A Gray Code of Ordered Trees

Shin-ichi Nakano
Gunma University
August 1, 2022

Abstract A combinatorial Gray code for a set of combinatorial objects is a sequence of all combinatorial objects in the set so that each object is derived from the preceding object by changing a small part.

In this paper we design a Gray code for ordered trees with \( n \) vertices such that each ordered tree is derived from the preceding ordered tree by removing a leaf then appending a leaf elsewhere. Thus the change is just remove-and-append a leaf, which is the minimum.

1 Introduction

A classical Gray code for \( n \)-bit binary numbers is a sequence of all \( n \)-bit binary numbers so that each number is derived from the preceding number by changing exactly one bit. A combinatorial Gray code for a set of combinatorial objects is a sequence of all combinatorial objects in the set so that each object is derived from the preceding object by changing a small (constant) part.

When we generate all combinatorial objects and the number of such objects is huge if we can compute them as a combinatorial Gray code then we can output (or store) each object as a small size of the difference from the preceding object and we may compute each object in a constant time. Also, when we repeatedly solve some problem for a class of objects, a solution for an object may help to compute a solution for a similar successive object. See surveys for combinatorial Gray codes [6, 4].

For binary trees with \( n \) vertices one can generate all binary trees so that each binary tree is derived from the preceding binary tree by a rotation operation at a vertex [2, 3]. The number of change of edges in a rotation operation is three [1, p9]. Also one can generate all binary trees with \( n \) vertices so that each tree is derived from the preceding tree by removing a subtree and place it elsewhere [1, Exercise 25]. However the levels of many vertices may be changed, where the level of a vertex is the number of vertices on the path from the vertex to the root.
In this paper we design a Gray code for ordered trees with \(n\) vertices such that each ordered tree is derived from the preceding ordered tree by removing a leaf then appending a leaf elsewhere. Thus the change is just remove-and-append a leaf, which is the minimum, and other vertices remain as they were including their levels. Our Gray code is based on a tree structure among the ordered trees.

The remainder of this paper is organized as follows. Section 2 gives some definitions and basic lemmas. In Section 3 we design our algorithm to construct a Gray code for the ordered trees with \(n\) vertices. Finally Section 4 is a conclusion.

2 Preliminaries

A tree is a connected graph with no cycle. A rooted tree is a tree with a designated vertex as the root. The level of a vertex \(v\) in a rooted tree is the number of vertices on the path from \(v\) to the root. The level of the root is 1. For each vertex \(v\) except the root if the neighbor vertex of \(v\) on the path from \(v\) to the root is \(p\) then \(p\) is the parent of \(v\) and \(v\) is a child of \(p\). The root has no parent.

In this paper we always draw each child vertex below its parent. A vertex with no child is called a leaf. An ordered tree is a rooted tree in which the left-to-right order of child vertices of each vertex is defined. The number of ordered trees with exactly \(n + 1\) vertices is known as the \(n\)-th Catalan number \(2^nC_n/(n + 1)\) [1, p12].

Given an ordered tree \(T\), let \(P_r(T) = (v_0, v_1, \ldots, v_k)\) be the path from the root \(v_0\) to a leaf \(v_k\) such that, for each \(i = 1, 2, \ldots, k\), \(v_i\) is the rightmost child of \(v_{i-1}\). \(P_r(T)\) is called the rightmost path of \(T\) and \(v_k\) is called the rightmost leaf of \(T\). The number of edges in \(P_r(T)\) is denoted by \(rpl(T)\).

For an ordered tree \(T\) if the rightmost child of the root has exactly one child as a leaf then we say \(T\) has the pony-tail.

For two distinct ordered trees \(T\) and \(T'\), if \(T'\) is derived from \(T\) by appending a new leaf as the rightmost leaf then removing other leaf, then we say \(T\) is copying \(T'\) (at level \(rpl(T')\)). When \(T\) is copying \(T'\) if the parent of the rightmost leaf of \(T'\) has two or more child vertices then \(rpl(T) \geq rpl(T')\) holds, otherwise, the parent of the rightmost leaf of \(T'\) has exactly one child vertex, which is the rightmost leaf, and \(rpl(T) = rpl(T') - 1\) holds. So if \(T\) is copying \(T'\), \(rpl(T) = 1\) and \(rpl(T') > 1\) then \(T'\) has the pony-tail.

Let \(S_k\) be the set of the ordered trees with exactly \(k\) vertices. In this paper we design, for each \(k = 1, 2, \ldots, n\), a combinatorial Gray code for \(S_k\), that is a sequence of all ordered trees in \(S_k\) such that each ordered tree is derived from the preceding ordered tree by removing a leaf then appending a leaf elsewhere. We call the change delete-and-append a leaf.

For an ordered tree \(T\) with \(n \geq 2\) vertices let \(p(T)\) be the ordered tree derived from \(T\) by removing the rightmost leaf. We say \(p(T)\) is the parent of \(T\), and \(T\) is a child of \(p(T)\). For any ordered tree \(T\) in \(S_n\) if we repeatedly compute the parent of the derived ordered tree we obtain the sequence \(T, p(T), p(p(T)), \ldots\) of ordered trees, which ends with the trivial ordered tree consisting of exactly
one vertex. We call the sequence the removing sequence of $T$ [5].

![Diagram of the family tree $F_n$ of $S_n$.]

Figure 1: The family tree $F_n$ of $S_n$.

By merging the removing sequences of the ordered trees in $S_n$ one can obtain an (unordered) tree $F_n$ of ordered trees [5] (See an example for $n = 5$ in Fig. 1) in which the root corresponds to the trivial ordered tree with exactly one vertex, each vertex at level $k$ corresponds to some ordered tree in $S_k$, and each edge corresponds to some ordered tree and its parent. We call the tree the family tree. Note that we have not decide yet the left-to-right order of the child ordered trees of each order tree in $F_n$. We have the following three lemmas.

**Lemma 1.** There is a bijection between the ordered trees in $S_k$ and the vertices at level $k$ in $F_n$.

*Proof.* Given an ordered tree $T$ with exactly $k$ vertices, by repeatedly appending a new leaf as the rightmost child of the root, one can obtain a descendant tree $T' \in S_n$ in $F_n$. Thus every order tree in $S_k$ appears in the removing sequence of some tree in $S_n$ and so corresponds to a vertex at level $k$ in $F_n$.

Clearly every vertex at level $k$ in $F_n$ corresponds to an ordered tree with exactly $k$ vertices. $\square$

**Lemma 2.** Let $T$ be an ordered tree in $S_k$ with $k < n$. $T$ has $rpl(T) + 1$ child ordered trees in $F_n$.

*Proof.* For each $i = 1, 2, \cdots, rpl(T) + 1$, by appending a new leaf as the rightmost child leaf of the vertex on $P_r(T)$ at level $i$, one can obtain a distinct child ordered tree. See Fig. 2 $\square$

We denote by $C(T, i)$ the child ordered tree of $T$ derived from $T$ by appending a new leaf as the rightmost child leaf of the vertex on $P_r(T)$ at level $i$. Thus $rpl(C(T, i)) = i$.  

3
Thus, by Lemma 2 every ordered tree $T$ in $S_k$ with $k < n$ except the ordered tree with exactly one vertex has two or more child ordered trees in $F_n$ since $rpl(T) \geq 1$. Clearly the ordered tree with exactly one vertex has exactly one child ordered tree in $F_n$.

**Lemma 3.** Any ordered tree is derived from its sibling ordered tree by delete-and-append a leaf.

**Proof.** Any ordered tree is derived from its sibling ordered tree by deleting the rightmost leaf then appending a leaf as the rightmost leaf at the suitable level.

In this paper we show that by suitably defining the left-to-right order of child ordered trees of each ordered tree in $F_n$, we can define an ordered tree $F_n^O$ such that, for each $k$, a Gray code for $S_k$ is appeared as the left-to-right sequence of the ordered trees corresponding to the vertices at level $k$ of $F_n^O$. Thus a Gray code for $S_n$ is appeared as the left-to-right sequence of the ordered trees corresponding to the leaves of $F_n^O$. See an example for $n = 5$ in Fig. 1.

### 3 Algorithm

In this section we design a Gray code for $S_k$ for each $k = 1, 2, \ldots, n$, where $S_k$ is the set of the ordered trees with exactly $k$ vertices.

**Induction on levels** We proceed by induction on levels. Let $F_k$ be the subtree of $F_n$ induced by $S_1 \cup S_2 \cup \cdots \cup S_k$. The Gray code for $S_1$ is trivial and unique since $|S_1| = 1$. Simillar for $S_2$ since $|S_2| = 1$. Assume that, for an integer $k < n$, we have defined a left-to-right order of child ordered trees of each ordered tree in $S_1 \cup S_2 \cup \cdots \cup S_{k-1}$, we have obtained an ordered tree $F_k^O$ corresponding to $F_k$, and we have constructed a Gray code for $S_k$ as the left-to-right sequence of the ordered trees corresponding to the leaves of $F_k^O$. Then we are going to define a left-to-right order of child ordered trees of each ordered tree in $S_k$ so that it extends $F_k^O$ to an ordered tree $F_{k+1}^O$ and a Gray code for $S_{k+1}$ is appeared as the left-to-right sequence of the ordered trees at the leaves of $F_{k+1}^O$. 

![Figure 2: An illustration for Lemma 2](image_url)
**Basic strategy of algorithm** Let \((T_1, T_2, \ldots)\) be our Gray code for \(S_k\). We are going to define a left-to-right order of child ordered trees of each \(T_i\) in \(S_k\), then we obtain a sequence of ordered trees, which is a Gray code for \(S_{k+1}\), say \((T'_1, T'_2, \ldots)\).

If two consecutive ordered trees \(T'_j\) and \(T'_{j+1}\) in the sequence are siblings in \(F_{k+1}^O\), then one can be derived from the other by delete-and-append a leaf by Lemma \[3\]. However if two consecutive ordered trees \(T'_j\) and \(T'_{j+1}\) are not siblings in \(F_{k+1}^O\), that is, \(T'_j\) is the rightmost child ordered tree of \(T_i\) and \(T'_{j+1}\) is the leftmost child ordered tree of \(T_{i+1}\) for some \(i\), then we have several cases to consider. We have the following lemma for \(T_i\) and \(T_{i+1}\).

**Lemma 4.** Assume that \(T_i\) can be derived from \(T_{i+1}\) by delete-and-append a leaf. Then the followings are hold.
(a) $C(T_i, 1)$ can be derived from $C(T_{i+1}, 1)$ by delete-and-append a leaf.

(b) If $\text{rpl}(T_i) = \text{rpl}(T_{i+1}) = 1$, then $C(T_i, 2)$ can be derived from $C(T_{i+1}, 2)$ by delete-and-append a leaf.

(c) If $\text{rpl}(T_i) = 1$, then $T_i$ is copying $T_{i+1}$ at level 1 and $T_{i+1}$ is copying $T_i$ at level 2, then $C(T_i, 2)$ can be derived from $C(T_{i+1}, 2)$ by delete-and-append a leaf.

(d) If $\text{rpl}(T_i) = 1$, and $T_{i+1}$ has no pony-tail (so $T_{i+1}$ is copying $T_i$ at level 1), then $C(T_i, 2)$ can not be derived from $C(T_{i+1}, 2)$ by delete-and-append a leaf (See Fig. 3 (d1) and (d2)), however $C(T_i, 2)$ can be derived from $C(T_{i+1}, 1)$ by delete-and-append a leaf. (See Fig. 3 (d3), (d4)).

(e) If $\text{rpl}(T_i) = 1$, $\text{rpl}(T_{i+1}) > 1$, $T_i$ has the pony-tail, and $T_{i+1}$ is copying $T_i$ at level 2, then $C(T_i, 2)$ can be derived from $C(T_{i+1}, 2)$ by delete-and-append a leaf. (See Fig. 3 (e)).

(f) If $\text{rpl}(T_i) = 1$, and $T_{i+1}$ is copying $T_i$ at level 1, then $C(T_i, 2)$ can not be derived from $C(T_{i+1}, 2)$ by delete-and-append a leaf (See Fig. 3 (f1)), however $C(T_i, 2)$ can be derived from $C(T_{i+1}, 1)$ by delete-and-append a leaf. (See Fig. 3 (f2)).

(g) If $\text{rpl}(T_i) = 1$, and $\text{rpl}(T_{i+1}) > 1$, then $C(T_i, 2)$ can be derived from $C(T_{i+1}, \text{rpl}(T_{i+1}))$ by delete-and-append a leaf. (See Fig. 3 (g)).

Proof. (a) (b) We have the following two cases. Case 1: $T_i$ is derived from $T_{i+1}$ by removing the rightmost leaf then appending a new leaf elsewhere. Case 2: $T_i$ is derived from $T_{i+1}$ by removing a leaf which is not the rightmost leaf then appending a new leaf elsewhere. For both cases the claim holds.

(c) Assume that $T_{i+1}$ is derived from $T_i$ by appending the rightmost leaf at level 1 then deleting a leaf $v$ (since $T_i$ is copying $T_{i+1}$), and $T_i$ is derived from $T_{i+1}$ by appending the rightmost leaf at level 2 then deleting a leaf $v'$ (since $T_{i+1}$ is copying $T_i$).

We can show that exactly one of $v$ or $v'$ is a child of the root, as follows. If $v$ is a child of the root of $T_i$ and $v'$ is a child of the root of $T_{i+1}$ then, since $T_i$
is copying $T_{i+1}$, the degree of the root of $T_i$ is equal to the degree of the root of $T_{i+1}$, and, since $T_{i+1}$ is copying $T_i$, the degree of the root of $T_{i+1}$ minus 1 is equal to the degree of the root of $T_i$, a contradiction. Also if $v$ is not a child of the root of $T_i$ and $v'$ is not a child of the root of $T_{i+1}$ then, since $T_i$ is copying $T_{i+1}$, the degree of the root of $T_i$ plus 1 is equal to the degree of the root of $T_{i+1}$, and, since $T_{i+1}$ is copying $T_i$, the degree of the root of $T_{i+1}$ is the degree of the root of $T_i$, a contradiction. Thus exactly one of $v$ or $v'$ is a child of the root.

Assume first that $v$ is a child of the root of $T_i$. Let $x_1, x_2, \ldots, x_d$ be the child vertices of the root in $T_i$ except $v$ in right-to-left order, and $y_1, y_2, \ldots, y_{d+1}$ the child vertices of the root in $T_{i+1}$ in right-to-left order. Since $T_i$ is copying $T_{i+1}$, after removing $v$ from $T_i$, the subtrees rooted at $x_1, x_2, \ldots, x_d$ are identical to the subtrees rooted at $y_1, y_2, \ldots, y_{d+1}$, respectively. Also since $T_{i+1}$ is copying $T_i$, after removing $v'$ from $T_{i+1}$, the subtrees rooted at $y_2, y_3, \ldots, y_{d+1}$ except one (corresponding to the trivial subtree rooted at $v$) are identical to the subtrees rooted at $x_2, x_3, \ldots, x_d$, respectively. If $v'$ belong to a subtree rooted at, say $y_j$, then, since $T_i$ is copying $T_{i+1}$, the subtree rooted at $x_{j-1}$ is identical to the subtree rooted at $y_j$, and also, since $T_{i+1}$ is copying $T_i$, after removing $v'$ from the subtree rooted at $y_j$, if it is identical to the subtree rooted at $x_{j-1}$, then, a contradiction. Thus $v'$ belong to the subtree corresponding to the subtree rooted at $v$, that is $v'$ is the only child of a child (corresponding to $v$) of the root. See Fig.3 (c). Now $C(T_i, 2)$ is derived from $C(T_{i+1}, 2)$ by delete-and-append a leaf.

Similar for the case where $v'$ is a child of the root of $T_{i+1}$.

(d) Since $T_{i+1}$ has no pony-tail, either (Case 1) the rightmost child vertex of the root of $T_{i+1}$ has two or more child vertices (See Fig.3 (d1)), or (Case 2) the rightmost child vertex of the rightmost child vertex of the root of $T_{i+1}$ has one or more child vertices (See Fig.3 (d3)). Since $rpl(T_i) = 1$ the rightmost child vertex of the root of $T_i$ has no child vertex. For Case 1, the rightmost child vertex of the root of $C(T_{i+1}, 2)$ has three or more child vertices, while the rightmost child vertex of the root of $C(T_i, 2)$ has exactly one child vertex. Thus $C(T_i, 2)$ can not be derived from $C(T_{i+1}, 2)$ by delete-and-append a leaf. See Fig.3 (d1). For Case 2 we need to remove at least two vertices and append at least two vertices to obtain $C(T_i, 2)$ from $C(T_{i+1}, 2)$. Thus $C(T_i, 2)$ can not be derived from $C(T_{i+1}, 2)$ by delete-and-append a leaf. See Fig.3 (d3). However $C(T_i, 2)$ can be derived from $C(T_{i+1}, 1)$ by delete-and-append a leaf. See Fig.3 (d2) and (d4).

(e) See Fig.3 (e).

(e') Similar to (e).

(f) See Fig.3 (f1) and (f2).

(g) See Fig.3 (g).

(g') Similar to (g).

\[ \square \]

**Step of algorithm** Let $(T_1, T_2, \ldots)$ be a Gray code for $S_k$ corresponding to the leaves of $F_k^O$ and we are going to define a left-to-right order of child ordered trees of each ordered tree in $S_k$ and construct a Gray code $(T'_1, T'_2, \ldots)$ for $S_{k+1}$ corresponding to the leaves of $F_{k+1}^O$. When we start step $i$ assume that
we have already defined the left-to-right order of the child ordered trees of
$T_1, T_2, \ldots, T_{i-1}$ and the leftmost child ordered tree of $T_i$, and in step $i$ we are
going to define the left-to-right order of the child ordered trees of $T_i$ except the
leftmost one, and the leftmost child ordered trees of $T_{i+1}$. See Fig.4. The part
we are going to define in the current step $i$ is depicted as a grey rectangle. We
proceed with several cases based on $rpl(T_i), rpl(T_{i+1})$ and the leftmost child of
$T_i$, as explained later.

Figure 4: An illustration for step $i$ of the algorithm.

Loop invariants

Our algorithm satisfies the following two conditions at each step $i$. (Note
that (co1) is independent of $i$.)

(co1) For consecutive three ordered trees $T_{u-1}, T_u, T_{u+1}$ at level $k$, if $rpl(T_{u-1}) = rpl(T_{u+1}) = 1$ and $rpl(T_u) > 1$ then $T_u$ has the pony-tail and $T_{u+1}$ is
copying $T_u$ at level 2. Also if $rpl(T_{u-1}) = rpl(T_{u+1}) \geq 2$ then $rpl(T_u) > rpl(T_u)$.

(co2) For consecutive three ordered trees $T'_{u'-1}, T'_{u'}, T'_{u'+1}$ at level $k+1$ with
$u'+1 \leq i'$, where $T'_{i'}$ is the leftmost child ordered tree of $T_i$, if $rpl(T'_{u'-1}) = rpl(T'_{u'+1}) = 1$ and $rpl(T'_{u'}) > 1$ then $T'_{u'}$ has the pony-tail and $T'_{u'+1}$ is
copying $T'_{u}$ at level 2. Also if $rpl(T'_{u'-1}) = rpl(T'_{u'+1}) \geq 2$ then
$rpl(T'_{u'-1}) > rpl(T'_{u})$.

The intuitive reason why we need those condition is as follows.
Assume that there are $T_{u-1}, T_u, T_{u+1}$ with $rpl(T_{u-1}) = rpl(T_{u+1}) = 1, rpl(T_u) > 1, T_u$ has no pony-tail, and $C(T_u, 1)$ is the leftmost child of $T_u$ (see Fig.5(a)), and
if we try to set $C(T_u, 1)$ at the rightmost child of $T_u$, then we fail to construct
a Gray code for $S_{k+1}$ since the same tree appear twice. (See Fig.5(b).) So our
algorithm try to exclude any occurrence of such consecutive three ordered trees.
Note that even when $rpl(T_{u-1}) = rll(T_{u+1}) = 1, rpl(T_u) > 1$ and $C(T_u, 1)$ is
the leftmost child of $T_u$, if $T_u$ has the pony-tail and $T_{u+1}$ is copying $T_u$ (see
Fig.5(c)), then we can set $C(T_u, 2)$ at the rightmost child of $T_i$ and $C(T_{u+1}, 2)$
at the leftmost child of $T_{i+1}$ (by Lemma4(e')) and we can proceed successfully.
(See an example in Fig.5(d).)
Algorithm First we set $C(T_1, 1)$ as the leftmost child of $T_1$.
Assume that we have done each step 1, 2, · · · , $i - 1$. Now we execute the next step $i$ of our algorithm if $T_{i+1}$ exists. (If $T_i$ is the last ordered tree in the Gray code of $S_k$ then we order the remaining child of $T_i$ with decreasing order of $rpl$ from left to right. See Fig. 4(b). Note that if $rpl(T_i) \geq 3$ then $C(T_i, 1)$ never appear at the second leftmost child of $T_i$.)

We have the following four cases for step $i$.

**Case 1:** $rpl(T_i) = 1$ and $rpl(T_{i+1}) = 1$.

**Case 1a:** If $C(T_i, 1)$ is the leftmost child of $T_i$ then we set $C(T_i, 2)$ as the rightmost child of $T_i$ and $C(T_{i+1}, 2)$ as the leftmost child of $T_{i+1}$ (by Lemma 4(b)).

**Case 1b:** Otherwise, $C(T_i, 1)$ is not the leftmost child of $T_i$ then we set $C(T_i, 1)$ as the rightmost child of $T_i$ and $C(T_{i+1}, 1)$ as the leftmost child of $T_{i+1}$ (by Lemma 4(a)).

**Case 2:** $rpl(T_i) = 1$ and $rpl(T_{i+1}) > 1$.

We have two subcases.

**Case 2a:** $T_{i+1}$ has no pony-tail. (So $T_{i+1}$ is copying $T_i$.)

**Case 2a1:** If $C(T_i, 1)$ is the leftmost child of $T_i$ then we set $C(T_i, 2)$ as the rightmost child of $T_i$ and $C(T_{i+1}, 1)$ as the leftmost child of $T_{i+1}$ (by Lemma 4(d)).

**Case 2a2:** If $C(T_i, 1)$ is not the leftmost child of $T_i$ then we set $C(T_i, 1)$ as the rightmost child of $T_i$ and $C(T_{i+1}, 1)$ as the leftmost child of $T_{i+1}$ (by Lemma 4(a)).

**Case 2b:** $T_{i+1}$ has the pony-tail and $T_i$ is copying $T_{i+1}$.

**Case 2b1:** If $C(T_i, 1)$ is the leftmost child of $T_i$ then we set $C(T_i, 2)$ as the rightmost child of $T_i$ and $C(T_{i+1}, 2)$ as the leftmost child of $T_{i+1}$ (by Lemma 4(e)).

**Case 2b2:** If $C(T_i, 1)$ is not the leftmost child of $T_i$ then we set $C(T_i, 1)$ as the
rightmost child of $T_i$ and $C(T_{i+1}, 1)$ as the leftmost child of $T_{i+1}$ (by Lemma 4(a)).

Case 2c: $T_{i+1}$ has the pony-tail and $T_{i+1}$ is copying $T_i$.

Case 2c1: If $C(T_i, 1)$ is the leftmost child of $T_i$ then we set $C(T_i, 2)$ as the rightmost child of $T_i$ and $C(T_{i+1}, 1)$ as the leftmost child of $T_{i+1}$ (by Lemma 4(f)).

Case 2c2: If $C(T_i, 1)$ is not the leftmost child of $T_i$ then we set $C(T_i, 1)$ as the rightmost child of $T_i$ and $C(T_{i+1}, 1)$ as the leftmost child of $T_{i+1}$ (by Lemma 4(a)).

Case 3: $rpl(T_i) > 1$ and $rpl(T_{i+1}) = 1$.

We have two subcases.

Case 3a: $T_i$ has no pony-tail. (So $T_i$ is copying $T_{i+1}$.)

Case 3a1: If $C(T_i, 1)$ is the leftmost child of $T_i$ then we can prove that this case never occur, as follows.

We have set $C(T_i, 1)$ as the leftmost child of $T_i$ with $rpl(T_i) > 1$ in the preceding step of either Case 2a1, 2a2, 2b2, 2c1 or 2c2. In those cases $rpl(T_{i-1}) = 1$ holds, and in Case 3a1 $rpl(T_i) > 1$ and $rpl(T_{i+1}) = 1$ hold and $T_i$ has no pony-tail. This contradicts to (co1).

Case 3a2: If $C(T_i, 1)$ is not the leftmost child of $T_i$ then we set $C(T_i, 1)$ as the rightmost child of $T_i$ and $C(T_{i+1}, 1)$ as the leftmost child of $T_{i+1}$ (by Lemma 4(a)). Set other child ordered trees of $T_i$ between the leftmost child and the rightmost child with decreasing order of $rpl$ from left to right.

Case 3b: $T_i$ has the pony-tail and $T_{i+1}$ is copying $T_i$.

Case 3b1: If $C(T_i, 1)$ is the leftmost child of $T_i$ then we set $C(T_i, 2)$ as the rightmost child of $T_i$ and $C(T_{i+1}, 2)$ as the leftmost child of $T_{i+1}$ (by Lemma 4(c')). Set the remaining child $C(T_i, 3)$ of $T_i$ as the middle child of $T_i$.

Case 3b2: If $C(T_i, 1)$ is not the leftmost child of $T_i$ then we set $C(T_i, 1)$ as the rightmost child of $T_i$ and $C(T_{i+1}, 1)$ as the leftmost child of $T_{i+1}$ (by Lemma 4(a)). Set the remaining child as the middle child of $T_i$.

Case 3c: $T_i$ has the pony-tail and $T_i$ is copying $T_{i+1}$.

Case 3c1: $C(T_i, 1)$ is the leftmost child of $T_i$. If $T_{i+1}$ is also copying $T_i$ then we set $C(T_i, 2)$ as the rightmost child of $T_i$ and $C(T_{i+1}, 2)$ as the leftmost child of $T_{i+1}$ (by Lemma 4(c)) and set the remaining child as the middle child of $T_i$. Otherwise one can prove that this case never occur. Similar to Case 3a1.

Case 3c2: If $C(T_i, 1)$ is not the leftmost child of $T_i$ then we set $C(T_i, 1)$ as the rightmost child of $T_i$ and $C(T_{i+1}, 1)$ as the leftmost child of $T_{i+1}$ (by Lemma 4(a)). Set the remaining child as the middle child of $T_i$.

Case 4: $rpl(T_i) > 1$ and $rpl(T_{i+1}) > 1$.

Case 4a: $C(T_i, 1)$ is the leftmost child of $T_i$.

Case 4a1: $rpl(T_i) \leq rpl(T_{i+1})$.

We set $C(T_i, rpl(T_i))$ as the rightmost child of $T_i$ and $C(T_{i+1}, rpl(T_i))$ as the leftmost child of $T_{i+1}$ (by Lemma 4(g')).

Set other child ordered trees of $T_i$ between the leftmost child $C(T_i, 1)$ and the rightmost child $C(T_i, rpl(T_i))$ with increasing order of $rpl$ from left to right.

Case 4a2: $rpl(T_i) > rpl(T_{i+1})$.
We set $C(T_i, rpl(T_{i+1}))$ as the rightmost child of $T_i$ and $C(T_{i+1}, rpl(T_{i+1}))$ as the leftmost child of $T_{i+1}$ (by Lemma 4(g)).

Set other child ordered trees of $T_i$ between the leftmost child $C(T_i, 1)$ and the rightmost child $C(T_i, rpl(T_{i+1}))$ with increasing order of $rpl$ from left to right.

**Case 4b:** $C(T_i, 1)$ is not the leftmost child of $T_i$.

Let $T$ be the leftmost child of $T_i$.

**Case 4b1:** $rpl(T_i) \leq rpl(T_{i+1})$.

If $rpl(T_i) < rpl(T_{i+1})$ then we set $C(T_i, 1)$ as the rightmost child of $T_i$ and $C(T_{i+1}, rpl(T_i))$ as the leftmost child of $T_{i+1}$ (by Lemma 4(g')).

Otherwise $rpl(T_i) = rpl(T_{i+1})$ holds. If $rpl(T) = rpl(T_i)$ then we set $C(T_i, rpl(T_i) + 1)$ as the rightmost child of $T_i$ and $C(T_{i+1}, rpl(T_i) + 1)$ as the leftmost child of $T_{i+1}$, and if $rpl(T) \neq rpl(T_i)$ then we set $C(T_i, rpl(T_i))$ as the rightmost child of $T_i$ and $C(T_{i+1}, rpl(T_i))$ as the leftmost child of $T_{i+1}$ (by Lemma 4(g')).

Set other child ordered trees of $T_i$ between the leftmost child and the rightmost child with decreasing order of $rpl$ from left to right. (Note that if $rpl(T_i) \geq 3$ then $C(T_i, 1)$ never appear at the second leftmost child of $T_i$.)

**Case 4b2:** $rpl(T_i) > rpl(T_{i+1})$ and $rpl(T) \neq rpl(T_{i+1})$.

We set $C(T_i, rpl(T_{i+1}))$ as the rightmost child of $T_i$ and $C(T_{i+1}, rpl(T_{i+1}))$ as the leftmost child of $T_{i+1}$ (by Lemma 4(g')). Set other child ordered trees of $T_i$ between the leftmost child and the rightmost child with decreasing order of $rpl$ from left to right. (Note that $C(T_i, 1)$ never appear at the second leftmost child of $T_i$ since $rpl(T_i) \geq 3$ holds.)

**Case 4b3:** $rpl(T_i) > rpl(T_{i+1})$ and $rpl(T) = rpl(T_{i+1})$.

We show this case never occur in the lemma below.

The description of the four cases for step $i$ is completed.

We have the following three lemmas.

**Lemma 5.** Case 4b3 never occur.

*Proof.* Assume for a contradiction that the case occurs. (In Case 4b we have defined $T$ as the leftmost child of $T_i$.)

If $rpl(T) > 2$, then we have set $T$ in Case 4 of the preceding step $i - 1$. If $rpl(T_{i-1}) \leq rpl(T_i)$ and we set $C(T_i, rpl(T_{i-1}))$ as $T$ in either Case 4a1 or Case 4b1, then $rpl(T_{i-1}) = rpl(T) = rpl(T_{i+1}) < rpl(T_i)$ holds, which contradicts to (c1). Otherwise if $rpl(T_{i-1}) = rpl(T_i)$ and we set $C(T_i, rpl(T_i))$ or $C(T_i, rpl(T_i) + 1)$ as $T$ in Case 4b1, then it contradict to the condition $rpl(T_i) > rpl(T_{i+1})$ and $rpl(T) = rpl(T_{i+1})$ of Case 4b3. Otherwise, $rpl(T_{i-1}) > rpl(T_i)$ holds, then we set $C(T_i, rpl(T_i))$ as $T$ in either Case 4a2 or Case 4b2, so $rpl(T) = rpl(T_i)$ holds, which contradicts to Case 4b3.

If $rpl(T) = 2$, then we set $T$ in either Case 2b1, 4a1, 4a2, 4b1 or 4b2 of the preceding step $i - 1$. If we set $T$ in Case 2b1 then $T_i$ has the pony-tail and $rpl(T_i) = 2$, which contradicts to $rpl(T_i) > rpl(T_{i+1}) > 1$. If we set $T$ in Case 4a1 or Case 4b1 then $rpl(T_{i-1}) = rpl(T) = rpl(T_{i+1}) < rpl(T_i)$,
which contradicts to (co1). If we set $T$ in either Case 4a2 or Case 4b2 then $rpl(T_{i-1}) > rpl(T_i) = rpl(T)$ which contradicts to Case 4b3.

Lemma 6. (a) If $rpl(T) = 1$, $T'$ has no pony-tail and $T'$ is copying $T$, then $C(T', 1)$ is copying $C(T, 2)$.
(b) If $rpl(T) = 1$, $T'$ has the pony-tail and $T'$ is copying $T$, then $C(T', 1)$ is copying $C(T, 2)$.

Proof. (Sketch.) See Fig. 6.

We need above lemma in the proof of the next lemma.

Lemma 7. Assume that (co1) is satisfied. If (co2) is satisfied for $i = 1, 2, \ldots, s$ then, after executing step $i = s$, (co2) is satisfied for $i = s + 1$.

Proof. First part of (co2) We have the following three cases to consider. For each case we can prove (co2) is satisfied for $i = s + 1$, as follows.

Case 1: $T'_{u-1}$ is the rightmost child of $T_{s-1}$, $T'_u$ is the leftmost child of $T_s$ and $T'_{u+1}$ is the second leftmost child of $T_s$.

If those three ordered trees violate (co2) then $rpl(T'_{u-1}) = rpl(T'_{u+1}) = 1 < rpl(T'_u)$ holds.

Only Case 4b1 with $rpl(T_{s-1}) < rpl(T_s)$ sets $T'_{u-1}$ and $T'_u$, so that $rpl(T'_{u-1}) = 1 < rpl(T'_u)$. If so $rpl(T_s) \geq 3$ holds. However no case set (the second leftmost child of $T_s$) $T'_{u+1}$ with $rpl(T'_{u+1}) = 1$ since if $rpl(T_s) \geq 3$ then no case set $C(T_s, 1)$ as the second leftmost child of $T_s$. Thus (co2) is satisfied.

Case 2: $T'_{u-1}$, $T'_u$ and $T'_{u+1}$ are children of $T_s$.
Those three ordered trees never violate (co2) since they are children of \( T_s \) and have distinct \( rpl \)’s.

**Case 2:** \( T_{u'}^0 \) is the second rightmost child of \( T_{s-1} \), \( T_{u'} \) is the rightmost child of \( T_{s-1} \) and \( T_{u'+1} \) is the leftmost child of \( T_s \).

If those three ordered trees violate (co2) then \( rpl(T_{u'-1}') = rpl(T_{u'+1}') = 1 < rpl(T_u') \) holds. This occurs only when we set \( T_{u'} \) and \( T_{u'+1} \) in either Case 2a1 or Case 2c1. For those cases \( rpl(T_{s-1}) = 1 \) holds, and \( rpl(T_{u'-1}') = rpl(T_{u'+1}') = 1 \), \( T_u' \) has the pony-tail and \( T_{u'+1} \) is copying \( T_u' \) by Lemma 6(a) and (b). Thus (co2) is satisfied.

**Second part of (co2)** If \( T_{u'-1}' \), \( T_{u}' \) and \( T_{u'+1}' \) are siblings, since each child ordered tree has a distinct \( rpl \), the claim is satisfied. So assume otherwise, that is \( T_{u'-1}' \) and \( T_{u'+1}' \) are not siblings. We have the following two cases.

**Case 1:** \( T_{u'} \) and \( T_{u'+1} \) are not siblings.

Now \( T_{u'-1}' \) and \( T_{u}' \), are siblings. If \( T_{u'-1}', T_{u}' \) violate (co2) then \( 2 \leq rpl(T_{u'-1}') < rpl(T_u') \) and \( rpl(T_{u'}) > rpl(T_{u'+1}') \geq 2 \) hold. No case set \( T_{u'} \) and \( T_{u'+1} \) with \( rpl(T_{u'}) > rpl(T_{u'+1}') \geq 2 \). Thus this case never occur.

**Case 2:** \( T_{u'-1}' \) and \( T_{u'} \) are not siblings.

Now \( T_{u'} \) and \( T_{u'+1} \) are siblings. If \( T_{u'-1}', T_{u}' \) violate (co2) then \( 2 \leq rpl(T_{u'+1}') < rpl(T_u') \) and \( rpl(T_{u'}) > rpl(T_{u'-1}') \geq 2 \) hold. No case set \( T_{u'-1}' \) and \( T_{u}' \) with \( rpl(T_{u'}) > rpl(T_{u'-1}') \geq 2 \). Thus this case never occur. \( \Box \)

Now we have the following theorem.

**Theorem 8.** There is a Gray code for ordered trees with \( n \) vertices such that each ordered tree is derived from the preceding ordered tree by removing a leaf then appending a leaf.

By constructing the necessary part of \( F_n \) on the fly one can generate each ordered tree in a Gray code for \( S_n \) in \( O(n^2) \) time for each ordered tree.

### 4 Conclusion

In this paper we have designed a Gray code for ordered trees with \( n \) vertices such that each ordered tree is derived from the preceding ordered tree by removing a leaf then appending a leaf.

Can we design a Gray code for binary trees with \( n \) vertices such that each binary tree is derived from the preceding binary tree by removing a leaf then appending a leaf?

### References

[1] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Generating All Trees, History of Combinatorial Generation.* Addison-Wesley, 2006.
[2] Joan M. Lucas. The rotation graph of binary trees is hamiltonian. *J. Algorithms*, 8(4):503–535, 1987.

[3] Joan M. Lucas, Dominique Roelants van Baronaigien, and Frank Ruskey. On rotations and the generation of binary trees. *J. Algorithms*, 15(3):343–366, 1993.

[4] Torsten Mütze. Combinatorial gray codes - an updated survey. *CoRR*, abs/2202.01280, 2022.

[5] Shin-Ichi Nakano. Efficient generation of plane trees. *Inf. Process. Lett.*, 84(3):167–172, 2002.

[6] Carla D. Savage. A survey of combinatorial gray codes. *SIAM Rev.*, 39(4):605–629, 1997.