i, is called the *right subtree* of i. Also, The path from the root to its leftmost leaf is called the *left arm* of T.

A. Left-weight sequences

For a binary tree T, the *left weight* of a node $i \in T$, denoted by $w(T, i)$, is defined to be the number of leaves in L_i. Pallo [6] further defined the integer sequence $w(T) = (w_1, w_2, \ldots, w_n)$ to be the *left-weight sequence* (LW-sequence for short) of T, where we simply write w_i instead of $w(T, i)$. Note that LW-sequences are the most adoption for encoding binary trees. For example, Fig. 1 shows a binary tree T with 9 internal nodes whose LW-sequence is $w(T) = (1, 2, 1, 1, 1, 2, 1, 5, 7)$. Pallo [6] characterized an integer sequence $(w_1, w_2, \ldots, w_n)$ to be an LW-sequence of a binary tree if and only if the following conditions are fulfilled for all $i \in \{1, 2, \ldots, n\}$: (i) $1 \leqslant w_i \leqslant i$ and (ii) $i - w_i \leqslant j - w_j$ for all $i - w_i + 1 \leqslant j \leqslant i$. The following lemma shows that a node on the left arm of a subtree is easy to be checked.

Lemma 1. *(See [12], [13].) Let T be a binary tree and $j \in T$ an internal node. A node i is contained in the left arm of R_j if and only if $i - w_i = j$.*

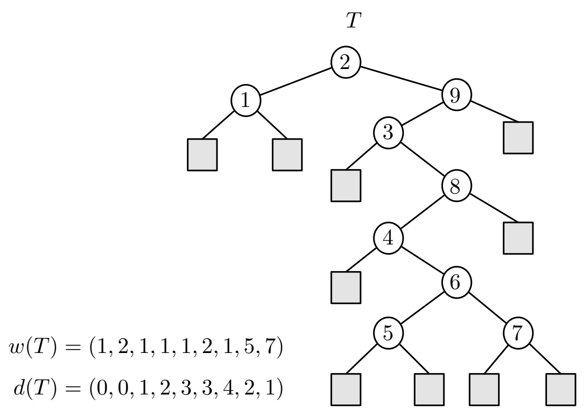
T
$w(T) = (1, 2, 1, 1, 1, 2, 1, 5, 7)$
$d(T) = (0, 0, 1, 2, 3, 3, 4, 2, 1)$

Fig. 1. LW-sequences and LD-sequences of a binary tree.

B. Left-distance sequences

Mäkinen [5] used a measure called *left distance* to represent a binary tree. For a binary tree T, a *left-distance sequence* (LD-sequence for short) of T, denoted by $d(T) = (d_1, d_2, \ldots, d_n)$, is an integer sequence to represent such a binary tree, where the left distance d_i for each node $i \in T$ is recursively defined as follows:

$$d_i = \begin{cases} 0 & \text{if } i \text{ is the root of } T; \\ d_{p(i)} & \text{if } i \text{ is a left child;} \\ d_{p(i)} + 1 & \text{if } i \text{ is a right child,} \end{cases}$$

where $p(i)$ stands for the parent of node i in T. Obviously, every node lying on the left arm of T has the left distance 0. Moreover, for a node i, every node lying on the left arm of R_i has left distance $d_i + 1$. For instance, the LD-sequence of the binary tree shown in Fig. 1 is $d(T) = (0, 0, 1, 2, 3, 3, 4, 2, 1)$. Mäkinen [5] characterized an integer sequence $(d_1, d_2, \ldots, d_n)$ to be an LD-sequence if and only if the following conditions are fulfilled: $d_1 = 0$ and $0 \leqslant d_i \leqslant d_{i-1} + 1$ for $2 \leqslant i \leqslant n$.

C. Sequence correspondences

Let $\mathbf{W}(n)$ and $\mathbf{D}(n)$ be the sets of all LW-sequences and LD-sequences with length n, respectively. Lucas et al. [4] revealed close relationship between these two sets. By transforming from a particular representation of binary trees, called *codewords*, introduced by Zerling [19] to LW-sequences and LD-sequences respectively, Lucas et al. showed that the lists of $\mathbf{W}(n)$ and $\mathbf{D}(n)$ preserve the reverse lexicographic order. Indeed, the relationship is based solely on the property of sequences and not of the corresponding trees. Since till now there is no explicit scheme to transform LW-sequences to LD-sequences or vice versa, this inspires us to provide transformations between these two types of sequences. Before this, we need some auxiliary properties.

Lemma 2. *Let T be a binary tree. For every internal node $i \in T$,*

$$d_i = \begin{cases} 0 & \text{if } i = 1; \\ d_{i-1} + 1 & \text{if } i \geqslant 2 \text{ and } w_i = 1; \\ d_{i-w_i+1} & \text{otherwise.} \end{cases}$$

Proof. Clearly, $d_1 = 0$. Let i be the node in the left arm of a subtree R_j for $j \geqslant 1$ such that $w_i = 1$. It is obvious that $i = j+1 \geqslant 2$. Since every node lying on the left arm of R_j has left distance d_j+1, we have $d_i = d_j+1 = d_{i-1}+1$. Moreover, if i' is a node in the left arm of R_j and $i' \neq i$, by Lemma 1 we have $j = i' - w_{i'}$. Thus, $d_{i'} = d_i = d_{j+1} = d_{i'-w_{i'}+1}$. □

Lemma 3. *Let T be a binary tree and $i \in T$ an internal node. Then*

$$w_i = \begin{cases} i & \text{if } d_i = 0; \\ 1 & \text{if } d_i \neq 0 \text{ and } d_i > d_{i-1}; \\ w_{i-1} + 1 & \text{if } d_i \neq 0 \text{ and } d_i = d_{i-1}; \\ i - F(d_i) + w_{F(d_i)} & \text{if } d_i \neq 0 \text{ and } d_i < d_{i-1}, \end{cases}$$

where $F(d_i) = \max\{j \in T\colon\ d_j = d_i \text{ for } j < i\}$.

Proof. We imagine that T is the right subtree of a dummy node numbered by 0. If $d_i = 0$, then the node i is contained in the left arm of R_0 (i.e., the left arm of T). By Lemma 1, we have $w_i = i$. We now consider $d_i \neq 0$. For $d_i > d_{i-1}$, the node i is contained in the left arm of R_{i-1}. Again by Lemma 1, we have $i - w_i = i - 1$, and thus $w_i = 1$. For $d_i = d_{i-1}$, it is clear that $i-1$ is the left child of i, and the right child of $i-1$ must be a leaf. Thus, $w_i = w_{i-1} + 1$. Finally, for $d_i < d_{i-1}$, it is clear that $i - 1 \in L_i$ and the node numbered by $F(d_i)$ is the left child of i. Since the number of leaves in $R_{F(d_i)}$ is $i - F(d_i)$, this implies that $w_i = i - F(d_i) + w_{F(d_i)}$. □

Based on Lemma 2 and Lemma 3, linear time transformations between LW-sequences and LD-sequences are shown in Fig 2. Therefore, we obtain the result of Theorem 4.

Theorem 4. *Transformations between an LW-sequence and an LD-sequence of length n can be done in $\mathcal{O}(n)$ time.*

```
Procedure LW-sequence to LD-sequence
begin
    $d_1 \leftarrow 0$;
    for $i \leftarrow 2$ to $n$ do
        if $w_i = 1$ then
            $d_i \leftarrow d_{i-1} + 1$;
        else
            $d_i \leftarrow d_{i+1-w_i}$;
```

```
Procedure LD-sequence to LW-sequence
begin
    for $i \leftarrow 1$ to $n$ do
        if $d_i = 0$ then $w_i \leftarrow i$;
        else
            if $d_i > d_{i-1}$ then $w_i \leftarrow 1$;
            if $d_i = d_{i-1}$ then $w_i \leftarrow w_{i-1} + 1$;
            if $d_i < d_{i-1}$ then $w_i \leftarrow i - F(d_i) + w_{F(d_i)}$;
        $F(d_i) \leftarrow i$;
```

Fig. 2. Linear time transformations between LW- and LD-sequences.

III. Flip-Flap Trees and Gray-Code Order Enumerations

In [13], Wu et al. adopted the following coding tree, called *flip-flap tree*, to describe the Gray-code order of all LW-sequences of length n in a systematic way. A flip-flap tree $\mathbb{T}_n$ is a rooted tree consisting of n levels such that every node is associated with a label w_i and every path from the root of $\mathbb{T}_n$ to a leaf represents a distinct LW-sequence $(w_1, w_2, \ldots, w_n)$. A non-leaf node $x \in \mathbb{T}_n$ has an *up-fragment* (respectively, a *down-fragment*) if the labels of x's children in $\mathbb{T}_n$ are arranged (from left to right) in increasing order (respectively, decreasing order). In particular, a flip-flap tree introduced in [13] fulfills the following conditions: (1) there is only one node (i.e., the root) with label 1 in level 1; (2) for each level $i \geqslant 2$, the two particular labels 1 and i appear in the boundary of each fragment; (3) every non-leaf node has either an up-fragment or a down-fragment; (4) if a node has an up-fragment, then its adjacency siblings (if exist) must have a down-fragment, and vice versa (i.e., up-fragments and down-fragments alternately appear in each level of $\mathbb{T}_n$). According to the correspondences between $\mathbf{W}(5)$ and $\mathbf{D}(5)$, Fig. 3 shows a flip-flap tree $\mathbb{T}_5$, where every node is associated with two label w_i and d_i (For brief, we denoted by w_i/d_i).

In the above arrangement of $\mathbb{T}_n$ if x and y are two adjacent leaves, the sequences from the root to x and to y are said to be *consecutive*. Wu et al. [13] showed that in a flip-flap tree any two consecutive LW-sequences differ in exactly one digit. This establishes the base to enumerate LW-sequences in Gray-code order. Although the list of LW-sequences produces a Gray-code order, it should be note that the list of the corresponding LD-sequences does not. For instance, the second and the third LD-sequences generated from $\mathbb{T}_5$ are $(0,0,0,0,1)$ and $(0,0,0,1,2)$, respectively (see Fig. 3). Nonetheless, the usage of LD-sequences has the advantage of ranking binary trees in the arrangement of $\mathbb{T}_n$. The following property is significant when we design the ranking algorithm.

Proposition 5. *Let $x \in \mathbb{T}_n$ be a non-leaf node with label (i.e., the left distance) k in the ith level. Then, x has $k+2$ sons labeld by $0, 1, \ldots, k+1$, where these sons are arranged in either an up-fragment or a down fragment.*

In what follows, unless otherwise stated, we use left distance to indicate the label of a node in $\mathbb{T}_n$. Let $A_{i,k}$ denote the number of leaves in the subtree rooted at a node with label k in the ith level of $\mathbb{T}_n$. Obviously, $A_{n,k} = 1$ for $0 \leqslant k \leqslant n-1$ and $A_{n-1,k} = k+2$ for $0 \leqslant k \leqslant n-2$. In general, by Proposition 5, we have

$$A_{i,k} = \sum_{j=0}^{k+1} A_{i+1,j} \tag{1}$$

where $1 \leqslant i \leqslant n-1$ and $0 \leqslant k \leqslant i-1$.

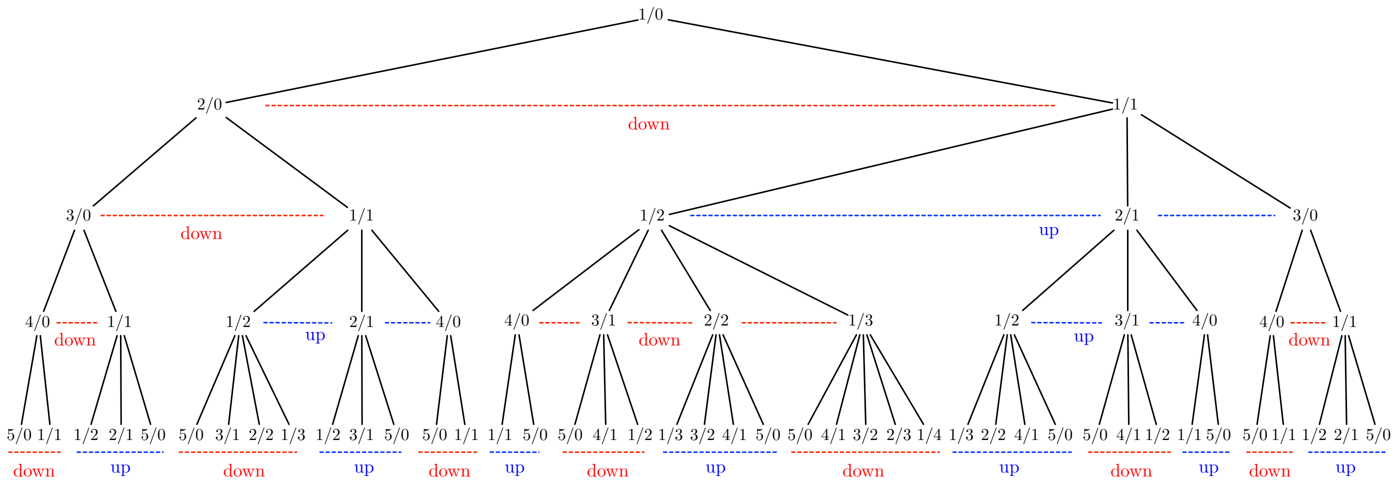

Fig. 3. A flip-flap tree $\mathbb{T}_5$ for LW-sequences and LD-sequences.

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

$$B_{i,k} = \sum_{j=0}^{k} A_{i,j} \tag{2}$$

where $1 \leqslant i \leqslant n$ and $0 \leqslant k \leqslant i-1$. Solving Eq. (2), we obtain the following lemma.

Lemma 6. *Let $n \geqslant 2$ be an integer. For each $i = 2, \ldots, n$ and $1 \leqslant k \leqslant i-1$, we have*

$$B_{i,k} = \frac{k+1}{2m+k+1}\binom{2m+k+1}{m} \tag{3}$$

where $m = n-i+1$.

Table I and Table II show the computational results of $A_{i,k}$ and $B_{i,k}$ for $n = 5$. For convenience, such tables are called *triangle table* and *accumulation table*, respectively.

TABLE I. THE TRIANGLE TABLE FOR $n = 5$

$A_{i,k}$		k				
n	i	0	1	2	3	4
5	1	42				
	2	14	28			
	3	5	9	14		
	4	2	3	4	5	
	5	1	1	1	1	1

TABLE II. THE ACCUMULATION TABLE FOR $n = 5$

$B_{i,k}$		k				
n	i	0	1	2	3	4
5	1	42				
	2	14	42			
	3	5	14	28		
	4	2	5	9	14	
	5	1	2	3	4	5

Note that the term $B_{i,k}$ indicates the total number of leaves for those subtrees rooted at nodes with labels from 0 to k in the ith level of $\mathbb{T}_n$. In addition, from Eq. (3) we can obtain the following corollary which is useful for designing ranking algorithm.

Corollary 7. *Let $n \geqslant 2$ be integer. For each $i = 2, \ldots, n$ and $1 \leqslant k \leqslant i-1$, we have*

$$B_{i-1,k} = \frac{B_{i,k}}{m+1} \cdot \frac{(2m+k+1)(2m+k+2)}{m+k+2} \tag{4}$$

where $m = n-i+1$.

IV. Ranking algorithm

In this section, we will develop the ranking algorithm. Let $\mathscr{T}_n$ be the set of binary trees with n internal nodes. It is well-known that $|\mathscr{T}_n| = \frac{1}{n+1}\binom{2n}{n}$ (i.e., the nth level of Catalan number [2]). Since the correspondence between LW- and LD-sequences has already built in the previous section, we can obtain the rank of a binary tree $T \in \mathscr{T}_n$ in the Gray-code order of LW-sequences if the rank of T associating with the LD-sequence in $\mathbb{T}_n$ is provided.

For a given binary tree T associated with LD-sequence $d(T) = (d_1, d_2, \ldots, d_n)$, let x_i be the node with label d_i in $\mathbb{T}_n$, and let $R(i)$ be the rank of x_i in the ith level of $\mathbb{T}_n$. Note that the goal of our ranking algorithm is to find $R(n)$. For instance, if we consider the path x_1, x_2, x_3, x_4, x_5 with labels $0, 1, 2, 2, 0$ in $\mathbb{T}_5$ (see Fig. 3), we have $R(1) = 0$, $R(2) = 1$, $R(3) = 2$, $R(4) = 7$ and $R(5) = 22$. Since each level of $\mathbb{T}_n$ begins with an up fragment for LD-sequences and the two types of fragments appear alternately, it is easy to check the following property.

Proposition 8. *For $1 \leqslant i < n$, if the rank of a node in the ith level of $\mathbb{T}_n$ is even (respectively, odd), then its sons are arranged in an up fragment (respectively, a down fragment).*

Moreover, since the ranking is performed from left to right in $\mathbb{T}_n$ and it starts with 0 at the beginning, we first set $R(i) = 0$ for each $i = 1, 2, \ldots, n$. From Proposition 8, there are two cases to update $R(j)$ for $i+1 \leqslant j \leqslant n$. If $R(i) \equiv 0$ (mod) 2, the sons of x_i are arranged in an up fragment. In this case, we have

$$R(j) = R(j) + B_{\ell, d_{i+1}-1} \tag{5}$$

provided that $d_{i+1} \neq 0$ and $\ell = n-j+i+1$. On the other hand, if $R(i) \equiv 1$ (mod) 2, the sons of x_i are arranged in a down fragment. In this case, we have

$$R(j) = R(j) + (B_{\ell, d_i+1} - B_{\ell, d_{i+1}}) \tag{6}$$

provided that $d_{i+1} \neq d_i + 1$ and $\ell = n-j+i+1$. According to Eqs. (5) and (6), we design the algorithm shown in Fig. 4.

```
Function Ranking($d_1, d_2, \ldots, d_n$)
  begin
    for $i = 1$ to $n$ do  $R(i) = 0$;
    for $i = 1$ to $n-1$ do
      if $R(i) \equiv 0$ (mod) 2 then   // an up fragment
        if $d_{i+1} \neq 0$ then
          for $j = i+1$ to $n$ do
            $R(j) = R(j) + B_{\ell, d_{i+1}-1}$ where
            $\ell = n-j+i+1$;
      else                              // a down fragment
        if $d_{i+1} \neq d_i + 1$ then
          for $j = i+1$ to $n$ do
            $R(j) = R(j) + (B_{\ell, d_i+1} - B_{\ell, d_{i+1}})$
            where $\ell = n-j+i+1$;
    return $R(n)$;
```

Fig. 4. The ranking algorithm.

Example 1. To perform Ranking$(0, 1, 2, 2, 0)$, we initially set $R(1) = R(2) = R(3) = R(4) = R(5) = 0$. When $i = 1$, the sons of x_1 are arranged in an up fragment because $R(1) = 0$ is even. Since $d_2 \neq 0$, we have the following updates:

$$R(2) = R(2) + B_{5,0} = 0 + 1 = 1,$$

$$R(3) = R(3) + B_{4,0} = 0 + 2 = 2,$$

$$R(4) = R(4) + B_{3,0} = 0 + 5 = 5,$$

$$R(5) = R(5) + B_{2,0} = 0 + 14 = 14.$$

When $i = 2$, the sons of x_2 are arranged in a down fragment because $R(2) = 1$ is odd. In this case, since $d_{i+1} = d_i + 1 = 3$, we keep $R(3)$, $R(4)$ and $R(5)$ to be unchanged. When $i = 3$, the sons of x_3 are arranged in an up fragment because $R(3) = 2$ is even. Since $d_4 \neq 0$, we have the following updates:

$$R(4) = R(4) + B_{5,1} = 5 + 2 = 7,$$

$$R(5) = R(5) + B_{4,1} = 14 + 5 = 19.$$

Finally, when $i = 4$, the sons of x_4 are arranged in a down fragment because $R(4) = 7$ is odd. Since $d_5 \neq d_4 + 1$, we perform the following update:

$$R(5) = R(5) + B_{5,3} - B_{5,1} = 19 + 4 - 1 = 22.$$

As a result, the algorithm outputs $R(5) = 22$. $\square$

Obviously, building the accumulation table using Eq. (3) requires $\mathcal{O}(n^2)$ time and space. Now, we will show that every element in the accumulation table can indeed be accessed in a constant time provided we make a preprocessing in advance. We observe that the numerator of a fractional number in the right hand side of Eq. (4) includes the product of two consecutive integers. Let $x = 2m + k + 1$ and $y = m + k + 2$. Also, define $f(x) = x(x+1)$ and $g(y) = y$. Then, Eq. (4) can be reformulated as follows:

$$B_{i-1,k} = \frac{B_{i,k}}{m+1} \cdot \frac{f(x)}{g(y)} \tag{7}$$

where $m = n - i + 1$. To obtain some requisite terms in the accumulation table from Eq. (7), we build a preprocessing table of size $(2n - 3) \times 2$ to store the terms of $f(z)$ and $g(z)$ for $3 \leqslant z \leqslant 2n - 1$, where the first term $z = 2m + k + 1 = m + k + 2 = 3$ is due to the initial conditions $m = 1$ (i.e., $i = n$) and $k = 0$. For example, for $n = 5$, the desired table is as follows.

z	$g(z)$	$f(z)$
3	3	12
4	4	20
5	5	30
6	6	42
7	7	56
8	8	72
9	9	90

Clearly, $f(z+1) = f(z) \cdot \frac{(z+2)}{z}$ and $g(z+1) = g(z)+1$. Thus, the table can be constructed in O(n) time. In addition, since every element in the last row of the accumulation table is easy to obtain (i.e., $B_{n,k} = k + 1$), we can compute all elements in the same column by using the preprocessing table.

For example, we have known $B_{4,1} = 5$. Since $i = 4$ (i.e., $m = 5 - 4 + 1 = 2$) and $k = 1$ in this stage, we have the terms $g(y) = y = m+k+2 = 5$ and $f(x) = x(x+1) = (6)(7) = 42$ where $x = 2m + k + 1 = 6$. Thus, using x and y as indices to search for the preprocessing table, we can computed $B_{3,1}$ as follows:

$$B_{3,1} = \frac{B_{4,2}}{m+1} \cdot \frac{f(6)}{g(5)} = \frac{5}{3} \cdot \frac{42}{5} = 14.$$

To continue the computation of $B_{2,1}$, since the current stage has changed i to 3 (i.e., $m = 3$) and $k = 1$, the indices x and y can be updated from their previous values by adding an extra offset 2 and 1, respectively. Thus, we have $x = 6 + 2 = 8$ and $y = 5 + 1 = 6$. Furthermore, we obtain

$$B_{2,1} = \frac{B_{3,1}}{m+1} \cdot \frac{f(8)}{g(6)} = \frac{14}{4} \cdot \frac{72}{6} = 42.$$

We repeat such a process until all required elements in the same column are calculated. Since building the preprocessing table needs $\mathcal{O}(n)$ time and space and the ranking algorithm can be run in $\mathcal{O}(n^2)$ time, we conclude the following.

Theorem 9. *For LW-sequences of binary trees with n internal nodes, determining the rank of a tree in a Gray-code oder can be done in $\mathcal{O}(n^2)$ time and $\mathcal{O}(n)$ space.*

V. Concluding Remarks

In this paper, we present a ranking algorithm of binary trees with n internal nodes encoded by LW-sequences in a Gray-code order. This algorithm can be run in $\mathcal{O}(n^2)$ time and $\mathcal{O}(n)$ space, respectively. As a future work, an interesting problem is to propose an unranking algorithm. To the best of our knowledge, so far no algorithm exists to solve such an unranking problem of Gray-code order for LW-sequences.

Acknowledgement

This work was partially supported by Ministry of Science and Technology of Taiwan under Grants MOST104-2221-E-262-005, NSC102-2221-E-141-001-MY3 and MOST104-2221-E-141-007.

References

[1] A. Ahmadi-Adl, A. Nowzari-Dalini, and H. Ahrabian, "Ranking and unranking algorithms for loopless generation of t-ary trees," *Logic J. IGPL*, vol. 19, pp. 33–43, 2011.

[2] D.E. Knuth, *The Art of Computer Programming: Vol. 4 Fascicle 4A – Generating All Trees*. Addison-Wesley, 2005.

[3] J.F. Korsh and P. LaFollette, "A loopless Gray code for rooted trees," *ACM Trans. Algorithms*, vol. 2, pp. 135–152, 2006.

[4] 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.

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

[6] J. Pallo, "Enumerating, ranking and unranking binary trees," *Comput. J.*, vol. 29, pp. 171–175, 1986.

[7] D. Roelants van Baronaigien and F. Ruskey, "A Hamiltonian path in the rotation lattice of binary trees," *Congr. Numer.*, vol. 59, pp. 313–318, 1987.

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

[9] D. Roelants van Baronaigien, "A loopless algorithm for generating binary tree sequences," *Inform. Process. Lett.*, vol. 39, pp. 189–194, 1991.

[10] V. Vajnovszki, "On the loopless generation of binary tree sequences," *Inform. Process. Lett.*, vol. 68, pp. 113–117, 1998.

[11] V. Vajnovszki, "Generating a Gray code for P-sequences," *J. Math. Model. Algorithms*, vol. 1, pp. 31–41, 2002)

[12] Ro-Yu Wu, Jou-Ming Chang, and Yue-Li Wang, "A linear time algorithm for binary tree sequences transformation using left-arm and right-arm rotations," *Theor. Comput. Sci.*, vol. 355, pp. 303–314, 2006.

[13] Ro-Yu Wu, Jou-Ming Chang, H.-C. Chan, and Kung-Jui Pai, "A loopless algorithm for generating multiple binary tree sequences simultaneously," *Theor. Comput. Sci.*, vol. 556, pp. 25–33, 2014.

[14] Ro-Yu Wu, Jou-Ming Chang, An-Hang Chen, and Ming-Tat Ko, "Ranking and unranking of non-regular trees in Gray-code order," *IEICE Trans. Fund.*, vol. E96-A, pp. 1059–1065, 2013.

[15] Ro-Yu Wu, Jou-Ming Chang, An-Hang Chen, and Chun-Liang Liu, "Ranking and unranking t-ary trees in a Gray-code order," *Comput. J.*, vol. 56, pp. 1388–1395, 2013.

[16] Ro-Yu Wu, Jou-Ming Chang, and Yue-Li Wang, "Loopless Generation of non-regular trees with a prescribed branching sequence," *Comput. J.*, vol. 53, pp. 661–666, 2010.

[17] Ro-Yu Wu, Cheng-Hsien Hsu, and Jou-Ming Chang, "Loopless algorithms for listing Zaks' sequences in Gray-code order," *J. Internet Tech.*, vol. 15, pp. 679–684, 2014.

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

[19] D. Zerling, "Generating binary trees using rotations," *J. Assoc. Comput. Mach.*, vol. 32, pp. 694–701, 1985.