ISSN 1346-5597

Efficient Generation of Rooted Trees

Shin-ichi Nakano
Takeaki Uno

NII-2003-005E
July 2003

Efficient Generation of Rooted Trees

Shin-ichi Nakano[*] Takeaki Uno[†]

July 2, 2003

Abstract

In this paper we give an algorithm to generate all rooted trees with at most n vertices. The algorithm generates each tree in constant time on average. Furthermore the algorithm is simple, and clarifies a simple relation among the trees, that is a family tree of trees, and outputs trees based on the relation.

1 Introduction

It is useful to have the complete list of graphs with a specified property. One can use such a list to search for a counter-example to some conjecture, or to experimentally measure an average performance of an algorithm over all possible input graphs.

Many algorithms to generate given class of graphs are already known [B80] [LN01, N02, M98, W86]. Many nice textbooks have been published on the subject [G93, KS98, W89].

An algorithm to generate all rooted trees with n vertices is known[B80]. The algorithm generates each tree without duplications in constant time on average. The main idea in [B80] is to define the successor tree for each tree so that by repeatedly finding the successor tree of a derived successor tree one can generate all trees. The computation to find the successor tree is not so difficult but not so easy.

In this paper we give an algorithm to generate all rooted trees with "at most" n vertices. Our algorithm also generates each tree without duplications in constant time on average. Furthermore our algorithm is very simple and clarifies a simple relation among the trees, that is a family tree of the trees (see Fig. 1), and outputs trees based on the relation.

The rest of the paper is organized as follows. Section 2 gives some definitions. Section 3 shows a tree structure among rooted trees. Section 4 presents our algorithm. Finally Section 5 is a conclusion.

2 Preliminaries

In this section we give some definitions.

Let G be a connected graph with n vertices. An edge connecting vertices x and y is denoted by (x, y). A *tree* is a connected graph without cycles. A *rooted* tree is a tree with one vertex r chosen as its *root* . For each vertex v in a rooted tree, let $UP(v)$ be the unique

[*]Gunma University, Kiryu-Shi 376-8515, Japan, `nakano@cs.gunma-u.ac.jp`
[†]National Institute of Informatics, Tokyo 101-8430, Japan, `uno@nii.jp`

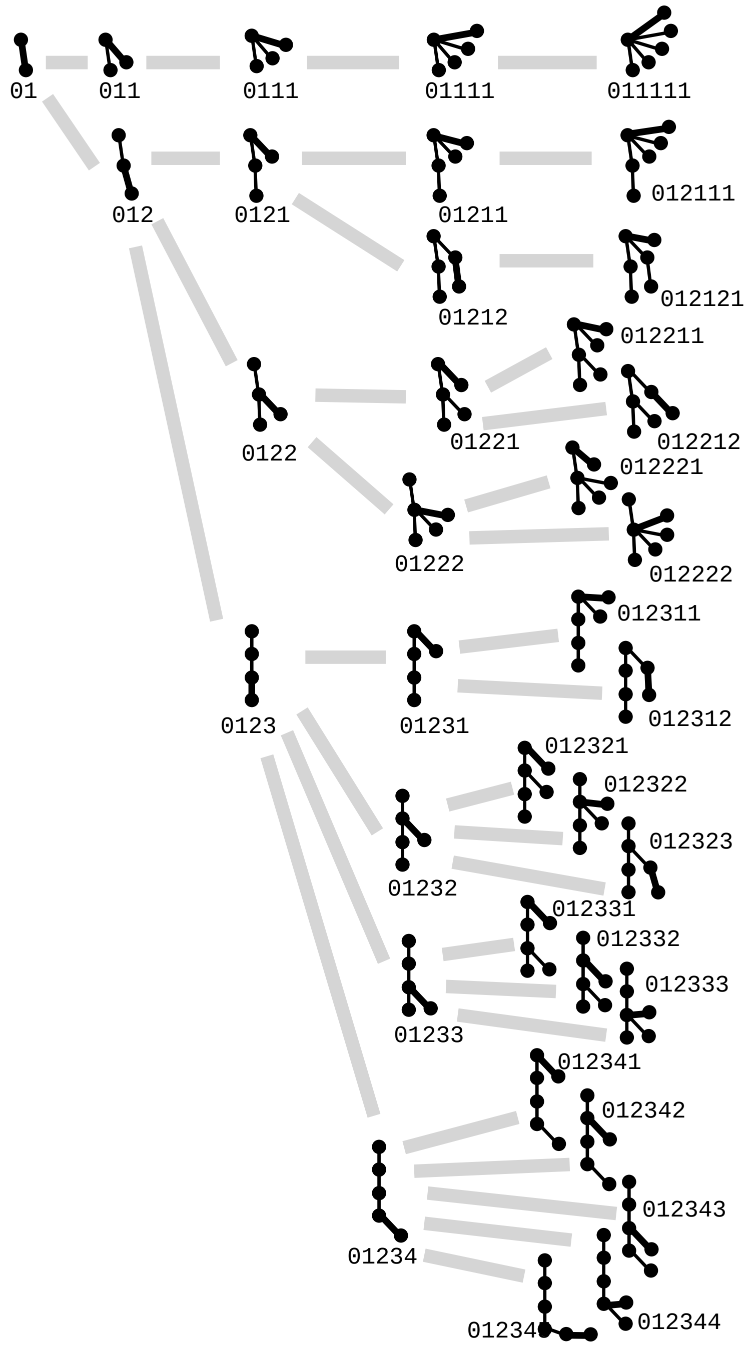

Figure 1: The family tree T_6 of rooted plane trees having at most six vertices.

path from v to the root r. If $UP(v)$ has exactly k edges then we say that the *depth* of v is k. The *parent* of $v \neq r$ is its neighbor on $UP(v)$, and the *ancestors* of $v \neq r$ are the vertices on $UP(v)$ except v. The parent of the root r and the ancestors of r are not defined. We say that if v is the parent of u then u is *a child* of v, and if v is an ancestor of u then u is *a descendant* of v. A *leaf* is a vertex having no child.

A *rooted plane tree* is a rooted tree with a left-to-right ordering specified for the children of each vertex. For a rooted plane tree T with root r_0, let $RP = (r_0, r_1, \cdots, r_k)$ be the path such that r_i is the rightmost child of r_{i-1} for each i, $1 \leq i \leq k$, and r_k is a leaf of T. We call RP *the rightmost path* of T, and r_k *the rightmost leaf* of T. We denote by $T(v)$ the rooted plane subtree consisting of vertex v and all descendants of v preserving the left-to-right ordering for the children of each vertex.

3 The depth sequence of a rooted plane tree

Let T be a rooted plane tree with n vertices, and $(v_1, v_2, \cdots, v_n)$ be the vertices of T in preorder[A95]. Let $dep(v_i)$ be the depth of v_i for $i = 1, 2, \cdots, n$. Then the sequence $L(T) = (dep(v_1), dep(v_2), \cdots, dep(v_n))$ is called the *depth sequence* of T. For example, see Fig. 2. Note that those trees in Fig. 2 are isomorphic as rooted trees, but non-isomorphic as rooted plane trees.

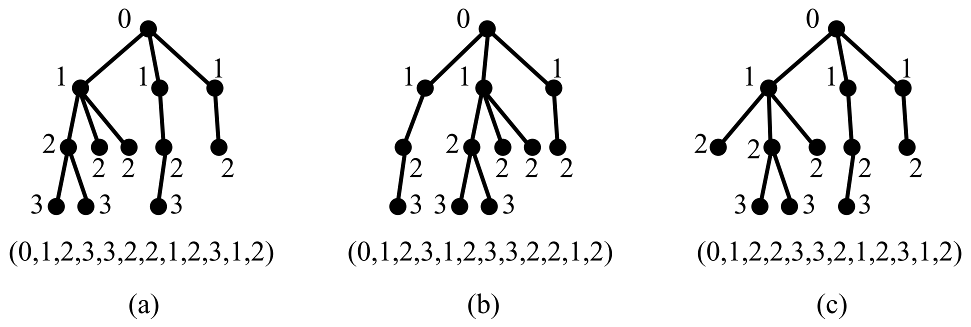

Figure 2: The depth sequences.

Let T_1 and T_2 be two rooted plane trees, and $L(T_1) = (a_1, a_2, \cdots, a_c)$ and $L(T_2) = (b_1, b_2, \cdots, b_d)$ be the depth sequences of them. If $a_i = b_i$ for each $i = 1, 2, \cdots, k-1$ (possibly $k = 1$) and either $a_k > b_k$ or $c > k - 1 = d$, then we say that $L(T_1)$ is *heavier* than $L(T_2)$.

Given a rooted tree T, we can observe that T corresponds to many non-isomorphic rooted plane trees, since we can choose many left-to-right orderings for the children of each vertex. Let T_h be the rooted plane tree corresponding to T having the heaviest depth sequence $L(T_h)$. Then we say T_h is the *left-heavy embedding* of T, and $L(T_h)$ is the *left-heavy depth sequence* of T. For example the rooted plane tree in Fig. 2(a) is the left-heavy embedding of a rooted tree, however Fig. 2(b) and (c) are not, since the one in Fig. 2(a) is heavier than them.

Thus we have assigned a unique distinct rooted plane tree, which is the left-heavy embedding, for each rooted tree. Let S_n be the set of all left-heavy embeddings having at most n and at least two vertices. If we generate all rooted plane trees in S_n, then it also means the generation of all rooted trees having at most n and at least two vertices. So we are going to generate all rooted plane trees in S_n.

We have the following two lemmas.

Lemma 1 *A rooted plane tree T is in S_n if and only if for every pair of consecutive child vertices v_1 and v_2, which appear in this order in the left-to-right ordering, $L(T(v_1)) \geq L(T(v_2))$ holds.*

Proof : By contradiction. ∎

Lemma 2 *Let T be a rooted plane tree in S_n having two or more vertices. Then the rooted plane tree derived from T by removing the rightmost leaf is also in S_n.*

Proof : Removing the rightmost leaf never changes the condition of $L(T(v_1)) \geq L(T(v_2))$ in Lemma 1. Thus each derived tree is also in S_n. ∎

Assume that T is a rooted plane tree in S_n having three or more vertices. We denote by $P(T)$ the rooted plane tree derived from T by removing the rightmost leaf. We say that $P(T)$ is *the parent tree* of T and T is *a child tree* of $P(T)$. By the lemma above $P(T)$ is also in S_n. Given a rooted plane tree T in S_n, by repeatedly removing the rightmost leaf, we can have the unique sequence $T, P(T), P(P(T)), \cdots$ of rooted plane trees in S_n, which eventually ends with K_2. By merging these sequences we can construct *the family tree* T_n of S_n such that the vertices of T_n correspond to the trees in S_n, and each edge corresponds to each relation between some T and $P(T)$. For instance T_6 is shown in Fig. 1.

4 Algorithm

In this section we give an algorithm to construct T_n.

If we can generate all child trees of a given tree in S_n, then in a recursive manner we can generate T_n, and which means we can generate all rooted trees having at most n vertices.

Let T be a rooted plane tree with the rightmost path $(r_0, r_1, \cdots, r_a)$. Let $T + i$ be a rooted plane tree derived from T by adding a new vertex v as the rightmost child of r_i. We can observe that each child tree of $T \in S_n$ is in $\{T + 0, T + 1, \cdots, T + a\}$, however not all trees in $\{T + 0, T + 1, \cdots, T + a\}$ are child trees of T, so we need to check whether each $T + i$ is a child tree of T or not.

We need some notation here. If a vertex r_{i-1} on the rightmost path has two or more child vertices, then we denote by s_i the child vertex of r_{i-1} preceding r_i. Thus s_i is the 2nd last child vertex of r_{i-1}.

We now have the following lemma.

Lemma 3 *Let T be a rooted plane tree in S_n with the rightmost path $(r_0, r_1, \cdots, r_a)$. Then $T + k$ is a child tree of T if and only if for each i, $i = 1, 2, \cdots, k$, either r_{i-1} has only one child vertex r_i in T, or $L(T(s_i)) \geq L(T(r_i))$ holds in $T + k$, where r_i is the rightmost child vertex of r_{i-1} and s_i is the child vertex of r_{i-1} preceding r_i.*

Proof : Since $T \in S_n$ the condition $L(T(v_1)) \geq L(T(v_2))$ in Lemma 1 is hold in T at every consecutive child vertices v_1 and v_2, and the condition remains as it was in $T + k$ except for $(v_1, v_2) = (s_1, r_1), (s_2, r_2), \cdots, (s_k, r_k)$. The claim checks all of these possible changes. ∎

If we generate each possible child tree $T + k$ of T and check whether it is actually a child tree or not based on the lemma above, then we need much running time. However we can save the running time as follows. We still need some definitions here.

Let T be a plane tree in S_n. We say that T is *active* at depth i if (i) the rightmost path contains a vertex r_i having depth i, (ii) r_i has two or more child vertices, (iii) $L(T(r_{i+1}))$

is a prefix of $L(T(s_{i+1}))$, where r_{i+1} is the rightmost child vertex of r_i and s_{i+1} is the child vertex of r_i preceding r_{i+1}. Intuitively, if T is active at depth i, then we are copying subtree $T(r_{i+1})$ from $T(s_{i+1})$.

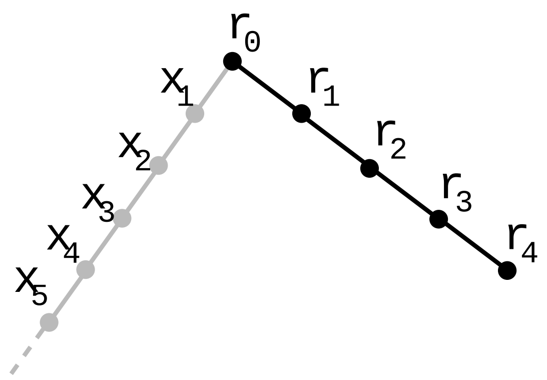

Figure 3: A path is active at depth 0.

If $T \in S_n$ is a path $(r_0, r_1, \cdots, r_a)$ then T is active at none of depth. For convenience, we imaginarily construct a rooted tree consisting of a path $(x_1, x_2, \cdots, x_{n-1})$ and consider x_1 as the first child vertex of the root r_0 of T, then we regard that T is active at depth 0. See Fig. 3. On the other hand, if $T \in S_n$ is not a path, let i be the depth of the last vertex in $L(T)$ having two or more child vertices, then T is active at depth i. Thus every $T \in S_n$ is active at some depth.

We say that the *copy-depth* of T is cd if T is active at depth cd but not active at any depth in $\{0, 1, \cdots, cd - 1\}$.

Now we are going to generate all child trees of a rooted plane tree T in S_n. We have the following three cases.

Case 1: T has n vertices.

Then T is a leaf in T_n, so T has no child tree.

Otherwise, we assume the copy-depth of T is cd, and the rightmost path of T is $(r_0, r_1, \cdots, r_a)$.

Case 2: If $L(T(s_{cd+1})) = L(T(r_{cd+1}))$ (Intuitively the copy is completed.)

The child trees of T are $T + 0, T + 1, \cdots, T + cd$.

Since T is left-heavy and the copy-depth of T is cd, for $i = 1, 2, \cdots, cd$, we have (if s_i exists) $L(T(s_i)) > L(T(r_i))$ and $L(T(r_i))$ is not a prefix of $L(T(s_i))$ in T. So even if we possibly append one depth to $L(T(r_i))$ we still have $L(T(s_i)) > L(T(r_i))$ for $i = 1, 2, \cdots, cd$. Thus by Lemma 3, $T + 0, T + 1, \cdots, T + cd$ are child trees of T. However, for each $T + i$, $i = cd + 1, cd + 2, \cdots, a$, we have $L(T(s_{cd+1})) < L(T(r_{cd+1}))$, so it is not left-heavy.

The copy-depth of $T + cd$ is cd, and the copy-depth of $T + i$ is i for $i = 0, 1, \cdots, cd - 1$.

Case 3: If $L(T(s_{cd+1})) \neq L(T(r_{cd+1}))$ (Intuitively the copy is not completed yet.)

Let $L(T(s_{cd+1})) = (dep(u_1), dep(u_2), \cdots, dep(u_b))$, $L(T(r_{cd+1})) = (dep(v_1), dep(v_2), \cdots, dep(v_c))$, and $dep(u_{c+1}) = d$. (Intuitively we are copying $T(r_{cd+1})$ from $T(s_{cd+1})$ and u_{c+1} is the next vertex to be copied.)

The child trees of T are $T + 0, T + 1, \cdots, T + (d - 1)$.

Note that for each of $T + d, T + (d + 1), \cdots, T + a$, we have $L(T(s_{cd+1})) < L(T(r_{cd+1}))$, so it is not left-heavy.

The copy-depth of $T + (d - 1)$ is cd. The copy-depth of $T + i$ is i for $i = 0, 1, \cdots, cd$.

We need some explanation for the copy-depth of $T+i$ for $i = cd+1, cd+2, \cdots, d-2$. We can observe that the copy-depth of $T+i$ is never less than cd, and $T+i$ is active at i. So the copy-depth of $T+i$ is somewhere between i and cd.

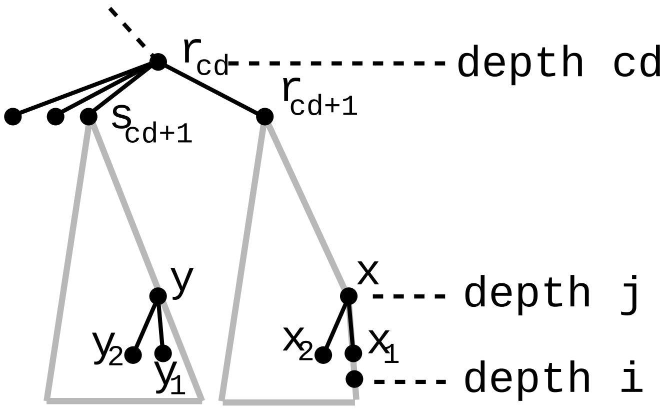

Figure 4: Illustration for Case 3.

Assume for the contradiction that the copy-depth of $T+i$ is $j < i$. Let $dep(x)$ be the last occurrence of depth j in $L(T+i)$. By the assumption above x has two or more child vertices. Let x_1 be the rightmost child vertex of x and x_2 be the child vertex of x preceding x_1. See Fig. 4. Let y be the vertex in $T(s_{cd+1})$ corresponding to x in $T(r_{cd+1})$, and y_1 and y_2 are vertices in $T(s_{cd+1})$ corresponding to x_1 and x_2 in $T(r_{cd+1})$. (Note that we are copying $T(r_{cd+1})$ from $T(s_{cd+1})$.) Now since $T \in S_n$, we have $L(T(y_2)) \geq L(T(y_1))$. By the choice of i, $L(T(y_1)) > L(T(x_1))$ holds, and $L(T(x_1))$ is not a prefix of $L(T(y_1))$. Since the copy-depth of T is cd, $L(T(y_2)) = L(T(x_2))$. Then $L(T(x_2)) = L(T(y_2)) \geq L(T(y_1)) > L(T(x_1))$ holds, and $L(T(x_1))$ is not a prefix of $L(T(y_1))$. Thus $L(T(x_1))$ is not a prefix of $L(T(x_2))$, and the copy-depth of $T+i$ is not j, a contradiction.

Thus the copy-depth of $T+i$ is i for $i = cd+1, cd+2, \cdots, d-2$.

Based on the case analysis above we have the following algorithm.

```
   Procedure find-all-children($T$, $cd$)
   { $T$ is the current tree, and $cd$ is the copy-depth of $T$.}
   begin
 1 Output $T$
 2 if $T$ has $n$ vertices                {Case 1}
 3 then return
 4 else if $L(T(s_{cd+1})) = L(T(r_{cd+1}))$
 5 then                                   {Case 2}
 6   begin
 7   for $i = 0$ to $cd$
 8     find-all-children($T+i$, $i$)
 9   end
10 else { $L(T(s_{cd+1})) > L(T(r_{cd+1}))$ } {Case 3}
11   begin
12   { Let $d$ be the depth of the next vertex to be copied.}
13   for $i = 0$ to $d-2$
14     find-all-children($T+i$, $i$)
15   find-all-children($T+(d-1)$, $cd$)
```

```
16   end
   end
   Algorithm find-all-trees($n$)
   begin
     Output $K_1$
     {$K_1$ is the only tree having exactly one vertices. }
     find-all-children($K_2$, 0)
   end
```

An execution of the algorithm is shown in Fig. 5.

Theorem 1 *The algorithm uses $O(n)$ space and runs in $O(f(n))$ time, where $f(n)$ is the number of nonisomorphic rooted trees having at most n vertices.*

Proof : Since we traverse the family tree T_n and output each tree in S_n at every vertex of T_n, we can generate all rooted trees having at most n vertex.

We maintain the last two occurrences of each depth value of the current depth sequence in two arrays of length n. We record the update of the two arrays in a stack, and restore the arrays if return occur. Thus we can find s_i and r_i in constant time for each i.

We also maintain the current copy-depth *cd* and the vertex next to be copied, so that with the help of the two arrays we can find r_{cd+1} and s_{cd+1} in constant time and we can check the condition in Line 4 in constant time. Also with the help of the two arrays we can compute the value d in Case 3 in constant time.

Other parts of the algorithm need only constant time for each edge of T_n. Thus the algorithm runs in $O(f(n))$ time. Note that the algorithm does not output entire trees but the difference from the previous tree.

For each recursive call we need a constant amount of space, and the depth of recursive call is bounded by n. Thus the algorithm uses $O(n)$ space. ∎

5 Conclusion

In this paper we have given an algorithm to generate all rooted trees having at most n vertices. The algorithm is simple, generates each tree in constant time on average, and clarifies a simple relation among the trees, that is a family tree of the trees.

Can we generate efficiently all (rooted) trees with n vertices and with diameter d?

References

[A95] A. V. Aho and J. D. Ullman, *Foundations of Computer Science*, Computer Science Press, New York, (1995). 1

[B80] T. Beyer and S. M. Hedetniemi, *Constant time generation of rooted trees*, SIAM J. Comput., 9, (1980), pp.706-712.

[G93] L. A. Goldberg, *Efficient algorithms for listing combinatorial structures*, Cambridge University Press, New York, (1993).

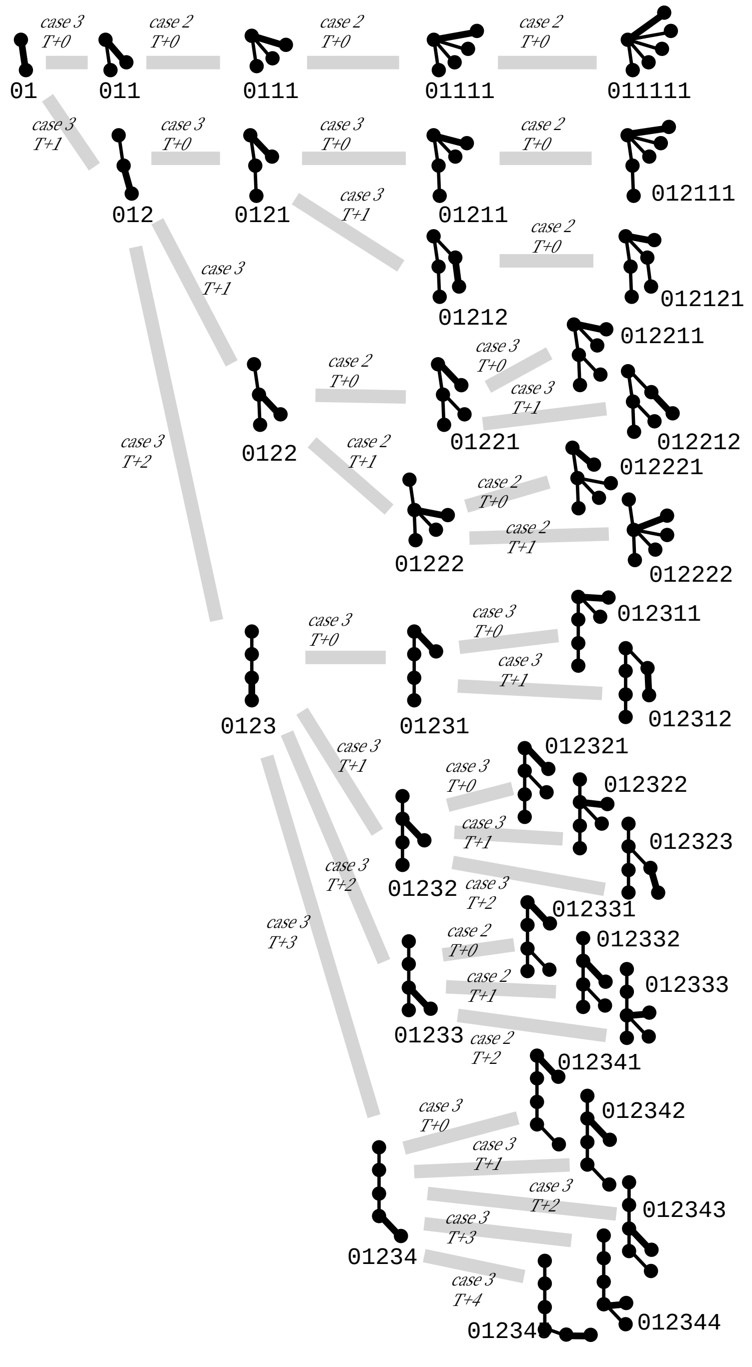

Figure 5: An execution of the algorithm.

[KS98] D. L. Kreher and D. R. Stinson, *Combinatorial algorithms*, CRC Press, Boca Raton, (1998).

[LN01] Z. Li and S. Nakano, *Efficient generation of plane triangulations without repetitions*, Proc. ICALP2001, LNCS 2076, (2001), pp.433–443.

[M98] B. D. McKay, *Isomorph-free exhaustive generation*, J. of Algorithms, 26, (1998), pp.306-324.

[N02] S. Nakano, *Efficient Generation of Plane Trees*, Information Processing Letters, 84, (2002), pp.167–172.

[R78] R. C. Read, *How to Avoid Isomorphism Search When Cataloguing Combinatorial Configurations*, Annals of Discrete Mathematics, 2, (1978), pp.107–120.

[W89] H. S. Wilf, *Combinatorial Algorithms : An Update*, SIAM, (1989).

[W86] R. A. Wright, B. Richmond, A. Odlyzko and B. D. McKay, *Constant time generation of free trees*, SIAM J. Comput., 15, (1986), pp.540-548.