# Generating rooted and free plane trees

Joe Sawada*

February 15, 2005

**Running Head: Generating rooted and free plane trees**

**Mail to:**
Joe Sawada (sawada@cis.uoguelph.ca)
Department of Computing and Information Science
University of Guelph
Guelph, Ont
CANADA N1G 2W1

*University of Guelph, CANADA. Research supported by NSERC. email: sawada@cis.uoguelph.ca

**Abstract**

This paper has two main results. First, we develop a simple algorithm to list all non-isomorphic rooted plane trees in lexicographic order using a level sequence representation. Then, by selecting a unique centroid to act as the root of a free plane tree, we apply the rooted plane tree algorithm to develop an algorithm to list all non-isomorphic free plane trees. The latter algorithm also uses a level sequence representation and lists all free plane trees with a unique centroid first followed by all free plane trees with two centroids. Both algorithms are proved to run in constant amortized time using straightforward bounding methods.

**Keywords**: rooted plane tree, free plane tree, planar tree, necklace, chord diagram, CAT algorithm, generate

# 1 Introduction

The development of algorithms to list all non-isomorphic occurrences of some combinatorial object is a fundamental pursuit within the realm of theoretical computer science. Such algorithms find application in many diverse areas including: hardware and software testing, combinatorial chemistry, and computational biology. In addition, such lists are often studied with the hope of gaining a more thorough understanding of a particular class of objects.

When developing such algorithms, the ultimate performance goal is for the amount of computation to be proportional to the number of objects generated. Such algorithms are said to be CAT for constant amortized time. When analyzing such algorithms, the correct measure of computation is the total amount of data structure change and not the time required to print the objects. This is because many applications only process the part of the object that has undergone some change.

Trees are among the most fundamental of combinatorial objects. A *rooted tree* is a tree with a distinguished root node. Since the subtrees of each node are unordered, equivalence classes can be obtained by re-ordering the subtrees of a node. The first CAT algorithm for generating all non-isomorphic rooted trees was developed by Beyer and Hedetniemi [2] using a level sequence representation.

Trees without a distinguished root node (or connected graphs without cycles) are called *free trees*. Due to the absence of a root node, the generation of non-isomorphic free trees is a more difficult problem. The free tree generation algorithms of Wright, Richmond, Odlyzko, and McKay [15], and Li and Ruskey [7] handle this problem by using a unique center of the free trees to act as the root. When we consider free plane trees, we will also pick a unique node to act as the root; however instead of using a center, we will use a centroid. The definitions of center and centroid [6] are given in Section 4.1.

When a rooted tree is embedded in a plane, a cyclic ordering is induced on the subtrees of the root. Such trees are called *rooted plane trees* and equivalence classes are obtained by rotating (rather than re-ordering) the subtrees of the root node. If there is no specified root, a tree embedded in a plane is called a *free plane tree* (also *planar trees* or *plane trees*). As an example, the 10 non-isomorphic rooted plane trees with 5 nodes are shown in Figure 1.

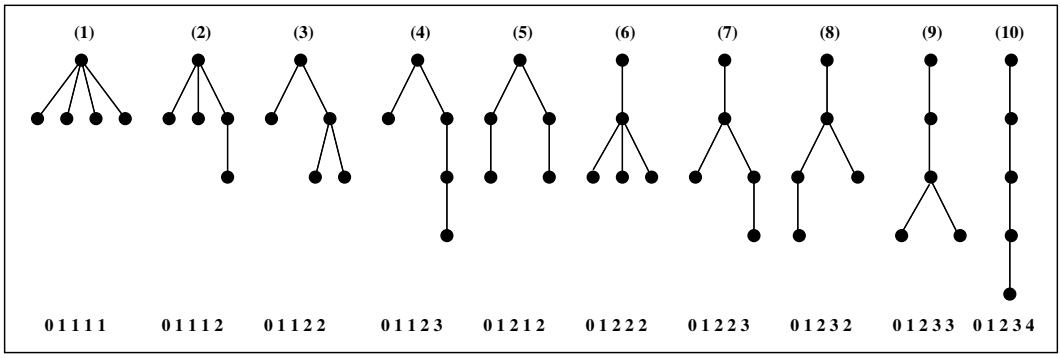


Figure 1: Non-isomorphic rooted plane trees with 5 nodes.

Notice that the trees (7) and (8) are equivalent rooted trees. Also, observe that the following sets of rooted plane trees $\{(1),(6)\}$, $\{(2),(3),(7),(8),(9)\}$, and $\{(4),(5),(10)\}$ correspond to the 3 equivalence classes of free plane trees.

One of the first papers discussing rooted and free plane trees was by Harary, Prins, and Tutte [5]. In that paper, generating functions were discovered to enumerate the number of non-isomorphic instances of these objects with $n$ nodes. Since then, the following closed formulae have been obtained by Walkup [14], where $r_n$ denotes the number of non-isomorphic rooted plane trees with $n$ nodes and $f_n$ denotes the number of non-isomorphic free plane trees with $n$ nodes. It is assumed that $n > 1$ and that $[\![P]\!]$ has value 1 if the proposition $P$ is true and 0 otherwise. The Euler totient function on a positive integer $n$, denoted $\phi(n)$, is the number of integers in the set $\{0, 1, \ldots, n-1\}$ that are relatively prime to $n$.

$$r_n = \frac{1}{2(n-1)} \sum_{d|(n-1)} \phi((n-1)/d) \binom{2d}{d},$$

$$f_n = r_n - \frac{1}{2n}\binom{2(n-1)}{n-1} + \frac{1}{n}\binom{n-2}{\frac{n}{2}-1}[\![n \text{ even}]\!].$$

Using these formulae we compute a table of the $r_n$ and $f_n$ as follows (the corresponding sequence numbers in Sloane's database [13] are A003239 and A002995 respectively):

| $n$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $r_n$ | 1 | 1 | 2 | 4 | 10 | 26 | 80 | 246 | 810 | 2704 | 9252 | 32066 |
| $f_n$ | 1 | 1 | 1 | 2 | 3 | 6 | 14 | 34 | 95 | 280 | 854 | 2694 |

A *chord diagram* with $n$ chords is a sequence of $2n$ points embedded on an oriented circle with the points joined pairwise by chords. A correspondence can be made between free plane trees with $n$ edges and chord diagrams where the $n$ chords are non-crossing. To observe the one-to-one correspondence, we place a node within each region of the chord diagram. There is an edge between two nodes if and only if the two regions share a chord. Clearly, by rotating the chord diagram, we obtain the same free plane tree. An algorithm for generating chord diagrams is given in [11], however, there is no efficient algorithm known for generating chord diagrams with non-crossing chords.

Some special cases of free plane trees have also been studied. For example, plane trees with bounded degree is studied in [8] and planted plane trees are studied in [4, 5]. Free plane trees also have an important application in the area of graph drawing [1].

In [9], Nakano develops an efficient algorithm to generate a type of rooted tree which he also calls a rooted plane tree. However, in his definition, a left to right order is placed on the children of each vertex, but circular rotation is *not* considered.

In this paper we consider two problems: the efficient generation of rooted plane trees and the efficient generation of free plane trees. As background, we discuss the level sequence representation for trees in Section 2. Also in that section, we present a generation algorithm for an object that is closely related to plane trees: necklaces. Then, in Section 3, we apply the necklace generation algorithm to generate rooted plane trees. This new algorithm can be implemented so the trees are output in lexicographic (or reverse lexicographic) order with respect to the level sequence representation. The rooted plane tree algorithm is then applied to generate all free plane trees in Section 4 by choosing a unique centroid to act as the root. For each algorithm, a straightforward analysis is given to prove that the algorithms run in constant amortized time.

# 2 Background

In this section we outline a common representation for trees and give a background on a related object called a necklace. We also present some algorithms that will be used in the development of our rooted and free plane tree algorithms discussed in Sections 3 and 4.

## 2.1 Representation

One of the most common ways to represent a rooted tree is by its *level sequence*. A level sequence is obtained by traversing the tree in preorder and recording the level (distance from the root) of each node as it is visited. For example, the trees in Figure 1 are shown with their corresponding level sequences in lexicographical order. The root is represented by a 0 and the children of the root are represented by a 1. Observe that an element in such a sequence cannot exceed the previous value by more than 1. This leads to the following result used by Scoins [12] to describe an algorithm for generating rooted trees.

THEOREM 1 *The sequence $a_0 a_1 \cdots a_{n-1}$ is a (pre-order) level sequence for some rooted tree $T$ if and only if $a_0 = 0$ and $1 \leq a_i \leq a_{i-1} + 1$ for all $1 \leq i < n$.*

This simple theorem can be used directly to generate all possible rooted trees with $n$ nodes using the level sequence representation. Such an algorithm is shown in Figure 2 where the initial call is RootedTree(1) and $a_0$ is initialized to 0. If we consider the computation tree of this recursive algorithm, then note that every internal node, except the root, has more than one child. Thus, the number of leaves (rooted trees with $n$ nodes) is greater than or equal to the number of internal nodes (rooted trees with less than $n$ nodes). Therefore, since each node is the result of a constant amount of work, the algorithm is CAT. As an example, the computation tree for $n = 5$ is shown in Figure 3.

```
procedure RootedTree ( t : integer )
local j : integer
    if t = n then Print()
    else
        for j ∈ {1, 2, ..., a_{t-1} + 1} do
            a_t := j
            RootedTree( t + 1 )
end
```

Figure 2: Algorithm for generating rooted trees with $n$ nodes.

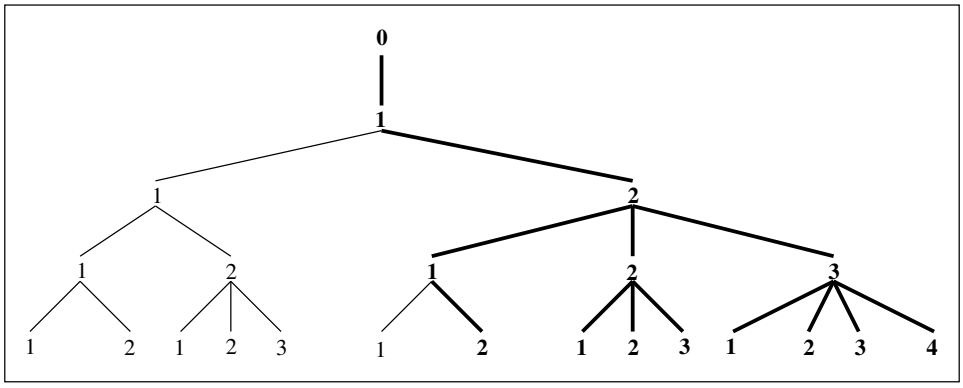


Figure 3: Computation tree of RootedTree($t$) for $n = 5$.

Now suppose that we are only interested in rooted trees with $n$ nodes that have a level sequence that is lexicographically greater than or equal to the level sequence of some rooted tree $T$. To generate such trees, we can simply trace the level sequence of $T$, branching to lexicographically larger trees where possible. If we allow a $O(n)$ initialization, then it is possible to generate such trees in constant amortized time using essentially the same argument as before. In Figure 3, the more thickly drawn edges (and nodes with bold labels) represent the computation tree for the generation of all rooted trees with 5 nodes whose level sequences are lexicographically greater than or equal to 01212.

Each of these algorithms will be applied in Section 4 to generate free plane trees.

## 2.2 Necklaces

A *necklace* is defined to be the lexicographically smallest element in an equivalence class of strings under rotation. Aperiodic necklaces are called *Lyndon words* and a prefix of a necklace is called a *prenecklace*. For example, the set of all length 4 binary necklaces is: $\{0000, \mathbf{0001}, \mathbf{0011}, 0101, \mathbf{0111}, 1111\}$. The strings in bold are the Lyndon words, and binary prenecklaces of length 4 are the necklaces combined with the strings $\{0010, 0110\}$

The following theorem [3] is the basis for a CAT algorithm for generating necklaces, Lyndon words, and prenecklaces. $\mathbf{P}_k(n)$ denotes the set of all $k$-ary prenecklaces with length $n$ and the function *lyn* on strings returns the length of the longest Lyndon prefix of the

```
procedure Necklace ( t, p : integer )
local j : integer
    if t = n then
        if p | n then Print()
    else
        for j ∈ {a_{t-p}, a_{t-p} + 1, ..., k - 1} do
            a_t := j
            if j = a_{t-p} then Necklace( t + 1, p )
            else Necklace( t + 1, t )
end
```

Figure 4: Algorithm for generating $k$-ary necklaces with length $n$ [3].

string:

$$lyn(a_1 a_2 \cdots a_n) = max\{1 \leq p \leq n \mid a_1 a_2 \cdots a_p \text{ is a Lyndon word}\}.$$

THEOREM 2 *Let $\alpha = a_1 \cdots a_{n-1}$ be a string in $\mathbf{P}_k(n-1)$ and let $p = lyn(\alpha)$. Then $\alpha b \in \mathbf{P}_k(n)$ if and only if $a_{n-p} \leq b \leq k-1$. Furthermore,*

$$lyn(\alpha b) \;=\; \begin{cases} p & \text{if } b = a_{n-p} \\ n & \text{if } a_{n-p} < b \leq k-1. \end{cases}$$

This theorem can immediately be applied to generate all prenecklaces. Necklaces can be generated by omitting prenecklaces where $n \bmod p \neq 0$, and Lyndon words can be generated by omitting prenecklaces where $n \neq p$. Pseudocode for an algorithm to generate $k$-ary necklaces with length $n$ is shown in Figure 4 where $a_0$ is initialized to 0, the function Print() outputs the current necklace $a_1 \cdots a_n$, and the initial call is Necklace(1,1). Maintaining the parameter $p$, which represents the length of the longest Lyndon prefix of the current string $a_1 a_2 \cdots a_{t-1}$, is the key to the algorithm. This algorithm will be modified in Section 3 to generate all rooted plane trees.

Another consequence of this Theorem 2 is that every prenecklace $\alpha$ that is not a necklace has the form $\alpha = (a_1 \cdots a_p)^j a_1 \cdots a_m$ where $p = lyn(\alpha)$, $j \geq 1$ and $0 < m < p$. Letting $n_i$ denote the number of occurrences of the symbol $i$ in $a_1 \cdots a_m$ we define $\gamma = 0^{n_0} 1^{n_1} 2^{n_2} \cdots$. We now define a function that will be useful later in the analysis of our algorithms for rooted and free plane trees. Let $f$ be a mapping from prenecklaces $a_1 a_2 \cdots a_n$ that are not necklaces such that $a_1 \neq a_n$ to all $k$-ary words:

$$f(\alpha) = \gamma (a_1 \cdots a_p)^j.$$

For example $f((1223123)^5 122312) = 112223(1223123)^5$. It follows from the proof of Lemma 4.1 in [10], that this mapping is one-to-one. In addition the length is preserved and $f(\alpha)$ is a Lyndon word (and hence a necklace).

# 3 Rooted plane trees

Recall that a *rooted plane tree* is a rooted tree embedded in the plane where equivalence can be obtained by rotating the subtrees of the root. In this section we consider two algorithms

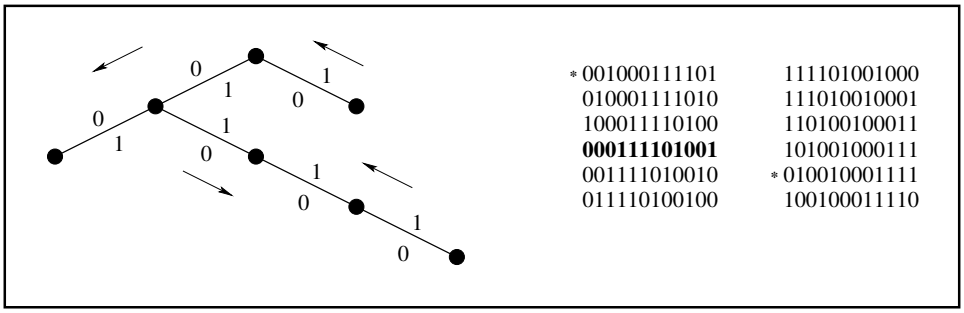


Figure 5: The corresponding equivalence class of strings for a rooted tree.

for generating all non-isomorphic rooted plane trees. The first method uses a bijection between rooted plane trees of size $n+1$ and binary necklaces with $n$ 0's and $n$ 1's. The latter object is an instance of a necklace with fixed density (the number of non-zero characters is fixed) and a CAT algorithm for generating such necklaces is given in [10]. The second algorithm uses a correspondence with another restricted class of necklaces and outputs the trees in a useful level sequence representation. Both algorithms run in constant amortized time.

## 3.1 A correspondence

A bijection [13] between rooted plane trees with $n+1$ nodes and binary necklaces with $n$ 0's and $n$ 1's can be obtained by traversing the outside of a tree (left to right), recording a 0 for each step away from the root, and recording a 1 for each step toward the root. For example, Figure 5 shows the equivalence class of strings obtained for the given rooted tree. Notice that the same set of strings is obtained if we rotate the subtrees of the root node. The necklace (the lexicographically smallest string) that corresponds to the tree in this example is the string in bold 000111101001. Using the fixed density necklace generation algorithm [10], we can generate the lexicographically smallest string of each equivalence class in constant amortized time. Each string generated by this algorithm, however, is not necessarily one obtained by starting from the root node. In this example, if we start from the root we obtain the string 001000111101. If we rotate the two subtrees of the root we obtain the string 010010001111. These strings are marked by a $*$ in Figure 5. Therefore, even though we have a CAT generation algorithm, the representation of the tree may not be very useful.

## 3.2 A fast and simple algorithm

The algorithm outlined in the previous subsection is a straightforward application of a previously known algorithm. However, the representation used for the rooted plane trees is not standard, particularly since no root node is established. Therefore, we develop another (simpler) algorithm where the trees are represented more naturally by their preorder level sequences. This new algorithm will also be applied in the next section when we focus on free plane trees.

```
procedure RootedPlane ( t, p : integer )
local j : integer
      if t = n then
            if p | (n − 1) then Print()
      else
            for j ∈ {a_{t−p}, ..., a_{t−1}, a_{t−1} + 1} do
                  a_t := j
                  if j = a_{t−p} then RootedPlane( t + 1, p )
                  else RootedPlane( t + 1, t )
end
```

Figure 6: Algorithm for generating rooted plane trees with $n$ nodes.

If we let a rooted plane tree be represented by its level sequence, then by ignoring the leading zero, which corresponds to the root, we are left with a sequence of the subtrees whose roots are represented by a 1. Given such a sequence, we obtain equivalencies by rotating a different 1 to the front of the string. For example, the rooted tree sequence 012312212 is equivalent to 012212123 and 012123122. Now if we choose the lexicographically smallest string to be the representative for each equivalence class of rooted trees (the *canonic rooted plane tree*), then we can generate all non-isomorphic rooted plane trees with $n$ nodes by generating all length $n-1$ necklaces $a_1 a_2 \cdots a_{n-1}$ over an alphabet of size $n$ with the added restrictions:

- $a_1 = 1$,
- $1 \leq a_i \leq a_{i-1} + 1$ for all $2 \leq i \leq n-1$, and
- 0 is appended to the front of the necklaces.

Such an algorithm can be obtained by applying these restrictions to the recursive $k$-ary necklace algorithm shown in Figure 4. In fact, the only modification required besides initialization is to modify the bound on the **for** loop from $k-1$ to $a_{t-1}+1$. The resulting algorithm is shown in Figure 6 where $a_0$ is initialized to 0 and $a_1$ is initialized to 1. The initial call is RootedPlane(2,1) and the function Print() prints the string $a_0 \cdots a_{n-1}$. Notice that the sequences can be listed in either an increasing or a decreasing order. Experimentally, this algorithm runs about 4 times faster than the fixed density necklace algorithm mentioned in Section 3.1.

## 3.3 Analysis

In this subsection we analyze the algorithm RootedPlane$(t, p)$ for generating all non-isomorphic rooted plane trees with $n$ nodes. Even though the original necklace algorithm in [3] is proved to run in constant amortized time, the result does not immediately apply to our modified algorithm.

By studying the algorithm, it is obvious that each recursive call is the result of a constant amount of work. Thus, the total amount of computation is proportional to the size of

the computation tree, where each node in the computation tree corresponds to a unique prenecklace with length less than $n$. Let $CompTree(n)$ denote the total number of such prenecklaces in the computation tree and let $\mathbf{P}(t)$ denote the set of all prenecklaces (sequences without the starting 0) with length $t$ in the computation tree. Then we have:

$$CompTree(n) = \sum_{t=1}^{n-1} |\mathbf{P}(t)|.$$

Our goal is to show that $CompTree(n)$ is bounded above by some constant times the total number $r_n$ of trees generated.

We say that a prenecklace $a_1a_2\cdots a_t$ is a *child* of $a_1a_2\cdots a_{t-1}$. Note that every prenecklace $\alpha = a_1a_2\cdots a_t$ in the computation tree has at least one child, namely $\alpha(a_t+1)$. Also, if $\alpha$ is a Lyndon word (i.e., $lyn(\alpha) = t$), then it will have $a_t + 1 \geq 2$ children. These observations are used in the proof of the following lemma.

LEMMA 1 *For* $1 \leq t < n-1$,

$$2|\mathbf{P}(t)| \leq |\mathbf{P}(t+1)|.$$

PROOF: Since each node $\alpha = a_1a_2\cdots a_t$ in $\mathbf{P}(t)$ has at least one child, we can prove the lemma by showing that the number of sequences in $\mathbf{P}(t)$ with only one child is less than the number of sequences with at least 3 children.

Suppose that $\alpha$ has only one child, where $lyn(\alpha) = p$. This implies that $\alpha$ is not a Lyndon word, and hence $p < t$. If we map $\alpha$ to $a_1a_2\cdots a_p234\cdots t-p+1$, then the resulting sequence is a Lyndon word in $\mathbf{P}(t)$ that has $t-p+2 \geq 3$ children. Because each $\alpha \in \mathbf{P}(t)$ has a unique longest Lyndon prefix (a consequence of Theorem 2), this mapping is one-to-one. Hence the number of sequences with only one child is less than or equal to the number of sequences with 3 or more children. Thus on average, the number of children of each sequence in $\mathbf{P}(t)$ is greater than or equal to 2. □

LEMMA 2 *For* $n \geq 1$,

$$|\mathbf{P}(n-1)| \leq 3r_n.$$

PROOF: Suppose that $\alpha = a_1\cdots a_{n-1}$ is in $\mathbf{P}(n-1)$ but is not a necklace (i.e., $0\alpha$ is not a canonic rooted plane tree). We consider two cases. If $a_{n-1} = 1$, notice that $a_1\cdots a_{n-2}2$ is a necklace (from Theorem 2) and thus $0a_1\cdots a_{n-2}2$ is a canonic rooted plane tree. Otherwise, if $a_{n-1} \neq 1$ then observe that since $a_1 = 1$, $f(\alpha)$ is a necklace and $0f(\alpha)$ is a canonic rooted plane tree, where $f$ is defined in Section 2.2. Since $f$ is a one-to-one mapping, the number of sequences $\alpha \in \mathbf{P}(n-1)$ where $0\alpha$ is not a canonic rooted plane tree is less than or equal to $2r_n$. Thus, $|\mathbf{P}(n-1)| \leq 3r_n$. □

Now applying these two lemmas we have:

$$\begin{aligned} CompTree(n) &= \sum_{i=1}^{n-1} |\mathbf{P}(t)| \\ &\leq 2|\mathbf{P}(n-1)| \\ &\leq 6r_n. \end{aligned}$$

This proves the following theorem.

THEOREM 3 *The algorithm* RootedPlane$(t, p)$ *for generating all non-isomorphic rooted plane trees with $n$ nodes runs in constant amortized time.*

# 4 Free plane trees

Recall that free plane trees are trees with a given planar embedding and no distinguished root. In this section we develop an algorithm for generating non-isomorphic free plane trees followed by an analysis that proves the algorithm runs in constant amortized time.

## 4.1 Algorithm

Due to the absence of a root, the problem of generating non-isomorphic free plane trees is more complicated than the rooted plane tree case. The approach we take is similar to the approaches of the free tree algorithms: we define a unique root for each free tree and then generate the resulting restricted classes of rooted plane trees.

Two natural candidates for determining a unique root are the *center*(s) and the *centroid*(s) [6]. A node $v$ is a *center* of a tree if the maximum distance to any other node in the tree is minimum. A tree with one center is called *unicentral*; a tree with two centers is called *bicentral*. If a tree is bicentral then the two centers must be adjacent.

The *size* of a tree (or subtree) is defined to be the number of nodes it contains. A node $v$ is a *centroid* of a tree if the size of the largest subtree that results when the node $v$ is removed is minimum. A tree may have either one or two centroids. If it has one centroid, we say that it is *unicentroidal*; if it has two centroids then we say that it is *bicentroidal*. A node $v$ is a unique centroid if and only if the size of its largest subtree is less than or equal to $\lfloor (n-1)/2 \rfloor$. If a tree is bicentroidal then the centroids are adjacent and the removal of the edge between them results in two subtrees of the same size. Clearly if $n$ is odd, then there are no bicentroidal trees.

If we pick a center to act as the unique root, then to use the rooted plane tree algorithm RootedPlane$(t, p)$, we need to maintain the depth of each subtree from the root. We define the *depth* of a subtree to be maximum distance from a node in the subtree to the root. If the trees are unicentral, then we must ensure that the two subtrees (from the root) with the largest depth in fact have the same depth. In the bicentral case, however, the problem is much harder because we must ensure that there exists a subtree with maximal depth $d$ and at least one subtree with depth $d-1$. The difficulty is that we may not know which subtree has maximal depth until the last few characters in the sequence are assigned. This makes the task of testing whether the root corresponds to the unique center a very difficult one to perform efficiently, no matter how the unique center is defined.

If we use a centroid to root the free trees, then the problem of generating non-isomorphic free plane trees becomes much easier. For unicentroidal trees we pick the unique centroid to be the root. For bicentroidal trees, when we remove the edge between the two centroids we obtain two subtrees with size $n/2$. The centroid we choose as the root is the one whose subtree's level sequence is the lexicographically smallest. If the sequences are the same then

```
procedure Unicentroid ( t, p, s : integer )
local j, max : integer
    if t = n then
        if p | (n − 1) then Print()
    else
        if s = ⌊(n − 1)/2⌋ then max := 1 else max := a_{t−1} + 1
        for j ∈ {a_{t−p}, ..., max − 1, max} do
            a_t := j
            if j = a_{t−p} = 1 then Unicentroid( t + 1, p, 1 )
            else if j = a_{t−p} then Unicentroid( t + 1, p, s + 1 )
            else Unicentroid( t + 1, t, s + 1 )
end
```

Figure 7: Algorithm for generating unicentroidal planar trees with $n$ nodes.

it does not matter which one we pick. We call the centroid chosen as the root the *canonical centroid*. The problem is now simplified to the generation of all rooted plane trees where the root corresponds to the canonical centroid. We divide the problem by considering the unicentroidal and bicentroidal trees separately.

In the unicentroidal case, we want to generate all rooted plane trees with the restriction that no subtree from the root can contain more than $\lfloor (n-1)/2 \rfloor$ nodes. This restriction is easily added to the algorithm RootedPlane$(t, p)$ by maintaining an additional parameter $s$ which indicates the size of the current subtree. Once a subtree reaches its maximal size of $\lfloor (n-1)/2 \rfloor$, then we must start a new subtree. This means that the next character in the level sequence must be 1. Another way to look at the problem is to ensure that the maximum distance between successive 1's in the level sequence representation is bounded by $\lfloor (n-1)/2 \rfloor$. This modified version of the rooted plane tree algorithm is shown in Figure 7. The initial call is Unicentroid$(2, 1, 1)$.

If $n$ is even, then we must also generate the bicentroidal rooted plane trees where the root is the canonical centroid. To generate such trees we do not have to use the rooted plane tree algorithm. We can simply generate two level sequences of size $n/2$ (corresponding to the two subtrees obtained when the edge is removed between the two centroids) such that the second level sequence is lexicographically greater than or equal to the first level sequence. Then by adding 1 to each value in the second level sequence and appending it to the first level sequence we obtain a rooted plane tree with $n$ nodes whose root is the canonical centroid. This final step effectively joins the two centroids back together by making the second centroid a child of the canonical centroid. This algorithm for generating bicentroidal free plane trees uses the two algorithms outlined in Section 2.1. First we are concerned with generating all rooted trees with $n/2$ nodes. Then for each rooted tree $\alpha = a_0 a_1 \cdots a_{n/2-1}$ generated, we want to generate all rooted trees with $n/2$ nodes that are lexicographically greater than or equal to $\alpha$. When the second sequence is generated, each value is incremented by 1.

Pseudocode for the algorithm just describe is shown in Figure 8 where the initial call is Bicentroid(1, FALSE) and the value $a_0$ is initialized to 0. The code fragment marked [C] in the algorithm is used to generate the first $n/2$ values in the level sequence. Notice the

```
procedure Bicentroid ( t: integer, samePrefix : boolean )
local j, min : integer
[A]  if t = n then Print()
[B]  else if t = n/2 then
         a_t := 1
         Bicentroid( t + 1, TRUE )
[C]  else if samePrefix = FALSE then
         if t > n/2 then min := 2 else min := 1
         for j ∈ {min, min + 1, ..., a_{t-1} + 1} do
             a_t := j
             Bicentroid( t + 1, FALSE )
[D]  else
         for j ∈ {a_{t-n/2} + 1, ..., a_{t-1}, a_{t-1} + 1} do
             a_t := j
             if j = a_{t-n/2} + 1 then Bicentroid( t + 1, TRUE)
             else Bicentroid( t + 1, FALSE )
end
```

Figure 8: Algorithm for generating bicentroidal planar trees with $n$ nodes.

similarity of this fragment to the algorithm RootedTree($t$). Fragment [B] is then used to start the second level sequence of $n/2$ that is not to be smaller than the first level sequence. Since we must increment each value in this second sequence, we assign the second centroid the value 1 making it a child of the canonical centroid. In the remainder of this discussion, however, when we refer to the second sequence, we will assume that its values have not been incremented even though the code actually does increment the values by 1. In [B] the boolean *samePrefix* is set to TRUE indicating that the current prefix of the second sequence is the same as the first sequence. When *samePrefix* is TRUE, we enter fragment [D]. In this fragment, the second sequence is generated so that its prefix is not smaller than the prefix of the first sequence. Once the prefix of the second sequence is greater than the prefix of the first sequence, the parameter *samePrefix* is set to FALSE and the remainder of the generation is done by [C]. As mentioned, since we are incrementing the values in the second sequence by 1, a minimum value of 2 must be maintained for the remainder of the sequence. When the combined level sequences have length $n$, the fragment [A] prints out the sequence.

## 4.2 Analysis

In this subsection we prove that our algorithm for generating non-isomorphic free plane trees runs in constant amortized time. The analysis considers the two sub-algorithms separately.

At first glance, it may appear that we can use the proof from the rooted plane trees case for the unicentroidal case. However, in this case we have the added restriction that each subtree can have at most $\lfloor (n-1)/2 \rfloor$ nodes and hence once a subtree is generated with this maximum number of nodes we must start a new subtree. This will yield a node in the computation tree with only one child where previously it may have had many. Instead we

come up with a new proof which uses the same idea of comparing the number of generated sequences with length $t$ and length $t+1$.

Let $CompTree'(n)$ denote the number of nodes (prenecklaces) in the computation tree for Unicentroid($t$, $p$, $s$) and let $\mathbf{P}'(t)$ denote the set of all prenecklaces with length $t$ (not including the leading 0) in the computation tree. Thus we have:

$$CompTree'(n) = \sum_{t=1}^{n-1} |\mathbf{P}'(t)|.$$

Our goal is to show that $CompTree'(n)$ is bounded by some constant times the total number of canonic unicentroidal free plane trees of size $n$.

LEMMA 3 *For* $1 \leq t < n-1$,

$$|\mathbf{P}'(t+1)| \geq \frac{5}{4}|\mathbf{P}'(t)|.$$

PROOF: First observe that there are no dead ends in the computation tree for Unicentroid($t$, $p$, $s$). In other words every prenecklace in $\mathbf{P}'(t)$ has at least one child for $1 \leq t < n-1$. This is because a given prenecklace $\alpha = a_1 a_2 \cdots a_t$ in $\mathbf{P}'(t)$ with $p = lyn(\alpha)$ and $s$ corresponding to the size of the final (rightmost) subtree can always be extended to a new prenecklace by appending the character $a_{t+1-p}$. Note in the case when the size of the rightmost subtree $s = \lfloor (n-1)/2 \rfloor$ that $\alpha$ must be a necklace. Thus, by the nature of the algorithm, it is always (and only) possible to append the character $a_1 = 1$ to obtain a new prenecklace. We will show that the average number of children for the prenecklaces in $\mathbf{P}'(t)$ is at least 5/4 by mapping sequences with exactly one child to those that have at least two children.

We start by considering all prenecklaces $\alpha = a_1 \cdots a_t$ that have only one child where $s'$ denotes the size of the rightmost subtree of $a_1 \cdots a_{t-1}$. We will look at two cases depending on the size of $s'$. If $s' = \lfloor (n-1)/2 \rfloor$, then as discussed $a_t$ must equal 1. We map all such prenecklaces to $\beta = b_1 \cdots b_t = 11a_1 \cdots a_{t-2}$. Observe that this will map at most $b_t + 1$ prenecklaces to an image $\beta$ because of the restriction that $a_{t-1} \leq a_{t-2} + 1$. In the second case we must have $s' < \lfloor (n-1)/2 \rfloor$. In this case, since $\alpha$ has only one child, there is only one possible value for $a_t$, namely $a_{t-p}$. If $a_t > a_{t-p}$ then $\alpha$ would be a Lyndon word and have at least two children. We map this prenecklace to the prenecklace $\beta = b_1 \cdots b_t = 1a_1 \cdots a_{t-1}$. Observe that this will map at most one prenecklace to an image $\beta$.

Combining these two cases, there are $m$ prenecklaces (each having exactly one child) mapped to each image $\beta$ where $m \leq (b_t + 1) + 1$. Furthermore, since $\alpha$ is a prenecklace, with $a_1 = 1$, it is obvious that $1\alpha$ will also be a prenecklace, and hence each $\beta$ will be a prenecklace in $\mathbf{P}'(t)$. Additionally, by the nature of the maps the rightmost subtree of any image $\beta$ must have size less than $\lfloor (n-1)/2 \rfloor$. Also, it is not hard to see that $\beta$ will be a Lyndon word if $b_t \neq 1$, and thus will have $b_t + 1$ children in the computation tree. If $b_t = 1$, then since we have inserted a 1 at the front of $\beta$, $\beta$ will also have $b_t + 1$ children since we can add either a 1 or a 2 to extend $\beta$ to a new prenecklace.

Finally, if we combine an image $\beta$ together with its $m$ pre-images (each with exactly one child), we have $m+1$ unique prenecklaces in $\mathbf{P}'(t)$ of length $t$ with a combined total of $m + b_t + 1$ children. Since all unaccounted for prenecklaces in $\mathbf{P}'(t)$ have at least 2 children, each prenecklace of length $t$ will have at least $(m + b_t + 1)/(m + 1)$ children on average.

Plugging in the upper bound for $m = b_t + 2$ and the lower bound of $b_t = 1$ minimizes this average at $5/4$. □

We can now apply this lemma and induction to obtain the following bound on the size of the computation tree:

$$CompTree'(n) \leq 5|\mathbf{P}'(n-1)|.$$

LEMMA 4 *For $n \geq 1$, $|\mathbf{P}'(n-1)|$ is less than 4 times the number of canonic unicentroidal free plane trees with $n$ nodes.*

PROOF: We partition the set of prenecklaces in $\mathbf{P}'(n-1)$ that are not necklaces into 3 sets: those that end $1^k$ where $k > 1$, those that end with $c1$ where $c \neq 1$ and those that do not end with 1. We will take each partition and show a one-to-one mapping from each prenecklace in the set to a necklace. Let $\alpha = a_1 \cdots a_{n-1}$. If $\alpha$ falls into the first set we map it to $a_1 \cdots a_{n-2}2$. If $\alpha$ falls in the second set, we map it to $1a_1 \cdots a_{n-2}$. Finally, if it falls into the third set, we map it to $f(\alpha)$ where $f$ is defined in Section 2.2. In all cases the mappings are one-to-one, where each image corresponds to a necklace in $\mathbf{P}'(n-1)$. Thus $|\mathbf{P}'(n-1)|$ is less than or equal to 4 times the number of canonical unicentroidal free plane trees with $n$ nodes. □

Now applying Lemma 4 directly to our last bound on the computation tree we obtain the following theorem.

THEOREM 4 *The algorithm Unicentroid(t,p,s) for generating all non-isomorphic unicentroidal free plane trees with $n$ nodes runs in constant amortized time.*

It is well known that the number of rooted trees with $n$ nodes, $T(n)$, can be counted by the Catalan numbers. For the bicentroidal algorithm Bicentroid($t$, *samePrefix*), all level sequences with length $n/2$ (rooted trees with $n/2$ nodes) are generated in constant amortized time (see Section 2.1). For each specific rooted tree with $n/2$ nodes, say $a_0a_1 \cdots a_{n/2-1}$, the algorithm generates all level sequences that are lexicographically greater than or equal to $a_0a_1 \cdots a_{n/2-1}$ in constant amortized time plus an initialization of $O(n)$ (again see Section 2.1). Since there are $T(n/2)$ rooted trees with $n/2$ nodes, it follows that there are $T(\frac{n}{2})(T(\frac{n}{2})+1)/2$ non-isomorphic bicentroidal free plane trees with $n$ nodes. These trees are generated in constant amortized time plus the $O(nT(n/2))$ time required to retrace the initial tree of size $n/2$. However, since $n < T(n/2)$ as $n$ gets large, this time for this retracing of the tree will also be $O(T(\frac{n}{2})^2)$. This proves the following theorem.

THEOREM 5 *The algorithm Bicentroid(t,* samePrefix*) for generating all non-isomorphic bicentroidal free plane trees with $n$ nodes runs in constant amortized time.*

It follows from Theorem 4 and Theorem 5 that we can generate all non-isomorphic free plane trees with $n$ nodes in constant amortized time using the algorithms Unicentroid($t$, $p$, $s$) and Bicentroid($t$, *samePrefix*).

# 5 Acknowledgement

Thanks to Frank Ruskey and Brendan McKay for suggesting the problem. Also many improvements to this paper have been made thanks to the thorough reporting of the anonymous referees.

# References


[1] G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis, *Graph Drawing*, Prentice Hall, 1998.

[2] T. Beyer and S.M. Hedetniemi, Constant time generation of rooted trees, SIAM Journal on Computing, 9 (1980) 706-712.

[3] K. Cattell, F. Ruskey, J. Sawada, M. Serra, and C.R. Miers, Fast algorithms to generate necklaces, unlabeled necklaces, and irreducible polynomials over GF(2), Journal of Algorithms, Vol. 37 No. 2 (2000) 267-282.

[4] N.G. deBruijn and B. Morselt, A note on plane trees, J. Combinatorial Theory, 2 (1967) 27-34.

[5] F. Harary, G. Prins, and T.W. Tutte, The number of plane trees, Indag. Math., 26 (1964) 319-329.

[6] C. Jordan, Sur les assemblages des lignes, J. Reine Angew. Math, 70 (1869) 185-190.

[7] G. Li and F. Ruskey, Advantages of forward thinking in generating rooted and free trees, $10th$ Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), (1999) S939-940 (short form abstract).

[8] C.L. Mallows and K.W. Wachter, Valency enumeration of rooted plane trees, J. Austral. Math. Soc, 13 (1972) 472-476.

[9] S. Nakano, Efficient generation of plane trees, Information Processing Letters, 84 (2002) 167-172.

[10] F. Ruskey and J. Sawada, An efficient algorithm for generating necklaces of fixed density, SIAM Journal on Computing, 29 (1999) 671-684.

[11] J. Sawada, A fast algorithm for generating non-isomorphic chord diagrams, to appear in SIAM Journal on Discrete Mathematics.

[12] H. Scoins, Placing trees in lexicographic order, Machine Intelligence, 3 (1967) 43-60.

[13] N. Sloane, The on-line encyclopedia of integer sequences: ID A003239, www.research.att.com/~njas/sequences/index.html.

[14] D.W. Walkup, The number of plane trees, Mathematika, 19 (1972) 200-204.

[15] R.A. Wright, B. Richmond, A. Odlyzko, and B.D McKay, Constant time generation of free trees, SIAM Journal on Computing, 15 (1986) 540-548.