

Ranking and Unranking Algorithm for Neuronal Trees in B-order

Mahdi Amani^{1,2} and Abbas Nowzari-Dalini¹

¹Department of Computer Science, School of Mathematics, Statistics, and Computer Science, Colleague of Science, University of Tehran, Tehran, Iran

²Dipartimento di Informatica, Università di Pisa, Pisa, Italy
E-mail: m_amani@di.unipi.it, nowzari@ut.ac.ir

Received 7 September 2015; accepted 11 November 2015

ABSTRACT

In this paper, we present two new ranking and unranking algorithms for neuronal trees in B-order. These algorithms are based on a generation algorithm which is given for integer sequences corresponding to neuronal trees by Pallo. A neuronal tree is a rooted tree with n external nodes (leaves) whose internal nodes have at least two children. These trees are used in computational neuroscience for modeling the connections between neurons in brain, and are also called neuronal dendritic trees. Up to our knowledge no other ranking and unranking algorithms are given for integer sequences corresponding to neuronal trees in B-order. The time complexity of the presented ranking and unranking algorithms for neuronal trees with n leaves are $O(n)$ and $O(n \log n)$, respectively.

Keywords: Tree Generation, Ranking, Unranking, B-order, neuronal tree, dendritic tree.

1. Introduction

Trees are one of the most important basic and simple data structures for organizing information in computer science. Trees have many applications including database generation, decision table programming, analysis of algorithms, string matching [15], switching theory, theoretical VLSI circuit design [29], image processing [25, 28], maintaining data [19], and as auxiliary structures for compressing data [13]. Trees are also widely used for showing the organization of real world data such family/genealogy trees [23], taxonomies, and modeling of the connections between neurons of the brain in computational neuroscience [6, 7, 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 and analyze algorithm complexity, and prove the correctness of an algorithm [15]. Therefore, the problem of generating trees has been thoroughly investigated in the literature and many papers have been published which deal with the generation of all trees. For example, we can mention the generation of binary trees in [2, 11, 30, 32], k -ary trees in [3, 10, 12, 14, 24, 34, 35], trees with n nodes and m leaves in [21, 27], neuronal trees in [4, 22, 30], non-regular trees in [33], AVL trees in [16], and spanning trees in [8, 20].

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 [35], and the orderings on the sequences are lexicographical [35], cool-lex [9], and Gray code [14, 17].

Beside the generation algorithm for trees, ranking and unranking algorithms are also important in the concept of tree generation [1, 24, 33, 35]. Given a specific order on the set of 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 sequence) in this order. The reverse operation of ranking is called unranking; it 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.

In this paper, we consider neuronal (dendritic) trees. Neuronal trees of size n are rooted trees with n external nodes (leaves) which internal nodes have at least two children. These trees are known with regard to their number of leaves [22]. Neuronal trees are used for modeling the ‘dendrites of a nerve cell’ in the brain [5, 22].

Generation of neuronal trees is first studied by Pallo [22]. He introduced an integer sequence codeword for encoding these trees and presented an efficient algorithm for generating the integer codewords of all neuronal trees with n external nodes. The corresponding trees are generated in B-order and the generation algorithm has $O(n)$ worst case time complexity. He presented a generation algorithm, but no ranking and unranking algorithms were presented.

An encoding of length n over six letters for a neuronal tree with n leaves and a generation algorithm on this encoding in A-order are given by Vajnovszki [30]. The presented generation algorithm has $O(n)$ time complexity in the worst case and $O(\log n)$ average time complexity. He also presented an unranking algorithm with the time complexity of $O(n^2)$ but no ranking algorithm was presented. After that, a new encoding on three letters for neuronal trees with n leaves is presented by Amani *et al.* [4]. The size of encoding is equal to the number of nodes of the tree (less than $2n$ and greater than $n+1$). They also presented a generation algorithm on this encoding with constant average time and $O(n)$ time complexity in the worst case. In this algorithm, the trees were generated in A-order. Due to the given encoding, both ranking and unranking algorithms were also presented with $O(n)$ and $O(n \log n)$ time complexity, respectively.

To our best knowledge, no ranking and unranking algorithms are designed for neuronal tree in B-order. In this paper, we present two new ranking and unranking algorithms for codewords corresponding to neuronal tree with n external nodes generated in B-order based on the Pallo [22] generation algorithm. The time complexity of the ranking algorithm is $O(n)$ and it is $O(n \log n)$ for the unranking algorithm.

The remaining of the paper is organized as follows. Section 2 introduces the definitions and notions that are used further. The Pallo [22] encoding and generation algorithm for neuronal trees in B-order are presented in Section 3. Based on Pallo generation algorithm, the ranking and unranking algorithms are given in Section 4. Finally, some concluding remarks are offered in Section 5.

2. Definitions

Formally, a *rooted tree* is a connected and undirected graph without any cycle with a

Ranking and Unranking Algorithm for Neuronal Trees in B-order

special node called the *root*. In a rooted tree, every node is connected to the root by exactly one path. For two connected nodes, the node nearest the root is called the *parent* and the other node called its *child*. Each child of a node is the root of a tree called *subtree* of this node. A rooted tree where the children of each node have a designated order is called *ordered tree*. The children of the same parent are known as *siblings*, and the *degree* of a node is defined as the number of its children. An *external node* (or *leaf*) is a node that has no children, and the other nodes (that do have children) are known as *internal nodes*.

According to the structure of neurons, dendrites of neurons have splits such that each branch in the splits is connected to at least two other branches except the terminal ones. Therefore, the term *neuronal tree* is used to refer to a tree in which each internal node has at least two children; in other words a neuronal tree is a tree whose nodes are either leaves or have at least 2 children [22, 30]. These trees are known with regard to their number of leaves. For example the tree given in Fig. 1 can be regarded as a neuronal tree with $n = 13$ leaves. Recall from [22], that in dendritic terminology, the root of a neuronal tree is taken to be the axon hillock and the external nodes are the tips of the terminal segments. The order of magnitude of branching at a node may be described as dichotomous if the degree of that node is 2, trichotomous if the degree is 3, and so on [7]. Formally, a neuronal tree can be defined as follows.

Definition 1. A neuronal tree T is defined as a finite set of one or more nodes such that:

1. T has a distinguished node r , called *root* of this tree, and if T has more than one node, then r is connected to $j \geq 2$ neuronal trees T_1, T_2, \dots, T_j and each tree T_i ($1 \leq i \leq j$), is called the *subtree* of T .
2. The root of T_i ($1 \leq i \leq j$) is considered as a *child* of r .
3. T_1 is the *leftmost subtree*, and its root is the *leftmost child* of r .
4. T_j is the *rightmost subtree* and its root is the *rightmost child* of r .

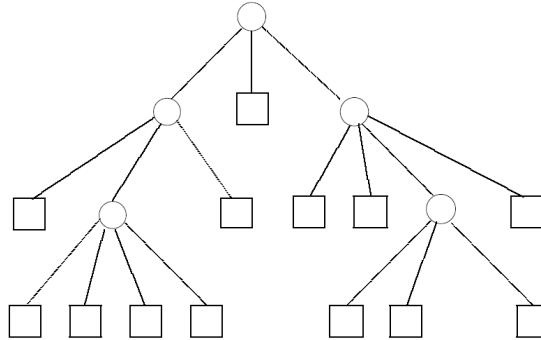


Figure 1: A sample neuronal tree with 5 internal nodes and 13 external nodes.

Let S_n denotes the set of neuronal trees with n external nodes. The number of trees in S_n is denoted by s_n (i.e., $s_n = |S_n|$); it corresponds to the well-known n^{th} Schröder number [26] and can be computed by a linear recurrence formula.

Theorem 1. [26] *The Schröder number counts the trees of the set S_n as follows.*

$$\begin{cases} S_n = \frac{3(2n-3)S_{n-1} - (n-3)S_{n-2}}{n}, & (\text{for } n > 2), \\ S_1 = S_2 = 1. \end{cases}$$

It is also proved in [30] that $S_n > 5^n$ for $n > 57$ and $S_n < 6^n$ for $n > 1$.

As mentioned, any generation algorithm imposes an ordering on the set of trees. In these algorithms trees are encoded as integer sequences and then these sequences are generated with a certain order and consequently their corresponding trees are generated in a specific order. Two such natural orderings are A-order and B-order [30, 31, 35] which are defined for neuronal trees as follows.

Definition 2. Let T and T' be two neuronal trees and $k = \max\{\deg(T), \deg(T')\}$, we say that T is less than T' in B-order ($T \prec_B T'$), iff

- $\deg(T) < \deg(T')$ or
- $\deg(T) = \deg(T')$, and $\exists_{1 \leq i \leq k} : \forall_{j \in \{1, 2, \dots, (i-1)\}} : T_j =_B T'_j$, and $T_i \prec_B T'_i$,

where $\deg(T)$ is defined as the degree of root of the tree T .

Definition 3. Let T and T' be two neuronal trees and $k = \max\{\deg(T), \deg(T')\}$, we say that T is less than T' in A-order ($T \prec_A T'$), iff

- $|T| < |T'|$, or
- $\deg(T) = \deg(T')$, and $\exists_{1 \leq i \leq k} : \forall_{j \in \{1, 2, \dots, (i-1)\}} : T_j =_A T'_j$, and $T_i \prec_A T'_i$,

where $|T|$ (size of T) is defined as the number of leaves in the tree T .

The most well-known ordering for integer sequences is the lexicographic ordering that is defined as follows.

Definition 4. Two integer sequences $v = (v_1, v_2, \dots, v_n)$ and $v' = (v'_1, v'_2, \dots, v'_m)$ are in lexicographic order (denoted by $v < v'$), if there exists $1 \leq i \leq \min(n, m)$, such that

1. $v_j = v'_j$ for all $1 \leq j < i$,
2. $v_i < v'_i$.

The Pallo's generation algorithm [22], generates the integer sequences corresponding to neuronal trees in lexicographical ordering, and their corresponding trees are generated in B-order.

As mentioned before, besides the generation algorithm for trees, ranking and unranking algorithms are also important in the concept of tree generation [10, 24, 35]. Let us consider an arbitrary class of trees of size n , the elements of this set can be listed based on any defined ordering such as A-order or B-order. With respect to the ordering (e.g.

Ranking and Unranking Algorithm for Neuronal Trees in B-order

A-order or B-order), the ‘position’ of tree T in that class is called *rank* of T , the *rank function* determines the rank of T ; the inverse operation of ranking is *unranking*, for a position r , the *unrank function* gives the tree T corresponding to this position.

3. Encoding and generation algorithm

In this section, we review the neuronal trees encoding and generation algorithm for these trees presented by Pallo in [22]. As mentioned, the main point in generating trees is to choose a suitable encoding to represent them, and instead of generating trees, their corresponding codewords are generated. Pallo encoded each neuronal tree in the set S_n by an integer sequence as follows [22].

Definition 5. *Given a neuronal tree T with n external nodes, the S-sequences $s = \{s_1, s_2, \dots, s_\ell\}$ corresponding to T is obtained as follows. Each internal node of tree T is labeled with its degree minus one and each external node with zero, then the labels are listed in pre-order traversal of T as sequence s .*

For example, the S-sequence corresponding to the neuronal tree T shown in Fig. 1 is the sequence $s = \{2, 2, 0, 3, 0, 0, 0, 0, 0, 3, 0, 0, 2, 0, 0, 0, 0\}$. An S-sequence s is called *feasible* if there is a tree $T \in S_n$ such that s is the S-sequences corresponding to T . We also denote the S-sequence corresponding to T by $s(T)$. The feasibility of an S-sequence corresponding to a neuronal tree is studied in the following theorem [22].

Theorem 2. *An integer sequence $s = \{s_1, s_2, \dots, s_\ell\}$ is a feasible codeword of a neuronal tree iff $s_\ell = 0$, and $\forall k \in [1, \ell - 1]$:*

$$\sum_{i=1}^k s_i > |\{j \in [1, k] : s_j = 0\}|.$$

Clearly, there is a one to one correspondence between a neuronal tree and a feasible codeword. With respect to the above theorem, the length of the feasible sequences alters between $n + 1$ to $2n - 1$. In the corresponding lexicographic ordering of the S-sequences, the first S-sequence is

$$\underbrace{\{1, 0, 1, 0, \dots, 1, 0, 0\}}_{2n-2}$$

with length $2n - 1$, and the last is

$$\{n-1, \underbrace{0, 0, 0, \dots, 0}_n\}$$

with length $n + 1$. Actually, the first sequence corresponds to a right-chain binary tree with $n - 1$ internal nodes and n external nodes, and the last sequence corresponds to a n -ary tree with one internal node and n external nodes. Therefore, we can present the following theorem [22].

Theorem 3. *Given two neuronal trees T and T' in S_n , T is less than T' in B-order*

$(T \prec_B T')$, iff $s(T)$ is lexicographically less than $s(T')$ (i.e., $s(T) < s(T')$).

Based on the above theorem, the generation of S-sequences in lexicographical ordering corresponds to the generation of neuronal trees in B-order. As an example, a list of 11 neuronal trees with $n = 4$ external nodes in B-order, and their corresponding S-sequences in lexicographical order are illustrated in Fig. 2.

The Pallo's generation algorithm [22] returns the successor of a given sequence $s = \{s_1, s_2, \dots, s_\ell\}$ with length $\ell = O(n)$. In this algorithm, the sequence s is scanned from right to left, the first non-zero element is obtained, and its position is assigned to k . If $k = 1$, then this sequence corresponds to the last sequence and there is no successor. Otherwise, the successor is computed as follows. First, the length of the new sequence is evaluated and this length is kept in ℓ . Later, the $(k - 1)^{\text{th}}$ element is incremented and a subsequence corresponding to a right-chain subtree with size $(s_k - 1)$ is replaced by the last $(s_k - 1)$ elements in the sequence and the elements from k to $\ell - 2s_k + 1$ are set to zero. In fact, this process is similar to the replacement of the right-most child of the node k by a right-chain binary subtree, and the appropriate number of external nodes is added as the first children of the k^{th} node.

The pseudocode for Pallo's algorithm [22] is presented in Fig. 3. The algorithm generates each sequence in constant average time. The time complexity of this algorithm in the worst case is $O(n)$.

4. Ranking and unranking algorithms

By having a generation algorithm in a specific order, the ranking of algorithm is desired. To represent a neuronal tree as an integer, we need to know its index in the exhaustive generated list by the generation algorithm. This index is called the rank of the neuronal tree (or corresponding S-sequence); i.e., the rank of an S-sequence corresponding to a neuronal tree with respect to some ordering is the number of previously generated codewords in that ordering. This is achieved by ranking algorithm: The ranking algorithm receives an S-sequence as the input and returns the index of the S-sequence. The reverse operation of the ranking is called unranking. An unranking algorithm determines the S-sequence corresponding to a neuronal tree having a particular rank. As mentioned, for neuronal trees in B-order, no ranking and unranking algorithms are presented in the literature. In this section, ranking and unranking algorithms for neuronal trees in B-order based on S-sequences are given.

Ranking and Unranking Algorithm for Neuronal Trees in B-order

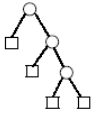
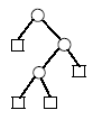
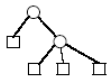

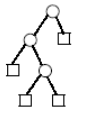
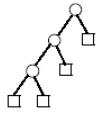





No.	1	2
tree		
S-sequence	1010100	1011000
3	4	5
		
102000	1100100	1101000
6	7	8
		
1110000	120000	200100
9	10	11
		
201000	210000	30000

Figure 2: List of the trees in S_4 in B-order and their corresponding S-sequences.

Mahdi Amani and Abbas Nowzari-Dalini

```

Procedure Pallo-Gen-Seq ( $s$ : S-seq;  $\ell$ : Integer) ;
Var  $i, j, t, q$  : Integer ;
Begin
   $i := \ell$  ;
  While ( $i > 1$ ) And ( $s_i = 0$ ) Do
     $i := i - 1$  ;
  If ( $i = 1$ ) Then Exit ;
  If ( $s_{i-1} = 0$ ) Then  $q := \ell - i + 1$  ;
  Else  $q := \ell - i$  ;
   $t := s_i - 1$  ;  $s_{i-1} := s_{i-1} + 1$  ;
   $\ell := i + t + q - 1$  ;  $s_\ell := 0$  ;
  For  $j := 1$  To  $t$  Do Begin
     $s_{\ell-2j} := 1$  ;  $s_{\ell-2j+1} := 0$  ;
  End ;
  For  $j := i$  To  $\ell - 2t - 1$  Do
     $s_j := 0$  ;
End ;

```

Figure 3: Pallo S-sequences generation algorithm.

Ranking and unranking algorithms usually use a precomputed table of the number of a subclass of given trees with some specified properties to achieve efficient time complexities; these precomputations will be done only once and stored in a table for further use [4, 16, 27, 33]. For designing the ranking and unranking algorithms of neuronal trees, we need some theorems and definitions. The following theorem presents another way for calculating the cardinality of S_n .

Theorem 4. For the cardinality of the set of neuronal trees with n leaves S_n , we have:

$$S_n = 2 \sum_{i=1}^{n-2} S_i S_{n-i} + S_{n-1} S_1, \quad S_1 = S_2 = 1.$$

Proof. See [4].

Theorem 5. Let $S_{n,m}$ be the number of neuronal trees with n leaves whose the root has degree equal to m ($2 \leq m \leq n$). Then we have:

1. if $m = 2$, then $S_{n,m} = S_{n,2} = \sum_{i=1}^{n-1} S_i S_{n-i}$.
2. if $m > 2$, then $S_{n,m} = \sum_{i=1}^{n-m+1} S_i S_{n-i,m-1}$.

Proof. Let T be a neuronal tree with n leaves and the root of degree m , then we have two cases.

1. If $m = 2$, then T has two children T_1 and T_2 , if $|T_1| = i$ then the number of such trees is equal to $S_i S_{n-i}$. Since i can change from 1 to $n - 1$, we have:

Ranking and Unranking Algorithm for Neuronal Trees in B-order

$$S_{n,2} = \sum_{i=1}^{n-1} S_i S_{n-i}.$$

2. If $m > 2$, then T has m subtrees T_1, T_2, \dots, T_m . If $|T_1| = i$, then the number of possible cases for T_1 is equal to S_i , and the number of cases for the other subtrees (T_2, \dots, T_m) is equal to $S_{n-i, m-1}$ (by ignoring T_1 , the remaining subtrees form a neuronal tree with $n-i$ leaves and with a root of degree $m-1$). Therefore we have $S_i S_{n-i, m-1}$ trees. In this case i can change from 1 to $n-m+1$, so the number of neuronal trees is equal to:

$$\sum_{i=1}^{n-m+1} S_i S_{n-i, m-1}.$$

Hence, the proof is complete.

Theorem 6. Let $N_{n,m}$ be the number of neuronal trees with n leaves whose first child has degree less than or equal to m ($2 \leq m \leq n$). Then we have:

$$N_{n,m} = \sum_{i=2}^m (2 \sum_{j=i}^{n-2} S_{j,i} S_{n-j} + S_{n-1,i}).$$

Proof. Let T be a neuronal tree satisfying the above condition, and let the first child of T be T_1 . Let $\deg(T_1) = i, 2 \leq i \leq m$, and the number of leaves in T_1 be j , j can change between i and $n-1$.

First, let us assume $j = n-1$, we will have a neuronal tree whose first child (T_1) has degree i and $j = n-1$ leaves, so T_1 has just one brother which is a leaf. Therefore, the number of such trees is equal to $S_{n-1,i} S_1 = S_{n-1,i}$.

Now, let us assume $i \leq j \leq n-2$, we will have a neuronal tree whose first child has degree i and j leaves and other children (brothers of T_1) have all together $n-j$ leaves. We have two different cases:

1. T has just two children T_1 with j leaves and T_2 with $n-j$ leaves. T_1 is a neuronal tree with the root of degree i and j leaves and T_2 can be any neuronal tree with $n-j$ leaves. Therefore, the number of such trees is equal to:

$$\sum_{j=i}^{n-2} S_{j,i} S_{n-j}.$$

2. T has $k > 2$ children T_1, T_2, \dots, T_k such that T_1 has j leaves and T_2, T_3, \dots, T_k have all together $n-j$ leaves. Since $k > 2$, if we ignore T_1 , a neuronal tree with $n-j$ leaves remains. T_1 is also a neuronal tree with the root of degree i and j leaves. Therefore the number of such trees is again equal to:

$$\sum_{j=i}^{n-2} S_{j,i} S_{n-j}.$$

Therefore in total we have:

$$N_{n,m} = \sum_{i=2}^m (2 \sum_{j=i}^{n-2} S_{j,i} S_{n-j} + S_{n-1,i}).$$

Hence, the proof is complete.

For the ranking and unranking algorithms we need to compute in advance $S_n, S_{n,m}$, and $N_{n,m}$. These computation can be performed in time $O(n)$ and $O(n^2)$. Now, with regard to the above theorems and definitions we can present a new formula to compute the rank.

Theorem 7. Let T be a neuronal tree with n leaves whose subtrees are defined by T_1, T_2, \dots, T_j , and for $1 \leq i \leq j$: $|T_i| = n_i$, $\deg(T_i) = d_i$, and $\sum_{i=1}^j n_i = n$, we have:

$$\begin{cases} \text{Rank}(T, 1) = 1, \\ \text{Rank}(T, n) = \sum_{i=1}^j (N_{n-\sum_{k=1}^{i-1} n_k, d_i-1} + 2(\text{Rank}(T_i, n_i) - 1) S_{n-\sum_{k=1}^i n_k}). \end{cases}$$

Proof. One way to compute the rank of tree T is to enumerate the number of trees generated before T .

The number of neuronal trees with n leaves whose first subtree is smaller than T_1 is equal to:

$$N_{n, d_1-1} + 2(\text{Rank}(T_1, n_1) - 1) S_{n-n_1},$$

and the number of neuronal trees with n leaves whose first subtree is equal to T_1 but the second subtree is smaller than T_2 is equal to:

$$N_{n-n_1, d_2-1} + 2(\text{Rank}(T_2, n_2) - 1) S_{n-(n_1+n_2)},$$

similarly, the number of neuronal trees with n leaves whose first $(i-1)$ subtrees are equal to T_1, T_2, \dots, T_{i-1} and the i^{th} subtree is smaller than T_i is equal to:

$$N_{n-\sum_{k=1}^{i-1} n_k, d_i-1} + 2(\text{Rank}(T_i, n_i) - 1) S_{n-\sum_{k=1}^i n_k}.$$

Therefore, regarding the above enumeration we have:

$$\begin{cases} \text{Rank}(T, 1) = 1, \\ \text{Rank}(T, n) = \sum_{i=1}^j (N_{n-\sum_{k=1}^{i-1} n_k, d_i-1} + 2(\text{Rank}(T_i, n_i) - 1) S_{n-\sum_{k=1}^i n_k}). \end{cases}$$

So the proof is complete.

Ranking and Unranking Algorithm for Neuronal Trees in B-order

```

Function CalculateL (Beg: Integer; Var Fin: Integer);
Var Tmp, Sum, Cur : Integer;
Begin
  Cur := Beg;
  Sum := 0;
  For i := 1 To C[Beg] + 1 Do Begin
    If C[Cur] = 0 Then Begin
      L[Cur] := 1;
      Cur := Cur + 1;
    End
    Else Begin
      L[Cur] := CalculateL(Cur, Tmp);
      Cur := Cur + Tmp;
    End;
    Sum := Sum + L[Cur];
  End;
  Fin := Cur + 1;
  Return(Sum);
End;

```

Figure 4: Algorithm for calculating the number of leaves in a subtree.

For computing the rank of an S-sequence stored in an array C , we need an auxiliary array $L[i]$ which keeps the number of leaves in the subtree whose root is labeled by $C[i]$ and corresponds to n_i in the above formula. This array is computed by the algorithm *CalculateL* given in Fig. 4. In this algorithm “*Beg*” is a variable that shows the position of the first character in the array C (when this algorithm is called for first time, the initial value for “*Beg*” is 1), and “*Fin*” is a “call by reference variable” that returns the position of the last leaf in the subtree whose root is labeled by $C[Beg]$. We need to emphasize that “*Tmp*” is just a local variable used to store the returned value of *Fin* after each recursive call. This algorithm is recursive and in each call, for an S-sequence stored in global array C , the number of leaves of the subtree rooted at $C[Beg]$ with the last leaf in $C[Fin]$ is calculated. This algorithm is executed just once before calling the ranking algorithm.

Considering Theorem 7 and the algorithm *CalculateL*, the ranking algorithm is given in Fig. 5. In this algorithm the variables “*Beg*” and “*Fin*” are similar to the variables used in the algorithm *CalculateL*, and “*Beg*” is initially set to 1. As we can see, this algorithm computes the rank of S-sequence corresponding to a neuronal tree using auxiliary array L and S-sequence array C ; this algorithm recursively returns two important values: first, the *Rank* of the subtree rooted at *Beg*, and second, call-by-reference variable *Fin* (position of the last leaf in the subtree whose root is labeled by $C[Beg]$).

```

Function Rank(Beg: Integer; Var Fin: Integer);
Var R, N, Point, PointFin, Leaves, n: Integer;
Begin
    n := L[Beg];
    If (n = 1) Then Begin
        Fin := Beg;
        Return(1);
    End
    Else Begin
        Point := Beg + 1; R := 0; Leaves := 0;
        While (Leaves < n) Do Begin
            R := R + Nn-Leaves, C[Point]-1
                + 2(Rank(point, PointFin)-1) × Sn-Leaves-L[Point];
            Leaves := Leaves + L[Point];
            If L[Point] = 1 Then
                Point := Point + 1;
            Else
                Point := PointFin + 1;
            End;
        End;
        Return(R + 1);
    End;
End;

```

Figure 5: Ranking algorithm.

Now the time complexity of this algorithm is discussed. Obviously the time complexity of the procedure “*CalculateL*” (shown in Fig. 4), which computes the number of leaves in each subtree is $O(n)$. This algorithm is executed just once before calling the ranking algorithm and it has no more time effects on the ranking algorithm, also the arrays S_n , $S_{n,m}$, and $N_{n,m}$ were precomputed in advance. Therefore we should calculate the time complexity of the ranking algorithm given in Fig. 5. Let T be a neuronal tree with n leaves whose subtrees are defined by T_1, T_2, \dots, T_j and for $1 \leq i \leq j : |T_i| = n_i$ and $\sum_{i=1}^j n_i = n$, and let $T(n)$ be the time complexity of the ranking algorithm given in Fig. 5, then we have: $T(n) = T(n_1) + T(n_2) + \dots + T(n_j) + \alpha j$ (where α is a constant value and αj is the time complexity of the non-recursive parts of the algorithm). In [4 (see the proof for Theorem 5)], Amani *et.al*, with a simple induction, show that for the above recursive formula, $T(n) = O(n)$. Therefore our ranking algorithm has the time complexity of $O(n)$ in the worst case.

If a and b are integer numbers, let $(a \text{ div } b)$ and $(a \text{ mod } b)$ denote *integer division* and *remainder* of the division of a and b , respectively ($a = (a \text{ div } b) \times b + (a \text{ mod } b)$). Before giving the description of the unranking algorithm we need to define two new operators. We define $(a \text{ div}^+ b)$ and $(a \text{ mod}^+ b)$ as follows.

Ranking and Unranking Algorithm for Neuronal Trees in B-order

1. If $b \mid a$, then $(a \operatorname{div}^+ b) = (a \operatorname{div} b) - 1$, and $(a \operatorname{mod}^+ b) = b$.
2. Otherwise, $(a \operatorname{div}^+ b) = (a \operatorname{div} b)$, and $(a \operatorname{mod}^+ b) = (a \operatorname{mod} b)$.

In the unranking algorithm, for a given rank R , we have to find an S-sequence C corresponding to T such that $\operatorname{Rank}(T, n) = R$. For unranking R , in each step of the algorithm we need to find the minimum $j > 0$ such that $N_{n,j} \geq R$. Then the number of leaves in the first subtree of T is $(j-1)$. Then we have to build the S-sequence for the first subtree recursively and update R . Considering the above discussion, the unranking algorithm is given in Fig. 6. In this algorithm, R is the input, Beg is a variable used to show the position of the first character in the global array C and initially is set to 1. The generated S-sequence is hold in the array C . The variable n is the number of leaves of the neuronal tree corresponding to C . As mentioned before, it is assumed that the precomputed arrays $S_n, S_{n,m}$, and $N_{n,m}$ are computed and stored in advance, therefore, with regard to the non recursive and recursive parts of the unranking algorithm, for a given neuronal tree T whose subtrees are defined by T_1, T_2, \dots, T_j , if the time complexity for unranking algorithm is shown by $T(n)$, we have:

$$T(n) = O(\log n_1 + \log n_2 + \dots + \log n_j) + T(n_1) + T(n_2) + \dots + T(n_j).$$

In [4 (see the proof for Theorem 6)], it has been proved that for the above recursive formula, $T(n) = O(n \log n)$. Therefore the time complexity of our presented unranking algorithm is $O(n \log n)$.

5. Conclusions

In this paper, we have introduced two new ranking and unranking algorithms for neuronal trees in B-order. The generation algorithm for neuronal trees in B-order is presented by Pallo. The time complexities of the presented ranking and unranking algorithms are $O(n)$ and $O(n \log n)$, respectively.

Acknowledgements

This research was partially supported by University of Tehran and Italian Ministry of Education, University, and Research (MIUR) under PRIN 2012C4E3KT national research project AMANDA.

Mahdi Amani and Abbas Nowzari-Dalini

```

Function UnRank (R, Beg, n: Integer);
Var Point, Deg, j: Integer ;
Begin
  If ((n = 0) or (R = 0)) Then
    Return (Beg - 1)
  Else Begin
    If (n = 1) Then Begin
      C[Beg] := 0;
      Return (Beg);
    End
  Else Begin
    Point := Beg + 1;
    Deg := 0;
    While (n > 1) Do Begin
      Deg := Deg + 1;
      j := min{i|Nn,i ≥ R};
      R := R - Nn,j-1;
      Point := UnRank ((div+(R, 2Sn-j) + 1), Point, j) + 1;
      R := mod+(R, 2Sn-j);
      n := n - j;
    End;
    C[Beg] := Deg;
    Return(Point - 1);
  End;
End;

```

Figure 6: Unranking algorithm.

REFERENCES

1. 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.
2. H. Ahrabian and A. Nowzari-Dalini, On the generation of binary trees in A-order, *International Journal of Computer Mathematics*, 71 (1999) 351-357.
3. H. Ahrabian and A. Nowzari-Dalini, Parallel Generation of t-ary trees in A-order, *Computer Journal*, 50 (2007) 581-588.
4. Amani, Mahdi, Abbas Nowzari-Dalini, and Hayedeh Ahrabian, Generation of Neuronal Trees by a New Three Letters Encoding, *Computing and Informatics*, 33(6) (2015) 1428-1450.
5. R. Alberich, G. Cardona, F. Rosselló and G. Valiente, An algebraic metric for phylogenetic trees, *Applied Mathematics Letters*, 22 (2009) 1320-1324.
6. S. Berger and L. Tucker, Binary tree representation of three-dimensional, recosytruted neuronal trees: a simple, efficient algorithm, *Computer Methods and Programs in Biomedicine*, 23 (1986) 231-235.
7. M. Berry and P. Bradley, The application of network analysis to the study of branching patterns of large dendritic fields, *Brain Research*, 109 (1976) 111-132.
8. E. Boros, K. Borys, V. Gurvich and G. Rudolf, Generating 3-vertex connected spanning subgraphs, *Discrete Mathematics*, 308 (2008) 6285-6297.
9. S. Durocher, P.C. Li, D. Mondal, F. Ruskey, and A. Williams, Cool-lex order and

Ranking and Unranking Algorithm for Neuronal Trees in B-order

- k-ary Catalan structures, *Journal of Discrete Algorithms*, 16 (2012) 287-307.
10. M.C. Er, Efficient generation of k-ary trees in natural order, *Computer Journal*, 35 (1992) 306-308.
 11. S.K. Ghosh, J. Ghosh, and R.K. Pal, A new algorithm to represent a given k-ary tree into its equivalent binary tree structure, *Journal of Physical Sciences*, 12 (2008) 253-264.
 12. S. Heubach, N. Li, and T. Mansour, Staircase tilings and k-Catalan structures, *Discrete Mathematics*, 308 (2008), 5954-5964.
 13. J. Katajainen and E. Makinen, Tree compression and optimization with application, *International Journal of Foundations of Computer Science*, 1 (1990) 425-447.
 14. J.F. Korsh and P. LaFollette, Loopless generation of Gray codes for k-ary trees, *Information Processing letters*, 70 (1999) 7-11.
 15. D.E. Knuth, *The Art of Computer Programming, Vol.1: Fundamental Algorithms, 2nd Ed.*, Addison-Wesley, Reading, MA, 1973.
 16. L. Li, Ranking and unranking AVL trees, *SIAM Journal of Computing*, 15 (1986) 1025-1035.
 17. J. Lucas, D. R.V. Baronaigien, and F.Ruskey, On rotations and the generation of binary trees, *Journal of Algorithms*, 15 (1993) 343-366.
 18. S. Mandal and M. Pal, A sequential algorithm to solve next-to-shortest path problem on circular-arc graphs, *Journal of Physical Sciences*, 10 (2006) 201-217.
 19. O.O. Olugbenga, E.F. Adebisi, S. Fatumo and A.Dawodu, PQ trees, consecutive ones problem and applications, *International Journal of Natural and Applied Sciences*, 4 (2008) 262-277.
 20. S.K. Pal, S. Sen, and P. Manna, Spanning tree based on analytical perspective of degree sequence, *Journal of Physical Sciences*, 13 (2009) 209-216.
 21. J. Pallo, Generating trees with n nodes and m leaves, *International Journal of Computer Mathematics*, 21 (1978) 133-144.
 22. J. Pallo, A simple algorithm for generating neuronal dendritic trees, *Computer Methods and Programs in Biomedicine*, 33 (1990) 165-169.
 23. K.D. Queiroz and J. Gauthier, Phylogeny as a central principle in taxonomy: Phylogenetic definitions of taxon names, *Systematic Zoology*, 39 (1990) 307-322.
 24. F. Ruskey, Generating t-ary trees lexicographically, *SIAM Journal of Computing*, 7 (1978) 424-439.
 25. H. Samet and R.E. Webber, Hierarchical data structures and algorithms for computer graphics, *IEEE Computer Graphics & Applications*, 8 (1988) 67-75.
 26. E. Schröder, Vier combinatorische problem, *Zeitschrift für Angewandte Mathematik und Physik*, 15 (1870) 361-376.
 27. E. Seyedi-Tabari, H. Ahrabian and A.Nowzari-Dalini, A new algorithm for generation of different types of RNA, *International Journal of Computer Mathematics*, 87 (2010) 1197-1207.
 28. I.P. Stewart, Quadrees: storage and scan conversion, *Computer Journal*, 29 (1986) 60-75.
 29. T. Uehara and W.M. Cleemput, Optical layout of cmos functional arrays, *IEEE Transaction on Computers*, 7 (1981) 305-312.
 30. V. Vajnovszki, Listing and random generation of neuronal trees coded by six letters, *The Automation, Computers, and Applied Mathematics*, 4 (1995) 29-40.

Mahdi Amani and Abbas Nowzari-Dalini

31. V. Vajnovszki and J. Pallo, Generating binary trees in A-order from codewords defined on four-letter alphabet, *Journal of Information and Optimization Science*, 15 (1994) 345-357.
32. R. Wu, J. Chang and Y. Wang, A linear time algorithm for binary tree sequences transformation using left-arm and right-arm rotations, *Theoretical Computer Science*, 335 (2006) 303-314.
33. R. Wu, J. Chang and C. Chang, Ranking and unranking of non-regular trees with a prescribed branching sequence, *Mathematical and Computer Modeling*, 53 (2011) 1331-1335.
34. L. Xiang, K. Ushijima and C. Tang, On generating k-ary trees in computer representation, *Information Processing letters*, 77 (2001) 231-238.
35. S. Zaks, Lexicographic generation of ordered trees, *Theoretical Computer Science*, 10 (1980) 63-82.