Enumeration of Unlabeled Tree by Dynamic Programming

Ryuji Masui[1], Aleksandar Shurbevski[2], and Hiroshi Nagamochi[3]

Department of Applied Mathematics and Physics, Kyoto University
[1]r.masui@amp.i.kyoto-u.ac.jp
[2]shurbevski@amp.i.kyoto-u.ac.jp
[3]nag@amp.i.kyoto-u.ac.jp

Abstract The enumeration of graphs with a certain specified structure is one of the classical problems in graph theory, and has a variety of applications including the enumeration of chemical compounds. When we enumerate"labeled" graphs, two "labeled" graphs are allowed to be isomorphic, i.e., they are the same "unlabeled" graph where labels on vertices are ignored. Contrary to this, enumerating "unlabeled" graphs means that no two generated graphs are allowed to be isomorphic. Most efficient algorithms for enumerating labeled graphs or unlabeled graphs have been designed based on the branch method, which usually uses a polynomial space and generates a graph one after another obeying a fixed but unseen order of graphs. Dynamic programming is another algorithm design technique based on which a given problem is decomposed into subproblems in a different way from the branch method. In this paper, we study how to design enumeration algorithms based on dynamic programming, taking the problem of enumerating "unlabeled" rooted trees as the first target of this object.

The ranking problem asks us to identify the rank k of a given rooted tree in a certain specified total order over all trees. Contrary to this, the unranking problem is the inverse of the ranking problem, which, given integers n and k, asks us to construct the k-th rooted tree with n vertices over a specified total order on all trees. We introduce a total order over all "unlabeled" rooted trees so that we can design both ranking and unranking algorithms based

[1]Technical Report 2019-003, July 25, 2019

on dynamic programming. Each of our ranking and unranking algorithms runs in polynomial time of n, which is independent of the number of all rooted trees with n vertices.

In order to design an enumeration/ranking/unranking algorithm, we use a mapping from ordered trees to sequences and a mapping from rooted trees to ordered trees. By these two mappings, rooted trees can be represented as sequences. We define a total order over all rooted trees to be the lexicographical ascending order over the corresponding sequences. Based on this total order over rooted trees, we show that every rooted tree can be represented as a sequence that consists of the number of vertices and the rank of subtrees connected to the root. From the observation, we treat the ranking/unranking problems as problems on certain types of sequences.

1 Introduction

The enumeration of graphs with a restricted structure is a classical problem in graph theory. Roughly speaking, we can divide the problem settings of enumeration for graphs into two types; the enumeration for labeled graphs and the enumeration for unlabeled graphs. A labeled graph is a graph such that each vertex has a unique label. Given a fixed set of vertex labels, the enumeration problem for labeled graphs is to generate all labeled graphs which is distinguishable from each other by the label of vertices or edge connectivity. Prüfer [15] showed a one-to-one mapping between all labeled trees with n vertices and the set $\{(s_1, s_2, \ldots, s_{n-2}) \mid 1 \leq s_i \leq n\}$ of integer sequences with length $n-2$. This implies that the number of all labeled trees with n vertices is n^{n-2}, and the problem to enumerate all labeled trees is equivalent to the problem to enumerate all integer sequences in $\{(s_1, s_2, \ldots, s_{n-2}) \mid 1 \leq s_i \leq n\}$. On the other hand, the enumeration problem for unlabeled graphs is to generate all non-isomorphic graphs, which means we ignore the label of vertices, and we distinguish by edge connectivity. Note that, depending on the definition of the isomorphism for graphs, the output of the enumeration problem is different. For instance, in the case of the enumeration for unlabeled trees, we should be careful about unrooted trees, rooted trees, and ordered trees.

We show an example of enumeration for all labeled trees, ordered trees, rooted trees, and unrooted trees with 4 vertices in Figure 1. Figure 1 (a) shows all labeled trees with 4 vertices but most of them are omitted due to space limitations. For each tree, we put the Prüfer code of the labeled tree. Given a labeled tree, its Prüfer code can be constructed as follows. We choose a leaf with the smallest label, and the first entry of Prüfer code is the label of neighbor of the selected leaf. Then we eliminate the leaf from the tree, and we continue the same process while the current graph has more than

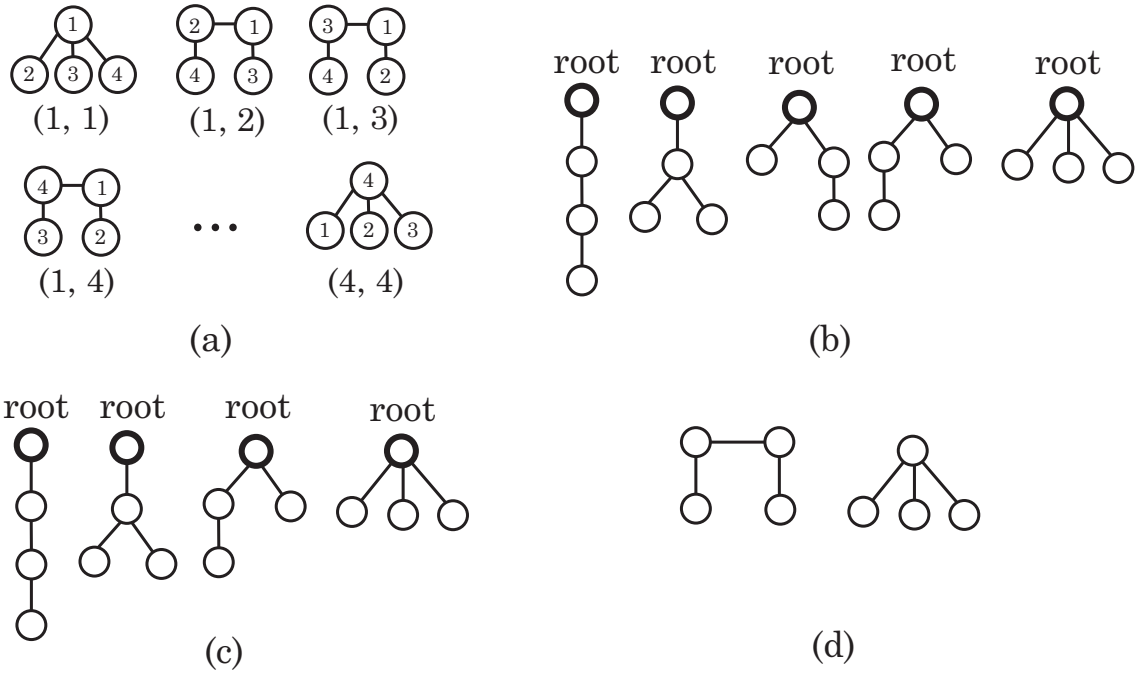

Figure 1: (a) All labeled trees with 4 vertices (omitted); (b) All ordered trees with 4 vertices; (c) All rooted trees with 4 vertices; and (d) All unrooted trees with 4 vertices.

2 vertices. On the other hand, given a Prüfer code with length $n-2$, we can construct the structure of corresponding labeled tree as follows. Let $S = \{1, 2, \ldots, n\}$, let P be a Prüfer code, and let S_P denote the set of integers in P. For the smallest integer i in $S \setminus S_P$ and the first entry j of P, we add an edge ij, and we eliminate i from S and the first entry of P. We continue the same process until P will be empty. Finally S will contain exactly two integers, and we add an edge between two vertices whose label are in S. Figure 1 (b) shows all ordered trees with 4 vertices. An ordered tree has a special vertex called "root" and the left-to-right order for children (Definition is given in Section 3.2). There are 5 ordered tree with 4 vertices. Figure 1 (c) shows all rooted trees with 4 vertices. A Rooted tree is a tree with a fixed root vertex. There are 4 rooted trees with 4 vertices. Figure 1 (d) shows all unrooted trees with 4 vertices. There are 2 unrooted trees with 4 vertices.

The enumeration of chemical graphs is an applications of graph enumeration and an important problem which has a long history, and has been widely applied in drug design [5] and structure elucidation [11]. Chemical compounds can be represented as labeled multigraphs whose vertices correspond to atoms and edges correspond to chemical bonds. Then, enumerating chemical compounds can be seen as the problem of enumerating restricted graphs, which is one of the most fundamental problems in

graph theory, and has numerous other applications in fields such as machine learning.

Molgen [3] is considered to be one of the best available enumeration tools for chemical compounds. OMG [14] is a recent open source program for enumerating chemical compounds. The algorithms of these enumeration tools for chemical compounds are not the most efficient because they treat general graph structures. Recently, studies have focused on designing efficient algorithms for enumerating restricted chemical compounds. One such tool is Enumol2[1], which can efficiently enumerate chemical compounds with tree-like structure, and the research leading to its development [6, 7, 16] has been identified as a new trend in chemoinformatics [18].

Alkanes are acyclic chemical compounds that only consist of hydrogen and carbon atoms, and single bonds between them. An alkane isomer can be represented as a tree with maximum degree at most four. Aringhieri et al. [2] designed an algorithm for enumerating all alkane isomers. The algorithm can generate each alkane isomer with n carbon atoms in $\mathcal{O}(n^4)$ time. Algorithms to generate trees with n vertices in general are available in the literature [4, 13, 19]. For applications to chemical graph enumeration, all vertices in a tree have small degrees, since atoms have a determined valence. Hence we are interested in developing an algorithm to enumerate trees with bounded degree.

Amani and Nowzari-Dalini [1] developed an algorithm to enumerate all degree-bounded ordered trees in a specific order in constant time on average per tree, but this algorithm enumerates topologically identical trees multiple times since they differ as ordered trees. However, in addition, they propose polynomial time algorithms for the related *ranking* and *unranking* problems. The ranking problem is to determine the rank of a given ordered tree in the specific order. On the other hand, the unranking problem is the inverse problem of ranking, that is, given a positive integer k, generate the k-th ordered tree according to the order. The unranking algorithm and enumeration algorithm provides a partial enumeration, and we can obtain any ordered tree following in practical time even if the number of vertices is sufficient large.

Zhuang and Nagamochi [20] gave a branch and bound based algorithm to generate all degree-bounded unrooted trees with n vertices and diameter at least d. This algorithm can generate all trees in constant time per tree and $\mathcal{O}(n)$ space in total. This algorithm is based on the technique called *family tree*. We define a parent-child relationship between enumeration targets, then we obtain a family tree such that the vertices correspond to enumeration targets, and the edges shows the parent-child relationship. By traversing the family tree, we can generate all enumeration targets without repetitions. However, it cannot give the total number of enumeration targets or directly obtain an arbitrary tree without generating all of them. It is tough to estimate the running time of the algorithm since we do not have the information about the total number of enumeration targets. In addition, for sufficiently large n, the algorithm gen-

[1]A demo available online at `http://sunflower.kuicr.kyoto-u.ac.jp/tools/enumol2/`

erates some outputs, but we cannot obtain the structure which will be generated the last of the algorithm in practical time.

In this paper, we focus on enumeration for rooted trees on n vertices with the maximum degree constraint, and we propose a dynamic programming based enumeration algorithm. Note that the maximum possible degree in a tree with n vertices is $n-1$, and hence the assumption on the degree bound can be eliminated by setting the degree bound to be $n-1$.

We introduce a mapping from ordered trees to sequences and a mapping from rooted trees to ordered trees. By these two mapping, rooted trees can be represented as sequences. We define an order for rooted trees to be the lexicographical ascending order of the corresponding sequences. Based on the order for rooted trees, we show that a rooted tree can be represented as the number of vertices and the rank of subtrees connected to the root. From these observation, we propose a 2-Phase enumeration algorithm for rooted trees. The first phase counts the number of rooted trees by dynamic programming. In the second phase, by using the counting information, the algorithm generates the predecessor tree one by one following the prescribed tree ordering.

In addition, by using the counting information obtained in Phase 1, we develop a polynomial time ranking/unranking algorithm. The ranking problem is to determine the rank of a given rooted tree in the tree ordering. Conversely, the unranking problem is to generate the k-th rooted tree following the order. The unranking algorithm and the sequential enumeration algorithm provide a partial enumeration, and we can obtain any consecutive part of rooted trees in practical time even if the number of vertices is sufficient large. Moreover, our enumeration algorithm can be easily parallelized by the partial enumeration.

The remaining of the thesis is organized as follows. In Section 2, we define some sets of integer sequences, and we show ranking/unranking algorithm for the sets. In addition, we propose an algorithm to generate the predecessor/successor sequence. These algorithm are used as a subroutine in our algorithms for enumeration/ranking/unranking for trees. Section 3 introduces the definition and notions of graphs that are used further, and we give a definition of a tree ordering. In Section 3, we also discuss a dynamic programing based counting algorithm for degree-bounded trees. Our enumeration algorithm for degree-bounded trees with n vertices is presented in Section 4 with $\mathcal{O}(n)$ time per tree. For trees with n vertices and the maximum degree bound Δ, after $\mathcal{O}(n^2\Delta^2)$ time dynamic programming based counting algorithm, the ranking algorithm is designed in Section 5 with $\mathcal{O}(\min\{n\Delta^2, n^2\})$ time complexity, and the unranking algorithms is proposed in Section 6 with $\mathcal{O}(\min\{n^2\Delta^2 \log \Delta, n^3 \log \Delta\})$ time complexity. In order to examine the practical performance of our enumeration algorithm, we implemented our sequential enumeration, and we show two experimental results about tree enumeration in Section 7. In section 8, we present our conclusions and future works.

2 Sequences

In this section, we introduce the definitions and notations for sequences, and we define some sets of integer sequences. For a set of integer sequences, we will discuss an algorithm to generate the predecessor/successor of a given sequence, and we will propose a ranking/unranking algorithm. In section 2.1, we mention the sets of integer sequences representing combination with/without repetitions. We show a bijection preserving lexicographical order between a set of integer sequences representing combination with repetitions and a set of integer sequences representing combination without repetitions. Based on this property, we propose a ranking/unranking algorithm for a set of integer sequences representing combination with repetitions by using the algorithm for a set of integer sequences representing combination without repetitions. In section 2.2, we introduce the *upper bounded sequences* which is equivalent to a concatenation of sequences representing combination with repetition. For a set of upper bounded sequences, we propose an algorithm to generate the lexicographic predecessor/successor of a given upper bounded sequence. In addition, we propose an algorithm for ranking/unranking of a given upper bounded sequences by using the ranking/unranking for a set of integer sequences representing combination with repetitions. Finally, in section 2.3, we introduce *positive descending sequence*, and we show an algorithm to construct the lexicographic predecessor in a set of positive descending sequences.

Let $\mathbb{Z}$ denote the set of all integers. Given two integers $a, b \in \mathbb{Z}$, let $[a, b]$ denote the set $\{x \in \mathbb{Z} \mid a \leq x \leq b\}$. For a set M, let $|M|$ denote the number of elements in M. For an integer sequence S, let $|S|$ denote the length of S, let $\max(S)$ denote the maximum entry of S, and let $\mathrm{sum}(S)$ denote the sum of all entries in S.

Let $\mathcal{S}$ be a set of integer sequences, and let σ be a total order over $\mathcal{S}$. Let $\mathrm{rank}_{\mathcal{S},\sigma} : \mathcal{S} \to [1, |\mathcal{S}|]$ denote the function such that $\mathrm{rank}_{\mathcal{S},\sigma}(S_1) < \mathrm{rank}_{\mathcal{S},\sigma}(S_2)$ if S_1 precedes S_2 in the order σ. We say that a sequence $S \in \mathcal{S}$ with $\mathrm{rank}_{\mathcal{S},\sigma}(S) = k$ is the k-th sequence in $\mathcal{S}$, or the rank of such a sequence S is k. Given a sequence S in $\mathcal{S}$ and a total order σ over $\mathcal{S}$, the *ranking problem* is to construct $\mathrm{rank}_{\mathcal{S},\sigma}(S)$. On the other hand, the *unranking problem* is the inverse problem of ranking, that is, given a positive integer $k \in [1, |\mathcal{S}|]$, generate the k-th sequence S in $\mathcal{S}$.

Let $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_m)$ be sequences in $\mathcal{S}$. We say that X is *lexicographically smaller* than Y, and denote this by $X \prec Y$, if there exists an integer $k \in [1, \min\{n, m\}]$ such that for all $i \in [1, k]$, $x_i = y_i$, and either $k = n < m$, or, $k < \min\{n, m\}$ and $x_{k+1} < y_{k+1}$. We also say that X is *lexicographically greater* than Y if $Y \prec X$. We define the *lexicographical ascending order* σ_{lex} to be the total order such that $\sigma_{\mathrm{lex}}(X) \prec \sigma_{\mathrm{lex}}(Y)$ if $X \prec Y$ holds. We call the reverse order of σ_{lex} the *lexicographical descending order*. Moreover, we define the co-lexicographical ascending order σ_{colex} to be the total order such that $\sigma_{\mathrm{colex}}(X) \prec \sigma_{\mathrm{colex}}(Y)$ if $(x_n, \ldots, x_2, x_1) \prec$

$(y_m, \ldots, y_2, y_1)$ holds. Let $\mathcal{S}$ be a set of integer sequences, and let X be a sequence in $\mathcal{S}$. We say that X is *lexicographically minimum* (resp., *lexicographically maximum*) in $\mathcal{S}$ if $\mathrm{rank}_{\mathcal{S},\sigma_{\mathrm{lex}}}(X) = 1$ (resp., $\mathrm{rank}_{\mathcal{S},\sigma_{\mathrm{lex}}}(X) = |S|$). Let $\min(\mathcal{S})$ (resp., $\max(\mathcal{S})$) denote the lexicographically minimum (resp., maximum) sequence in $\mathcal{S}$. For a sequence $Y \in \mathcal{S}$, we say that Y is the *lexicographic predecessor* (resp., *lexicographic successor*) of X, if $\mathrm{rank}_{\mathcal{S},\sigma_{\mathrm{lex}}}(Y) = \mathrm{rank}_{\mathcal{S},\sigma_{\mathrm{lex}}}(X) - 1$ (resp., $\mathrm{rank}_{\mathcal{S},\sigma_{\mathrm{lex}}}(Y) = \mathrm{rank}_{\mathcal{S},\sigma_{\mathrm{lex}}}(X) + 1$) holds. Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be sets of integer sequences, and let f be a bijection from $\mathcal{S}_1$ to $\mathcal{S}_2$. Let f^{-1} denote the inverse function of f. We say that f *preserves the lexicographical order* if for sequences $S_1, S_2 \in \mathcal{S}_1$, $f(S_1) \prec f(S_2)$ holds if and only if $S_1 \prec S_2$.

2.1 Combination Sequences

For two positive integers $n \geq 1$ and $m \geq n$, let $\mathcal{C}(n, m)$ denote the set $\{(c_1, c_2, \ldots, c_n) \mid m > c_1 > c_2 > \cdots > c_n \geq 0\}$ of integer sequences. The set $\mathcal{C}(n, m)$ is known as a representation of combinations without repetitions of n out of m elements, and it holds that $|\mathcal{C}(n, m)| = \binom{n}{m}$.

For two positive integers $n \geq 1$ and $m \geq n$, and a sequence $C = (c_1, c_2, \ldots, c_n) \in \mathcal{C}(n, m)$, it holds that $\mathrm{rank}_{\mathcal{C}(n,m),\sigma_{\mathrm{lex}}}(C) = \sum_{i=1}^{n} \binom{c_i}{n-i+1} + 1$, where $\binom{c_i}{n-i+1}$ is defined to be 0 if $c_i < n - i + 1$. This result is observed in the book by Knuth [9]. Hence we immediately obtain the following lemma.

Lemma 1. *Let $n \geq 1$ and $m \geq n$ be two positive integers, and let $C = (c_1, c_2, \ldots, c_n)$ be a sequence in $\mathcal{C}(n, m)$. Then it holds that $\mathrm{rank}_{\mathcal{C}(n,m),\sigma_{\mathrm{lex}}}(C) = \sum_{i=1}^{n} \binom{c_i}{n-i+1} + 1$, which can be obtained in $\mathcal{O}(n^2)$ time.*

Proof. For each $i \in [1, n]$, the value $\binom{c_i}{n-i+1}$ can be obtained in $\mathcal{O}(n - i + 1)$ time. Therefore the calculation of $\sum_{i=1}^{n} \binom{c_i}{n-i+1} + 1$ can be done in $\mathcal{O}(\sum_{i=1}^{n}(n - i + 1)) = \mathcal{O}(\sum_{i=1}^{n} i) = \mathcal{O}(n^2)$ time. □

Shimizu et al. [17] showed that for the set $\mathcal{C}(n, m)$ and an integer $k \in [1, |\mathcal{C}(n, m)|]$, there exists a total order σ (resp., σ') such that the k-th sequence in $\mathcal{C}(n, m)$ according to the order can be generated in $\mathcal{O}(n \log m)$ (resp., $\mathcal{O}(n^{3n+3})$) time. In the following paper, a total order of integer sequences is fixed to the lexicographical ascending order σ_{lex}. For the sake of simplicity, let $\mathrm{rank}_{\mathcal{S}}(S)$ denote $\mathrm{rank}_{\mathcal{S},\sigma_{\mathrm{lex}}}(S)$. We introduce naive unranking algorithms for the set $\mathcal{C}(n, m)$ following the lexicographical ascending order.

Lemma 2. *Let $n \geq 1$, $m \geq n$, and $k \in [1, |\mathcal{C}(n, m)|]$ be three positive integers, and let $C = (c_1, c_2, \ldots, c_n)$ be the k-th sequence in $\mathcal{C}(n, m)$ following the lexicographical ascending order. Let c be an integer with $\sum_{i=n-1}^{c-1} \binom{i}{n-1} < k \leq \sum_{i=n-1}^{c} \binom{i}{n-1}$, and let $(c'_2, \ldots, c'_n)$ be the $(k - \sum_{i=n-1}^{c-1} \binom{i}{n-1})$-th sequence in $\mathcal{C}(n - 1, c)$. Then it holds that $c_1 = c, c_i = c'_i$ for $i = 2, 3, \ldots, n$, which can be obtained in $\mathcal{O}(n^2 \log m)$ time.*

Proof. For an integer c, the number of sequences in $\mathcal{C}(n,m)$ whose first entry is at most c is $\sum_{i=n-1}^{c}\binom{i}{n-1}$. From the definition of the lexicographical ascending order, if an integer c satisfies $\sum_{i=n-1}^{c-1}\binom{i}{n-1} < k \leq \sum_{i=n-1}^{c}\binom{i}{n-1}$, then the first entry c_1 of the k-th sequence $\mathcal{C}(n,m)$ is c. For two sequences $S = (s_1, s_2 \ldots, s_n)$ and $S' = (s'_1, s'_2, \ldots, s'_m)$ with $s_1 = s'_1$, it holds that $(s_2, s_3, \ldots, s_n) \prec (s'_2, \ldots, s'_m)$ if and only if $S \prec S'$. Therefore the sequence $(c_2, \ldots, c_n)$ must be the $(k - \sum_{i=n-1}^{c-1}\binom{i}{n-1})$-th sequence in the set $\{(c_2, \ldots, c_n) \mid c > c_1 > c_2 > \cdots > c_n \geq 0\} = \mathcal{C}(n-1, c)$ of integer sequences.

The first entry of the k-th sequence of $\mathcal{C}(n,m)$ can be obtained in $\mathcal{O}(n \log m)$ time by using binary search. Hence we get that the total time complexity is $\mathcal{O}(n^2 \log m)$. □

For positive integers $n \geq 1$ and $m \geq 1$, let $\mathcal{R}(n,m)$ denote the set $\{(r_1, r_2, \ldots, r_n) \mid m > r_1 \geq r_2 \geq \cdots \geq r_n \geq 0\}$ of integer sequences. The set $\mathcal{R}(n,m)$ is known as a representation of combinations with repetitions of n out of m elements. For a sequence $R = (r_1, r_2, \ldots, r_n)$ in $\mathcal{R}(n,m)$, and the function $f(R) = (c_1, c_2, \ldots, c_n)$ such that $c_i = r_i + n - i$, the sequence $f(R)$ belongs to $\mathcal{C}(n, m+n-1)$. Such a function f is a bijection between $\mathcal{R}(n,m)$ and $\mathcal{C}(n, m+n-1)$. In addition, for two sequences $R_1, R_2 \in \mathcal{R}(n,m)$, we get that $f(R_1) \prec f(R_2)$ if and only if $R_1 \prec R_2$. Therefore by Lemmas 1 and 2, we have the following lemma.

Lemma 3. *Let $n \geq 1$ and $m \geq 1$ be two positive integers.*

(i) *The rank of a sequence $R \in \mathcal{R}(n,m)$ can be obtained in $\mathcal{O}(n^2)$ time.*

(ii) *For an integer $k \in [1, |\mathcal{R}(n,m)|]$, the k-th sequence in $\mathcal{R}(n,m)$ can be obtained in $\mathcal{O}(n^2 \log(n+m))$ time.*

Proof. For positive integers n and m, and a sequence $R = (r_1, r_2, \ldots, r_n)$ in $\mathcal{R}(n,m)$, we define a function $f(R) = (c_1, c_2, \ldots, c_n)$ such that $c_i = r_i + n - i$. From the definition of $\mathcal{R}(n,m)$, since we have $m > r_1 \geq r_2 \geq \cdots \geq r_n \geq 0$, we obtain $m+n-1 > c_1 > c_2 > \cdots > c_n \geq 0$. Hence the sequence $f(R)$ belongs to $\mathcal{C}(n, m+n-1)$. Conversely, for a sequence $C' = (c'_1, c'_2, \ldots, c'_n) \in \mathcal{C}(n, m+n-1)$, the sequence $R' = (r'_1, r'_2, \ldots, r'_n)$ such that $r'_i = c'_i - n + i$ belongs to $\mathcal{R}(n,m)$. In addition, the function f is injective. Therefore the function f is a bijection between $\mathcal{R}(n,m)$ and $\mathcal{C}(n, m+n-1)$. Moreover for two sequence $R_1, R_2 \in \mathcal{R}(n,m)$, we get that $f(R_1) \prec f(R_2)$ if and only if $R_1 \prec R_2$. Hence the bijection f preserves the lexicographical order.

Let R be a sequence $\mathcal{R}(n,m)$. The rank of the sequence R is equal to the rank of $f(R)$ in $\mathcal{C}(n, m+n-1)$. We can obtain the sequence $f(R)$ in $\mathcal{O}(n)$ time, and by Lemma 1, the rank of $f(R)$ can be obtained in $\mathcal{O}(n^2)$ time. Hence the rank of R can be obtained in $\mathcal{O}(n^2)$ time.

Let C denote the k-th sequence in $\mathcal{C}(n, m+n-1)$. Since the function f preserves the lexicographical order, $f^{-1}(C)$ is the k-th sequence in $\mathcal{R}(n,m)$. From Lemma 2,

the k-th sequence C of $\mathcal{C}(n, m+n-1)$ can be obtained in $\mathcal{O}(n^2 \log(n+m))$ time. In addition, given the sequence C, $f^{-1}(C)$ can be constructed in $\mathcal{O}(n)$ time. Therefore the k-th sequence of $\mathcal{R}(n, m)$ can be obtained in $\mathcal{O}(n^2 \log(n+m))$ time. □

2.2 Upper Bounded Sequences

Given a positive integer sequence $K = (k_1, k_2, \ldots, k_n)$ with length $n \geq 1$, we say that an integer sequence $S = (s_1, s_2, \ldots, s_n)$ is *upper bounded* by K if it holds that $0 \leq s_i < k_i$ for each $1 \leq i \leq n$, and $s_j \geq s_{j+1}$ for each $1 \leq j \leq n-1$ with $k_j = k_{j+1}$. Let $\mathcal{S}(K)$ denote the set of all sequences upper bounded by K. Given a positive integer sequence K and a sequence $S \in \mathcal{S}(K)$, we devise a method to generate the lexicographic predecessor of S in $\mathcal{S}(K)$.

Lemma 4. *Let $K = (k_1, k_2, \ldots, k_n)$ be a positive integer sequence with length $n \geq 1$, and let $S = (s_1, s_2, \ldots, s_n)$ be a sequence in $\mathcal{S}(K)$. Then:*

(i) *If $s_i = 0$ for all $i \in [1, n]$, then $S = \min(\mathcal{S}(K))$; and*

(ii) *If $s_i \neq 0$ for some $i \in [1, n]$, for the largest index p such that $s_p \geq 1$ and the largest index q such that $k_p = k_{p+1} = \cdots = k_q$, the lexicographic predecessor of S in $\mathcal{S}(K)$ is given as $S' = (s_1, s_2, \ldots, s_{p-1}, s'_p, s'_{p+1}, \ldots, s'_n)$ such that $s'_i = s_p - 1$, $p \leq i \leq q$ and $s'_i = k_i - 1$, $q < i \leq n$.*

Proof. (i) We prove that the sequence $S^* = (0, 0, \ldots, 0)$ with length n is $\min(\mathcal{S}(K))$. Clearly, S^* belongs to $\mathcal{S}(K)$, and for each entry, 0 is the minimum possible value in $\mathcal{S}(K)$. Hence there exists no lexicographic predecessor of S^*, and we have $S^* = \min(\mathcal{S}(K))$.

(ii) We prove that if $S \neq \min(\mathcal{S}(K))$, then the sequence S' is the lexicographic predecessor of S. Clearly, we have $S' \in \mathcal{S}(K)$ and $S' \prec S$. Suppose that there exists a sequence $X = (x_1, x_2, \ldots, x_n)$ in $\mathcal{S}(K)$ such that $S' \prec X \prec S$. For each $1 \leq i < p$, it holds that $x_i = s_i$. First we assume that $x_p = s_p$ holds. In this case, from the assumption of p, we have $s_i = 0$, $p < i \leq n$. This, however, contradicts that $X \prec S$ since $x_i \geq 0$, $p < i \leq n$, and we obtain $x_p \neq s_p$. Next we assume that $x_p = s_p - 1$ holds. From the definition of upper bounded, we have $x_i \leq s_q - 1$, $p < i \leq q$ and $x_i \leq k_i - 1$, $q < i \leq n$. This, however, contradicts that $S' \prec X$, and we obtain $x_p \neq s_p - 1$. Therefore we see that there exists no sequence X such that $S' \prec X \prec S$, and the sequence S' is the lexicographic predecessor of S in $\mathcal{S}(K)$. □

We show an algorithm based on Lemma 4 to generate the lexicographic predecessor of a given upper bounded sequence if one exists. Let $K = (k_1, k_2, \ldots, k_n)$ be a positive integer sequence, and let $S = (s_1, s_2, \ldots, s_n)$ be a sequence in $S(K)$. First we find the largest index p such that $s_p \geq 1$. If there exists no such p, then each entry of S is 0,

that is, $S = \min(\mathcal{S}(K))$. If $S \neq \min \mathcal{S}(K)$, then we find the greatest index q such that $k_p = k_{p+1} = \cdots = k_q$, and by Lemma 4, we construct the lexicographic predecessor of S in $\mathcal{S}(K)$. A description of the algorithm is shown in Algorithm 1.

Algorithm 1 Generate the lexicographic predecessor of an upper bounded sequence

Input: An integer sequence $K = (k_1, k_2, \ldots, k_n)$ and an upper bounded sequence $S = (s_1, s_2, \ldots, s_n) \in \mathcal{S}(K)$.

Output: If S has no lexicographic precedessor in $S(K)$, then a message, "S is minimum," otherwise the lexicographic predecessor of S in $\mathcal{S}(K)$.

1: **if** $s_i = 0$ for all $1 \leq i \leq n$ **then**
2: **output** message "S is minimum"
3: **else**
4: $p :=$ the largest index such that $s_p > 0$;
5: $q :=$ the largest index such that $k_p = k_{p+1} = \cdots = k_q$;
6: $s'_i := s_p - 1$ for each $p \leq i \leq q$;
7: $s'_i := k_i - 1$ for all $q < i \leq n$;
8: **output** $(s_1, s_2, \ldots, s_{q-1}, s'_p, \ldots, s'_n)$
9: **end if**

Theorem 1. *Let K be a positive integer sequence with length $n \geq 1$, and let S be an upper bounded sequence in $\mathcal{S}(K)$. Then testing whether S is lexicographically minimum sequence in $\mathcal{S}(K)$ or not can be done in $\mathcal{O}(n)$ time, and the lexicographic predecessor of S in $\mathcal{S}(K)$ if one exists can be constructed in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space.*

Proof. In order to determine whether $S = \min(\mathcal{S}(K))$ or not, we examine whether $s_i = 0$ or not for all $1 \leq i \leq n$. This can be done in $\mathcal{O}(n)$ time.

Two integers p and q of Lemma 4 can be obtained in $\mathcal{O}(n)$ time. Since the sequence $S' = (s_1, s_2, \ldots, s_{p-1}, s'_p, \ldots, s'_n)$ such that $s'_i = s_p - 1$, $p \leq i \leq q$, and $s'_i = k_i - 1$, $q < i \leq n$ can be obtained by $\mathcal{O}(n)$ arithmetic operations. Hence the lexicographic predecessor of S in $\mathcal{S}(K)$ can be obtained in $\mathcal{O}(n)$ time. The space complexity is $\mathcal{O}(n)$ since we only store the values of two sequences K and S with length n. □

Next given a positive integer sequence K and a sequence $S \in \mathcal{S}(K)$, we devise a method to generate the lexicographic successor of S in $\mathcal{S}(K)$.

Lemma 5. *Let $K = (k_1, k_2, \ldots, k_n)$ be a positive integer sequence with length $n \geq 1$, and let $S = (s_1, s_2, \ldots, s_n)$ be a sequence in $\mathcal{S}(K)$. Then:*

(i) *If $s_i = k_i - 1$ for each $i \in [1, n]$, then $S = \max(\mathcal{S}(K))$; and*

(ii) *If $s_i \neq k_i - 1$ for some $i \in [1, n]$, for the largest index p such that $s_p < k_p - 1$ and the smallest index q such that $k_q = k_{q+1} = \cdots = k_p$ and $s_q = s_{q+1} = \cdots = s_p$, then the lexicographic successor of S in $\mathcal{S}(K)$ is given as $S' = (s_1, s_2, \ldots, s_{q-1}, s_q + 1, 0, \ldots, 0)$.*

Proof. First we prove that the sequence $S^* = (k_1 - 1, k_2 - 1, \ldots, k_n - 1)$ is $\max(\mathcal{S}(K))$. Clearly, the sequence S^* belongs to $\mathcal{S}(K)$, and for each entry i, $k_i - 1$ is the maximum possible value in $\mathcal{S}(K)$. Hence there exists no lexicographic successor of S^*, and we have $S^* = \max(\mathcal{S}(K))$.

Next we prove that if $S \neq \max(\mathcal{S}(K))$, then the lexicographic successor of S is S'. Since either $s_{q-1} > s_q$ or $k_{q-1} \neq k_{q-1}$ holds, we have $S' \in \mathcal{S}(K)$, and $S \prec S'$ holds. Suppose that there exists a sequence $X = (x_1, x_2, \ldots, x_n)$ such that $S \prec X \prec S'$. For each $1 \leq i < q$, it holds that $x_i = s_i$.

First we assume that $x_q = s_q$. From the assumption of p and q, we have $s_i = s_q$, $q \leq i \leq p$ and $s_i = k_i - 1$, $p < i \leq n$. From the definition of upper bounded, $x_i \leq x_q = s_q$, $q \leq i \leq p$ and $x_i \leq k_i - 1$, $p < i \leq n$ hold. This, however, contradicts that $S \prec X$, and we obtain $x_p \neq s_p$. Next we assume that $x_q = s_q + 1$. We have $x_i \geq 0$ for each $q < i \leq n$. This, however, contradicts that $X \prec S'$, and we obtain $x_q \neq s_q + 1$.

Consequently, we conclude that there exists no sequence X such that $S \prec X \prec S'$, and the sequence S' is the lexicographic successor of S in $\mathcal{S}(K)$. □

We show an algorithm based on Lemma 5 to generate the lexicographic successor of a given upper bounded sequence. Let $K = (k_1, k_2, \ldots, k_n)$ be a positive integer sequence, and let $S = (s_1, s_2, \ldots, s_n)$ be a sequence in $\mathcal{S}(K)$. First we find the largest index p such that $s_p < k_p - 1$. If there exists no such p, then $s_i = k_i - 1$ for $i \in [1, n]$, that is, $S = \max(\mathcal{S}(K))$. We assume that $S \neq \max(\mathcal{S}(K))$. We find the smallest index q such that $k_q = k_{q+1} = \cdots = k_p$ and $s_q = s_{q+1} = \cdots = s_p$, and by Lemma 5, we construct the successor of S in $\mathcal{S}(K)$. A description of the algorithm is shown in Algorithm 2.

Theorem 2. *Let K be a positive integer sequence with length $n \geq 1$, and let S be an upper bounded sequence in $\mathcal{S}(K)$. Then testing whether S is lexicographically maximum sequence in $\mathcal{S}(K)$ or not can be done in $\mathcal{O}(n)$ time, and the lexicographic successor of S in $\mathcal{S}(K)$ if one exists can be obtained in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space.*

Proof. Clearly, for each $1 \leq i \leq n$, the i-th elements of $\max(\mathcal{S}(K))$ is $k_i - 1$. Hence in order to determine whether $S = \max(\mathcal{S}(K))$ or not, we examine whether $s_i = k_i - 1$ or not for each $1 \leq i \leq n$. This can be done in $\mathcal{O}(n)$ time.

Two integers p and q of Lemma 5 can be constructed in $\mathcal{O}(n)$ time. Since the sequence $S' = (s_1, s_2, \ldots, s_{q-1}, s_q + 1, 0, 0, \ldots, 0)$ can be obtained by $\mathcal{O}(n)$ arithmetic operations, we conclude that the lexicographic successor of S in $\mathcal{S}(K)$ can be obtained

Algorithm 2 Generate the lexicographic successor of an upper bounded sequence

Input: An integer sequence $K = (k_1, k_2, \ldots, k_n)$ and an upper bounded sequence $S = (s_1, s_2, \ldots, s_n) \in \mathcal{S}(K)$.

Output: If S has no lexicographic successor in $S(K)$, then a message, "S is maximum," otherwise the lexicographic successor of S in $\mathcal{S}(K)$.

1: **if** $s_i = k_i - 1$ for all $1 \leq i \leq n$ **then**
2: **output** message "S is maximum"
3: **else**
4: $p :=$ the largest index such that $s_p < k_p - 1$;
5: $q :=$ the smallest index such that $k_q = k_{q+1} = \cdots = k_p$ and $s_q = s_{q+1} = \cdots = s_p$;
6: **output** $(s_1, s_2, \ldots, s_{q-1}, s_q + 1, 0, 0, \ldots, 0)$
7: **end if**

in $\mathcal{O}(n)$ time. The space complexity is $\mathcal{O}(n)$ since we only store the values of two sequences K and S with length n. □

Next we mention ranking and unranking for upper bounded sequences in lexicographical ascending order. Without loss of generality, we can regard an upper bounded sequence as a concatenation of some sequences representing a combination with repetition. Given two sequences $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_m)$, let $A \otimes B$ denote the sequence $(a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m)$, called the *concatenation* of A and B. Given two sets $\mathcal{A}$ and $\mathcal{B}$ of sequences, let $\mathcal{A} \otimes \mathcal{B}$ denote the sets $\{A \otimes B \mid A \in \mathcal{A}, B \in \mathcal{B}\}$ of $|\mathcal{A}| \cdot |\mathcal{B}|$ sequences. Given n sequences indexed as S_i, $i = 1, \ldots, n$, let $\prod_{i=j}^{k} S_i$, $1 \leq j \leq k \leq n$ denote the sequence $S_j \otimes S_{j+1} \otimes \cdots \otimes S_k$. Given n sets of sequences, indexed as $\mathcal{S}_i$, $i = 1, \ldots, n$, let $\prod_{i=j}^{j} \mathcal{S}_i$, $1 \leq j \leq k \leq n$ denote the set $\mathcal{S}_j \otimes \mathcal{S}_{j+1} \otimes \cdots \otimes \mathcal{S}_k$ of $|\mathcal{S}_j| \cdot |\mathcal{S}_{j+1}| \cdots |\mathcal{S}_k|$ sequences.

Given two positive integers n and m, the *uniform sequence* $I(n, m)$ with length n and value m is defined to be the sequence $(m, m, \ldots, m)$ of length n. Let $K = (k_1, k_2, \ldots, k_n)$ be an integer sequence with length $n \geq 1$. We define the *uniform decomposition* of K to be a minimum number of uniform sequences $I_i = I(n_i, m_i)$, $i = 1, 2, \ldots, \ell$ such that $\prod_{i=1}^{\ell} I_i = K$, and define its *length* to be ℓ. From the definition of $\mathcal{R}(n, m)$ and $\mathcal{S}(K)$, we see that $\prod_{i=1}^{\ell} \mathcal{R}(n_i, m_i) = \mathcal{S}(K)$. This can be explained as follows.

For $i \in [1, \ell]$, let $S_i = (s_{i,1}, s_{i,2}, \ldots, s_{i,n_i})$ be a sequence in $\mathcal{R}(n_i, m_i)$ indexed by i. From the definition of $\mathcal{R}(n_i, m_i)$, we have $k_i = m_i > s_{i,1} \geq s_{i,2} \geq \cdots s_{i,n} \geq 0$, and the sequence $S = (s_1, s_2, \ldots, s_n) = \prod_{i=1}^{\ell} S_i$ with length $n = \sum_{i=1}^{\ell} n_i$ belongs to $\mathcal{S}(K)$ since $0 \leq s_j < k_j$, $1 \leq j \leq n$, and $s_j \geq s_{j+1}$ with $k_j = k_{j+1}$, $1 \leq j \leq n-1$. On the other hand, let S be a sequence in $\mathcal{S}(K)$. From the definition of $\mathcal{S}(K)$, for ℓ sequences indexed as $S_i = (s_{i,1}, s_{i,2}, \ldots, s_{i,n_i})$, $1 \leq i \leq \ell$ such that the length of S_i is n_i, $1 \leq i \leq \ell$, and $\prod_{i=1}^{\ell} S_i = S$, we have $0 \leq s_{i,j} < m_i = k_i$, $1 \leq j \leq n_i$ and $s_{i,j} \geq s_{i,j+1}$, $1 \leq j \leq n_i - 1$,

and S_i belongs to $\mathcal{R}(n_i, m_i)$. As a result, the set $\prod_{i=1}^{\ell} \mathcal{R}(n_i, m_i)$ is equivalent to $\mathcal{S}(K)$. Therefore we have the following lemma.

Lemma 6. *Let K be a positive integer sequence K with length $n \geq 1$, and let $I(n_i, m_i)$, $1 \leq i \leq \ell$ denote the uniform decomposition of K with length ℓ. Then it holds that $\mathcal{S}(K) = \prod_{i=1}^{\ell} \mathcal{R}(n_i, m_i)$.*

We introduce a lemma about the concatenation sequences in order to develop algorithms of ranking/unranking for upper bounded sequences. Given two sets of integer sequences $\mathcal{A}$ and $\mathcal{B}$, if all sequences in $\mathcal{A}$ (resp., $\mathcal{B}$) have the common length, then for sequences A, $A' \in \mathcal{A}$ and B, $B' \in \mathcal{B}$, it holds that $A' \otimes B' \prec A \otimes B$ if and only if either (i) $A' \prec A$ or (ii) $A' = A$, $B' \prec B$. Note that this property does not hold without the common length condition. Suppose that $\mathcal{A} = \{(1,2),(1,2,3)\}$ and $\mathcal{B} = \{(1,2),(5,4)\}$. It holds that $(1,2,3) \otimes (1,3) \prec (1,2) \otimes (5,4)$, but neither (i) $(1,2,3) \prec (1,2)$ nor (ii) $(1,2,3) = (1,2),(1,3) \prec (5,4)$ holds. From this observation, for sets of common length sequences, we have the following lemma.

Lemma 7. (i) *Let $\mathcal{A}$ and $\mathcal{B}$ be sets of integer sequences whose length are common for each set, and let A and B be sequences in $\mathcal{A}$ and $\mathcal{B}$, respectively. Then it holds that* $\mathrm{rank}_{\mathcal{A}\otimes\mathcal{B}}(A \otimes B) = (\mathrm{rank}_{\mathcal{A}}(A) - 1) \cdot |\mathcal{B}| + \mathrm{rank}_{\mathcal{B}}(B)$.

(ii) *Let $\ell \geq 1$ be an integer, for each $i \in [1, \ell]$, let $\mathcal{S}_i$ be a set of integer sequences, whose length are common for each set, and let S_i be a sequence in $\mathcal{S}_i$. Then it holds that* $\mathrm{rank}_{\prod_{i=1}^{\ell} \mathcal{S}_i}(\prod_{i=1}^{\ell} S_i) = \sum_{i=1}^{\ell}((\mathrm{rank}_{\mathcal{S}_i}(S_i) - 1) \cdot \prod_{j=i+1}^{\ell} |\mathcal{S}_i|) + 1$.

Proof. (i) Let A and A' (resp., B and B') be sequences in $\mathcal{A}$ (resp., $\mathcal{B}$). Since the length of A and A' (resp., B and B') are the same, from the definition of the lexicographical order, the sequence $A' \otimes B' \prec A \otimes B$ if and only if either $A' \prec A$ or $A' = A$ and $B' \prec B$ holds. The number of sequences $A' \otimes B' \in \mathcal{A} \otimes \mathcal{B}$ such that $A' \prec A$ is $(\mathrm{rank}_{\mathcal{A}}(A) - 1) \cdot |\mathcal{B}|$. In addition, the number of sequences $A' \otimes B' \in \mathcal{A} \otimes \mathcal{B}$ such that $A' = A$ and $B' \prec B$ is $\mathrm{rank}_{\mathcal{B}}(B) - 1$. Hence we have $\mathrm{rank}_{\mathcal{A}\otimes\mathcal{B}}(A \otimes B) = (\mathrm{rank}_{\mathcal{A}}(A) - 1) \cdot |\mathcal{B}| + \mathrm{rank}_{\mathcal{B}}(B)$ since the rank is the number of smaller sequences plus one.

(ii) By Lemma 7-(i), for an integer $j \in [1, \ell]$, we have $\mathrm{rank}_{\prod_{i=j}^{\ell} \mathcal{S}_i}(\prod_{i=j}^{\ell} S_i) = (\mathrm{rank}_{\mathcal{S}_j}(S_j) - 1) \cdot |\prod_{i=j+1}^{\ell} \mathcal{S}_i| + \mathrm{rank}_{\prod_{i=j+1}^{\ell} \mathcal{S}_i}(\prod_{i=j+1}^{\ell} \mathcal{S}_i)$. Hence we have

$$\begin{aligned} &\mathrm{rank}_{\prod_{i=1}^{\ell} \mathcal{S}_i}(\prod_{i=1}^{\ell} S_i) = (\mathrm{rank}_{\mathcal{S}_1}(S_1) - 1) \cdot |\prod_{i=2}^{\ell} \mathcal{S}_i| + \mathrm{rank}_{\prod_{i=2}^{\ell} \mathcal{S}_i}(\prod_{i=2}^{\ell} \mathcal{S}_i) \\ = &\sum_{i=1}^{\ell-1}((\mathrm{rank}_{\mathcal{S}_i}(S_i) - 1) \cdot \prod_{j=i+1}^{\ell} |\mathcal{S}_i|) + \mathrm{rank}_{\mathcal{S}_\ell}(S_\ell) \end{aligned}$$

$$= \sum_{i=1}^{\ell-1}((\mathrm{rank}_{\mathcal{S}_i}(S_i)-1)\cdot\prod_{j=i+1}^{\ell}|\mathcal{S}_i|)+(\mathrm{rank}_{\mathcal{S}_\ell}(S_\ell)-1)\prod_{j=\ell+1}^{\ell}|\mathcal{S}_i|)+1$$
$$= \sum_{i=1}^{\ell}((\mathrm{rank}_{\mathcal{S}_i}(S_i)-1)\cdot\prod_{j=i+1}^{\ell}|\mathcal{S}_i|)+1.$$

□

We devise a ranking algorithm for upper bounded sequences. Based on Lemmas 6 and 7-(ii), given a positive integer sequence K and a sequence $S \in \mathcal{S}(K)$, we can obtain the rank of S in $\mathcal{S}(K)$ by the following lemma.

Lemma 8. *Let K be a positive integer sequence with length $n \geq 1$, and let S be a sequence in $\mathcal{S}(K)$. Let $I_1, I_2, \ldots, I_\ell$ denote the uniform decomposition of K, let n_i (resp., m_i) denote the length (resp., value) of I_i for each $i \in [1,\ell]$, and let S_i denote the subsequence of S such that $|S_i| = n_i$ and $\prod_{i=1}^{\ell} S_i = S$. Then it holds that* $\mathrm{rank}_{\mathcal{S}(K)}(S) = \sum_{i=1}^{\ell}((\mathrm{rank}_{\mathcal{R}(n_i,m_i)}(S_i)-1)\cdot\prod_{j=i+1}^{\ell}|\mathcal{R}(n_j,m_j)|)+1$.

Proof. From Lemma 6, S_i belongs to $\mathcal{R}(n_i,m_i)$ for each $i \in [1,\ell]$. By Lemma 7-(ii), $\mathrm{rank}_{\mathcal{S}(K)}(S)$ is $\sum_{i=1}^{\ell}((\mathrm{rank}_{\mathcal{R}(n_i,m_i)}(S_i)-1)\cdot\prod_{j=i+1}^{\ell}|\mathcal{R}(n_j,m_j)|)+1$. □

Based on Lemma 8, we show an algorithm to obtain the rank of a given upper bounded sequence. Let K be a positive integer sequence, and let S be a sequence in $\mathcal{S}(K)$. First we find the uniform decomposition $I_1, I_2, \ldots, I_\ell$ of K with length ℓ. Let n_i (resp., m_i) denote the length (resp., value) of the I_i for each $i \in [1,\ell]$, and we construct the subsequence S_i of S corresponding to $I_i, 1 \leq i \leq \ell$. If we have the rank of S_i in $\mathcal{R}(n_i,m_i)$ and $\prod_{j=i}^{\ell}|\mathcal{R}(n_j,m_j)|$ for each $i \in [1,\ell]$, then, by Lemma 8, we obtain the rank of S in $\mathcal{S}(K)$. By Lemma 3, the rank of S_i can be obtained in $\mathcal{R}(n_i,m_i)$ in $\mathcal{O}(n_i^2)$ time. Since we have $\prod_{j=i}^{\ell}|\mathcal{R}(n_j,m_j)| = \prod_{j=i+1}^{\ell}|\mathcal{R}(n_j,m_j)|\cdot\binom{m_i+n_i-1}{n_i}$, for each $i \in [1,\ell]$, $\prod_{j=i+1}^{\ell}|\mathcal{R}(n_j,m_j)|$ can be obtained in $\mathcal{O}(n_i)$ time. We show a description of the algorithm in Algorithm 3.

Theorem 3. *Let K be a positive integer sequence with length $n \geq 1$, let S be a sequence in $\mathcal{S}(K)$, let $I_1, I_2, \ldots, I_\ell$ denote the uniform decomposition of K with length ℓ, and let n_i (resp., m_i) denote the length (resp., value) of I_i for each $i \in [1,\ell]$. Then* $\mathrm{rank}_{\mathcal{S}(K)}(S)$ *can be obtained in $\mathcal{O}(\sum_{i=1}^{\ell} n_i^2)$ time.*

Proof. We divide S into $S_i, 1 \leq i \leq \ell$ such that $|S_i| = n_i$ and $\prod_{i=1}^{\ell} S_i = S$. From Lemma 3, for each $i \in [1,\ell]$, $\mathrm{rank}_{\mathcal{R}(n_i,m_i)}(S_i)$ can be obtained in $\mathcal{O}(n_i^2)$ time. Given the value of $\prod_{j=i+1}^{\ell}|\mathcal{R}(n_j,m_j)|$, the value of $\prod_{j=i}^{\ell}|\mathcal{R}(n_j,m_j)|$ can be obtained in $\mathcal{O}(n_i)$ time. Hence the calculation of $\sum_{i=1}^{\ell}((\mathrm{rank}_{\mathcal{R}(n_i,m_i)}(S_i)-1)\cdot\prod_{j=i+1}^{\ell}|\mathcal{R}(n_j,m_j)|)+1$ of Lemma 8 can be done in $\mathcal{O}(\sum_{i=1}^{\ell}(n_i+n_i^2)) = \mathcal{O}(\sum_{i=1}^{\ell} n_i^2)$ time. Therefore the rank of S in $\mathcal{S}(K)$ can be obtained in $\mathcal{O}(\sum_{i=1}^{\ell} n_i^2)$ time. □

Algorithm 3 Ranking for Upper Bounded Sequences

Input: A positive integer sequence K, a sequence S in $\mathcal{S}(K)$.

Output: $\mathrm{rank}_{\mathcal{S}(K)}(S)$.

1: $r := 1; x := 1$;
2: $(I_1, I_2, \ldots, I_\ell) :=$ the uniform decomposition of K;
3: $n_i :=$ the length of I_i for each $i \in [1, \ell]$;
4: $m_i :=$ the value of I_i for each $i \in [1, \ell]$;
5: $S_i :=$ the subsequence of S corresponding to I_i for each $i \in [1, \ell]$;
6: **for** $i := \ell, \ell - 1, \ldots, 1$ **do**
7: $\quad r := r + (\mathrm{rank}_{\mathcal{R}(n_i, m_i)}(S_i) - 1) \cdot x$
8: $\quad x := x \cdot \binom{m_i + n_i - 1}{n_i}$
9: **end for**;
10: **output** r

Next we discuss the unranking problem for upper bounded sequences. From Lemmas 6 and 7-(ii), we have the following lemma.

Lemma 9. *Let K be a positive integer sequence with length $n \geq 1$, let $I_1, I_2, \ldots, I_\ell$ denote the uniform decomposition of K, and let n_i (resp., m_i) denote the length (resp., value) of I_i for each $i \in [1, \ell]$. Let k be an integer in $[1, |\mathcal{S}(K)|]$, for $1 \leq i \leq \ell$, let k_i and k'_i denote integers such that $k_0 = k - 1$ and $k'_i \prod_{j=i}^{\ell} |\mathcal{R}(n_j, m_j)| + k_i = k_{i-1}$ with $k_i < \prod_{j=i}^{\ell} |\mathcal{R}(n_j, m_j)|$ for $i \in [1, \ell]$. Let S_i indexed by i denote the $(k'_i + 1)$-th sequence in $\mathcal{R}(n_i, m_i)$ for each $i \in [1, \ell]$. Then it holds that the k-th sequence in $\mathcal{S}(K)$ in the lexicographical ascending order is $\prod_{i=1}^{\ell} S_i$.*

Proof. From Lemma 6, $\mathcal{S}(K)$ is equivalent to $\prod_{i=1}^{\ell} \mathcal{R}(n_i, m_i)$. Hence $\prod_{i=1}^{\ell} S_i$ belongs to $\mathcal{S}(K)$. In addition, from Lemma 7-(ii), we have

$$\begin{aligned}
\mathrm{rank}_{\mathcal{S}(K)}(\prod_{i=1}^{\ell} S_i) &= \sum_{i=1}^{\ell}((\mathrm{rank}_{\mathcal{R}(n_i,m_i)}(S_i) - 1) \cdot \prod_{j=i+1}^{\ell} |\mathcal{R}(n_j, m_j)|) + 1. \\
= \sum_{i=1}^{\ell}(k'_i \prod_{j=i+1}^{\ell} |\mathcal{R}(n_j, m_j)|) + 1 &= \sum_{i=1}^{\ell-1}(k'_i \prod_{j=i+1}^{\ell} |\mathcal{R}(n_j, m_j)|) + k_\ell + 1 \\
= \sum_{i=1}^{\ell-2}(k'_i \prod_{j=i+1}^{\ell} |\mathcal{R}(n_j, m_j)|) + k'_{\ell-1} \prod_{j=\ell}^{\ell} |\mathcal{R}(n_j, m_j)| &+ k_\ell + 1 \\
= \sum_{i=1}^{\ell-2}(k'_i \prod_{j=i+1}^{\ell} |\mathcal{R}(n_j, m_j)|) + k_{\ell-1} + 1 &= \cdots = k_0 + 1 = k.
\end{aligned}$$

□

Based on Lemma 9, we propose an algorithm to generate the k-th sequence in $\mathcal{S}(K)$ in the lexicographical ascending order. Let K be a positive integer sequence, and let k be an integer in $[1, |\mathcal{S}(K)|]$. First we find the uniform decomposition $I_1, I_2, \ldots, I_\ell$ of K. We assume that the value and length of I_i is n_i and m_i for each $i \in [1, \ell]$. We calculate $\prod_{j=i}^{\ell} |\mathcal{R}(n_j, m_j)|$ for each $i \in [1, \ell]$, and we construct the integers k_i and k'_i in Lemma 9. Then we generate the $(k'_i + 1)$-th sequence S_i in $\mathcal{R}(n_j, m_j)$ for each $i \in [1, \ell]$, and we construct the concatenation sequence $\prod_{i=1}^{\ell} S_i$. A description of the algorithm is shown in Algorithm 4. In this algorithm, the variable $x[i]$ stores the value $\prod_{j=i}^{\ell} |\mathcal{R}(n_j, m_j)|$ for each $i \in [1, \ell]$.

Algorithm 4 Unranking for Upper Bounded Sequences

Input: A positive integer sequence K and an integer $k \in [1, |\mathcal{S}(K)|]$.

Output: The k-th sequence in $\mathcal{S}(K)$ following the lexicographical ascending order.

1: $(I_1, I_2, \ldots, I_\ell) :=$ the uniform decomposition of K;
2: $x[\ell + 1] := 1$;
3: **for each** $i := \ell, \ell - 1, \ldots, 2$ **do**
4: $\quad n_i :=$ the length of I_i;
5: $\quad m_i :=$ the value of I_i;
6: $\quad x[i] := x[i + 1] \cdot \binom{m_i + n_i - 1}{n_i}$;
7: **end for**;
8: $k^* := k - 1$;
9: **for** $i := 1, 2, \ldots, \ell$ **do**
10: $\quad k' := \lfloor (k^* - 1)/x[i + 1] \rfloor$;
11: $\quad S_i :=$ $(k' + 1)$-th sequence of $\mathcal{R}(n_i, m_i)$;
12: $\quad k^* := k^* - k' \cdot x[i + 1]$
13: **end for**;
14: **output** $\prod_{i=1}^{\ell} S_i$

Theorem 4. *Let K be a positive integer sequence with length $n \geq 1$, let k be an integer in $[1, |\mathcal{S}(K)|]$, let $I_1, I_2, \ldots, I_\ell$ denote the uniform decomposition of K, and let n_i (resp., m_i) denote the length (resp., value) of I_i for each $i \in [1, \ell]$. Then the k-th sequence in $\mathcal{S}(K)$ following the lexicographical ascending order can be obtained in $\mathcal{O}(\sum_{i=1}^{\ell} n_i^2 \log(n_i + m_i))$ time.*

Proof. Given the value $\prod_{j=i+1}^{\ell} |\mathcal{R}(n_j, m_j)|$, the value $\prod_{j=i}^{\ell} |\mathcal{R}(n_j, m_j)|$ can be obtained in $\mathcal{O}(n_i)$ time. Therefore we obtain all k'_i in Lemma 9 in $\mathcal{O}(\sum_{i=1}^{\ell} n_i)$ time. From Lemma 3, for each $i \in [1, \ell]$, given an integer $k_i \in [1, |\mathcal{R}(n_i, m_i)|]$, the k_i-th sequence in $\mathcal{R}(n_i, m_i)$ following the lexicographical ascending order in $\mathcal{O}(n_i^2 \log(n_i + m_i))$ time. Hence by Lemma 9, the k-th sequence in $\mathcal{S}(K)$ can be constructed in $\mathcal{O}(\sum_{i=1}^{\ell} n_i^2 \log(n_i + m_i))$ time. □

2.3 Positive Descending Sequence

For an integer sequence $S = (s_1, s_2, \ldots, s_n)$, we say that S is a *positive descending sequence* with length n if it holds that $s_1 \geq s_2 \geq \cdots \geq s_n > 0$. Let $\mathcal{D}(n, d)$ denote the set of all positive descending sequences such that any sequence $S \in \mathcal{D}(n, d)$ satisfies $|S| \leq n$ and $\mathrm{sum}(S) = d$, and we define $\mathcal{D}(n, d, m) \triangleq \{S \in \mathcal{D}(n, d) \mid \max(S) = m\}$. Given a sequence $S \in \mathcal{D}(n, d)$, we examine how to generate the lexicographic predecessor of S in $\mathcal{D}(n, d)$. In order to obtain the lexicographic predecessor of a given sequence, we introduce a useful lemma.

Lemma 10. *Let $n \geq 1$ and $d \geq n$ be two positive integers, and let $S = (s_1, s_2, \ldots, s_\ell)$ be a sequence with the length ℓ in $\mathcal{D}(n, d)$.*

(i) *Let m be an integer with $d/n \leq m \leq d$. Then a sequence S is lexicographically maximum in $\mathcal{D}(n, d, m)$ if and only if $\ell = \lceil d/m \rceil$, $s_1 = s_2 = \cdots = s_{\ell-1} = m$, and $s_\ell = d - (\ell - 1)m$.*

(ii) *A sequence S is lexicographically minimum in $\mathcal{D}(n, d)$ if and only if $\ell = n$ and $s_1 - s_n \leq 1$, i.e., for $q = d - n\lfloor d/n \rfloor$, $s_1 = s_2 = \cdots = s_q = \lceil d/n \rceil$ and $s_{q+1} = s_{q+2} = \cdots = s_n = \lfloor d/n \rfloor$.*

(iii) *Assume that $\ell < n$ or $s_1 - s_\ell \geq 2$. Let k denote the largest index such that $s_k \geq 2$ if $\ell < n$, otherwise $s_k - s_n \geq 2$. Then the lexicographic predecessor of S in $\mathcal{D}(n, d)$ is given by the concatenation of the sequences $(s_1, s_2, \ldots, s_{k-1})$ and $\max(\mathcal{D}(n - k + 1, \sum_{k \leq i \leq n} s_i, s_k - 1))$.*

Proof. (i) We show that such a sequence S is the lexicographically maximum in $\mathcal{D}(n, d, m)$. Since S is a descending sequence, and it holds that $\mathrm{sum}(S) = d$ and $\max(S) = m$, such a sequence S is in $\mathcal{D}(n, d, m)$. Assume that there exists a sequence $X = (x_1, x_2, \ldots, x_{\ell'})$ in $\mathcal{D}(n, d, m)$ such that $S \prec X$ holds. Suppose that $\ell' > \ell$ and $x_i = s_i$ for each $i \in [1, \ell]$. From the definition of the positive descending sequence, we have $\mathrm{sum}(X) > \sum_{i=1}^{\ell} x_i = \mathrm{sum}(S) = d$. This contradicts that $\mathrm{sum}(X) = d$, and we see that there exists an integer $k \leq \min\{\ell, \ell'\}$ such that $x_k > s_k$ and $x_i = s_i$ for all $1 \leq i < k$. If $s_k = m$ holds, then $x_k > m$ must hold. However, this contradicts that $\max(X) = m$, and we have $s_k \neq m$. If $k = \ell$ hold, then from the assumption, we have $x_k > s_k$. This, however, contradicts that $\mathrm{sum}(X) = d$, and there exists no integer k such that $x_k > s_k$ and $x_i = s_i$ for all $1 \leq i < k$, and there exists no sequence X such that $S \prec X$ in $\mathcal{D}(n, d, m)$. Therefore we conclude that such a sequence S is the lexicographically maximum in $\mathcal{D}(n, d, m)$.

(ii) First we show that a sequence $S = (s_1, s_2, \ldots, s_n)$ in $\mathcal{D}(n, d)$ such that $s_1 - s_n \leq 1$ is uniquely determined. Suppose that the first k elements of S are s and the others are $s - 1$. Since $\mathrm{sum}(S) = ks + (n - k)(s - 1) = d = n\lfloor d/n \rfloor + q$ holds, we obtain

$s - \lfloor d/n \rfloor - 1 = (q - s)/n$. Since s and $\lfloor d/n \rfloor$ are integer, $(q - s)/n$ must be integer. However, we have $0 \leq q < n$ and $1 \leq s \leq n$. If $q = 0$ holds, then we obtain $k = n$ and $s = d/n = \lceil d/n \rceil$. Otherwise, $k = q$ and $s = \lfloor d/n \rfloor + 1 = \lceil d/n \rceil$ holds. Therefore such a sequence S is obtained.

Suppose that there exists a sequence $X = (x_1, x_2, \ldots, x_{\ell'})$ in $\mathcal{D}(n, d)$ such that $X \prec S$ holds. There exists an integer k such that $x_k < s_k$ and $x_i = s_i$ for all $1 \leq i < k$. In order to satisfy $\mathrm{sum}(X) = d$, there exists an element $x_j > s_j$ for an integer $j \in [k + 1, \ell']$. However, we obtain $x_k < x_j$ since it holds that $x_k \leq s_k - 1 < s_j + 1 \leq x_j$. This contradicts that X is a descending sequence, and we conclude that S is the lexicographically minimum sequence in $\mathcal{D}(n, d)$.

(iii) Let S' be the sequence obtained by concatenation of $(s_1, s_2, \ldots, s_{k-1})$ and $\max(\mathcal{D}(n - k + 1, \sum_{k \leq i \leq n} s_i, s_k - 1))$. Clearly, $S' \in \mathcal{D}(n, d)$ and $S' \prec S$ holds. Assume that there exists a sequence $X = (x_1, x_2, \ldots, x_{\ell'}) \in \mathcal{D}(n, d, m)$ such that $S' \prec X \prec S$ holds. For each $1 \leq i < k$, we have $s'_i = x_i = s_i$. In addition, $x_k = s_k$ or $x_k = s_k - 1$ holds. Let S_k, S'_k and X_k denote the subsequence from k-th to the end of the sequence S, S', and X, respectively.

Suppose that $x_k = s_k$. The subsequence X_k must be in $\mathcal{D}(n - k + 1, \sum_{k \leq i \leq n} s_i, s_k)$. Since S_k is the lexicographically minimum in $\mathcal{D}(n - k + 1, \sum_{k \leq i \leq n} s_i, s_k)$, this, however, contradicts that $X \prec S$, and we have $x_k \neq s_k$.

Next suppose that $x_{k-1} = s_{k-1} - 1$. The subsequence X_k must be in $\mathcal{D}(n - k + 1, \sum_{k \leq i \leq n} s_i, s_k - 1)$. Since S'_k is the lexicographically maximum in $\mathcal{D}(n - k + 1, \sum_{k \leq i \leq n} s_i, s_k - 1)$, this, however, contradicts that $S' \prec X$, and we have $x_{k-1} \neq s_{k-1} - 1$.

As a result, there exists no sequence $X \in \mathcal{D}(n, d)$ such that $S' \prec X \prec S$ holds, and S' is the predecessor of S in $\mathcal{D}(n, d)$. □

Based on Lemma 10, in Algorithm 5, we show an algorithm to find the lexicographic predecessor of a given sequence in $\mathcal{D}(n, d)$, if one exists. Let S be a given sequence in $\mathcal{D}(n, d)$. In this algorithm, first, we check whether $S = \min(\mathcal{D}(n, d))$ or not by Lemma 10-(ii). If $S \neq \min(\mathcal{D}(n, d))$, then, by Lemma 10-(i) and (iii), we construct the lexicographic predecessor of S in $\mathcal{D}(n, d)$.

Theorem 5. *Let $n \geq 1$ and $d \geq n$ be two integers, and let S be a positive descending sequence in $\mathcal{D}(n, d)$. Then testing whether S is lexicographically minimum in $\mathcal{D}(n, d)$ or not can be done in $\mathcal{O}(1)$ time, and the lexicographic predecessor of S in $\mathcal{D}(n, d)$ if one exists can be obtained in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space.*

Proof. From Lemma 10-(ii), in order to determine $S = \min(\mathcal{D}(n, d))$, we only examine whether both $\ell = n$ and $s_1 - s_\ell \leq 1$ hold or not. This can be done in $\mathcal{O}(1)$ time.

The index k of Lemma 10-(iii) can be found in $\mathcal{O}(n)$ time. In addition, the lexicographically maximum sequence in $\mathcal{D}(n, d)$ can be obtained in $\mathcal{O}(n)$ time by Lemma 10-

Algorithm 5 Generating the predecessor of $S \in \mathcal{D}(n,d)$

Input: Two positive integer $n \geq 1$ and $d \geq n$, and a positive descending sequence $S = (s_1, s_2, \ldots, s_\ell) \in \mathcal{D}(n,d)$.

Output: If S has a lexicographic predecessor in $\mathcal{D}(n,d)$, then the lexicographic predecessor of S; otherwise a message, "S is minimum."

1: **if** $\ell < n$ **then**
2: $k :=$ the largest index such that $s_k \geq 2$;
3: $(s'_k, s'_{k+1}, \ldots, s'_{\ell'}) := \max(\mathcal{D}(n-k+1, \sum_{k \leq i \leq n} s_i, s_k - 1))$;
4: **output** $(s_1, s_2, \ldots, s_{k-1}, s'_k, \ldots, s'_{\ell'})$
5: **else if** $\ell = n$, $s_1 - s_\ell \geq 2$ **then**
6: $k :=$ the largest index such that $s_k - s_\ell \geq 2$;
7: $(s'_k, s'_{k+1}, \ldots, s'_{\ell'}) := \max(\mathcal{D}(n-k+1, \sum_{k \leq i \leq n} s_i, s_k - 1))$;
8: **output** $(s_1, s_2, \ldots, s_{k-1}, s'_k, \ldots, s'_{\ell'})$
9: **else**
10: **output** message "S is minimum"
11: **end if**

(i). Therefore the time complexity to generate the lexicographic predecessor of a positive descending sequence in $\mathcal{D}(n,d)$ is $\mathcal{O}(n)$. The space complexity is $\mathcal{O}(n)$ because we only store the values of one sequence with length n. □

3 Preliminaries

In this section, we introduce the definitions and notions of graphs that are used further, including a dynamic programming based counting algorithm for degree-bounded rooted trees. In subsection 3.2, we propose a vertex ordering for ordered trees, and we introduce two mappings: one is a mapping from ordered trees to sequences based on the vertex ordering; the other is a mapping from degree-bounded rooted trees to ordered trees. After that, we define a order for degree-bounded rooted trees to be the lexicographic ascending order of corresponding sequence. In subsection 3.3, we design an dynamic programming based algorithm for counting degree-bounded rooted trees.

3.1 Graphs

A *graph* is defined to be an ordered pair of a finite set of vertices and a finite set of edges. In this paper, an edge is represented as an unordered pair of distinct vertices. Let G be a graph. The vertex set of G is denoted by $V(G)$, and the edge set of G is denoted by $E(G)$. An edge $e \in E(G)$ incident with vertices v_i and v_j is denoted by $e = v_iv_j$. The *degree* $\deg(v)$ of a vertex $v \in V(G)$ is defined to be the number of edges incident to v in G. The maximum degree of G is defined to be the largest degree in G. For a vertex r in G, let (G, r) denote the graph G rooted at the vertex r.

For two graphs G and G', we say that G and G' are *isomorphic* if there exists a bijection $\phi : V(G) \to V(G')$ with for all vertex pairs u and v of G, $uv \in E(G)$ if and only if $\phi(u)\phi(v) \in E(G')$. Such a bijection ϕ is called an *isomorphism* from G to G'. We also introduce an isomorphism between rooted graphs. For two rooted graphs (G, r) and (G', r'), we say that G and G' are *rooted isomorphic* if there exists an isomorphism ϕ from G to G' such that $\phi(r) = r'$.

Let us call a *labeled graph* G with n vertices a graph with a basic label if $V(G) = \{1, 2, \ldots, n\}$ and each edge is given as a pair of i and j for the end-vertices i and j of the edge. For a labeled graph G and a vertex $v \in V(G)$, let $\mathrm{lab}(v)$ denote the label of the vertex v.

A *path* of length k is defined to be a non-empty graph G such that $V(G) = \{v_1, v_2, \ldots, v_k\}$ and $E(G) = \{v_iv_{i+1} \mid i = 1, 2, \ldots, k-1\}$. A *cycle* is defined to be a graph obtained by adding the edge v_1v_k to a path of length k. A graph G is called *connected* if there exists a path from u to v for any two vertices $u, v \in V(G)$.

A *tree* (*unrooted tree*) is defined to be an acyclic connected graph. For a tree T and any two vertices $u, v \in V(T)$, let $P_T(u, v)$ denote the unique path from u to v in T. A tree $T = (G, r)$ with a fixed root $r \in V(G)$ is called a *rooted tree*. From Jordan's theorem [8], any unrooted tree T on n vertices has either such a vertex or an edge, the removal of which leaves no connected component with more than $\lfloor (n-1)/2 \rfloor$ or $n/2$ vertices respectively. Such a vertex is called a *unicentroid*, and such an edge a

bicentroid. Hence we can always treat any unrooted tree as a rooted tree.

Let $T = (G, r)$ be a rooted tree. The *parent* of $v \in V(T) - \{r\}$ is defined to be the vertex adjacent to v in $P_T(r, v)$, and the *ancestors* of $v \in V(T) - \{r\}$ are defined to be the vertices in $P_T(r, v)$. Note that the parent and the ancestors of the root r are not defined. For two vertices u and v in T, if v is the parent of u, then we call u a *child* of v, and if v is an ancestor of u, then we call u a *descendant* of v. If u and v are children of the same vertex, then we say that u and v are *siblings*. For a vertex $v \in V(T)$, the *subtree* $T(v)$ is defined to be the tree rooted at the vertex v formed by v and all its descendants in T. For a child v of the root in T, we call the subtree $T(v)$ a *root-subtree* in T.

3.2 Ordering of Trees

An *ordered tree* $T = (G, r, \pi)$ is defined to be a rooted tree (G, r) with a left-to-right ordering π specified for the children of each vertex. Let $T = (G, r, \pi)$ be an ordered tree. For a vertex $v \in V(T)$, let $T(v)$ denote the ordered subtree of T that consists of v and all descendants of v, preserving the order for the children of each vertex. For an integer $i \in [1, \deg(r)]$, let T_i denote the i-th root subtree of T following the left-to-right ordering π.

For an ordered tree, we mention three vertex orderings, *Depth First Search Pre-order* (DFS Pre-order), *Breadth First Search Pre-order* (BFS Pre-order), and *Siblings First Depth First Search* (Siblings First DFS). The DFS Pre-order starts from the root and visits vertices from the left to the right, and we give the index of vertices following this order. The BFS Pre-order starts from the root and visits vertices from closest to the root following the left-to-right ordering. We say that the reverse order of the DFS Pre-order (resp., BFS Pre-order) is the *DFS Post-order* (resp., *BFS Post-order*). In this paper, in order to utilize the counting information about degree-bounded rooted trees, we propose a vertex ordering for ordered trees. Given an ordered tree, we traverse the ordered tree by the depth first search that starts from the root and visits vertices from the left to the right, but we give an index for each of the children of the current vertex in left-to-right order before visiting next vertex. We call such a vertex labelling the *Siblings First DFS*.

Figure 2 shows an example of the DFS-Post order, BFS-Post order, and Siblings First DFS for an ordered tree. Figure 2 (a) shows an ordered tree $T = (G, r, \pi)$. The arrows shows the left-to-right ordering π for children of T. In Figure 2 (b), the vertices of T are indexed by the DFS Pre-order. In Figure 2 (c), the vertices of T are indexed by the BFS Pre-order. In Figure 2 (d), the vertices of T are indexed by the Siblings First DFS.

Let $T = (G, r, \pi)$ be an ordered tree with n vertices. For the vertices v_i, $1 \leq i \leq n$ indexed by the Siblings First DFS ordering of T, we define the *descendant representation*

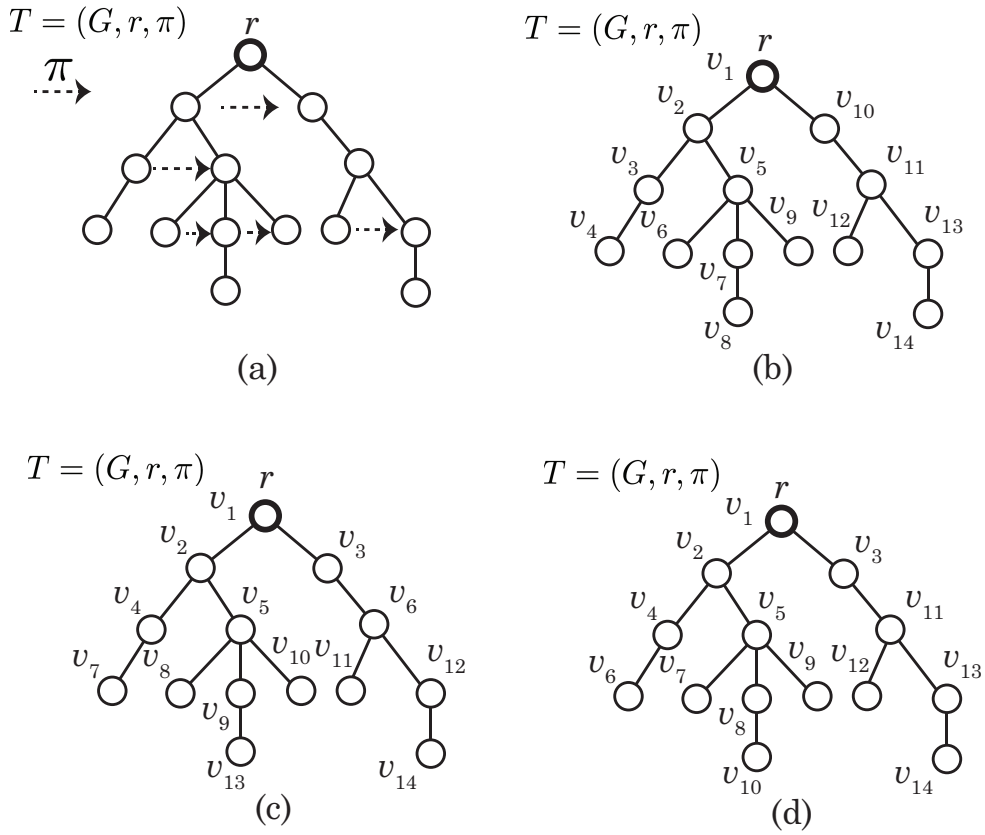

Figure 2: An example of indexing for an ordered tree: (a) An ordered tree; (b) DFS Pre-order; (c) BFS Pre-order; and (d) Siblings First DFS.

$L_D(T)$ of T to be the sequence

$$L_D(T) \triangleq (|V(T(v_1))|, |V(T(v_2))|, \ldots, |V(T(v_n))|).$$

Lemma 11. *Let D be the descendant representation of an ordered tree with n vertices. Then the structure of the ordered tree can be constructed from its descendant representation D in $\mathcal{O}(n)$ time.*

Proof. Let $(d_1, d_2, \ldots, d_n)$ be the descendant representation of an ordered tree $T = (G, r, \pi)$. The first entry d_1 of the descendant representation shows the number of vertices in the ordered tree T, and for an integer $i \in [2, n]$ such that $\sum_{j=2}^{i} d_j = n - 1$, the number $i-1$ shows the degree of the root, and for each $j \in [1, i-1]$, we see that the number of j-th root-subtree in the left-to-right ordering contains d_{j+1} vertices. Note that such an integer i uniquely exists since for $j \in [2, \deg(r) + 1]$, d_j is the number of vertices in a root-subtree, and $\sum_{j=2}^{\deg(r)+1} d_j$ is the number of vertices in T except for the root, that is $n - 1$. Hence we obtain the degree of the root of T in $\mathcal{O}(\deg(r))$ time.

The subsequence obtained from (2+deg(r))-th entry to (2+deg(r)+d_2-1)-th entries of the descendant sequence show the number of descendant of vertices in the first root-subtree of T. We see that the concatenation of d_2 and $(d_{2+\deg(r)}, d_{2+\deg(r)+1}, \ldots, d_{2+\deg(r)+d_2-1})$ is the descendant representation of the first root-subtree T_1 of T. Similarly, the next d_3-1 entries show the number of descendant of vertices in the 2nd root-subtree T_2 of T, and the concatenation of d_3 and the next d_3-1 entries is the descendant representation of the 2nd root-subtree T_2 of T.

In the same manner, the descendant representation of each root-subtree can be obtained. We apply similar identification for each root-subtree recursively, and the structure of the ordered tree T can be obtained. Identification of the descendant representation of the root-subtrees can be done in $\mathcal{O}(\deg(r))$ time. The time complexity to construct the structure of ordered tree T recursively is $\mathcal{O}(\sum_{v \in V(T)} \deg(v))$. Since we have $\sum_{v \in V(T)} \deg(v) = 2|E(T)| = 2(n-1)$, the structure of the ordered tree can be constructed in $\mathcal{O}(n)$ time. □

Figure 3 shows an example of construction for an ordered tree from its descendant representation. Figure 3 (a) shows an ordered tree $T = (G, r, \pi)$ and its descendant representation $L_D(T)$. Figure 3 (b) shows the first step to construct the structure of T from $L_D(T)$. Since the first entry of $L_D(T)$ is 12, we see that T has 12 vertices. Next we identify the degree of the root r. Since the sum of second and third entry of $L_D(T)$ are $7 + 4 = 11 = 12 - 1$, we see that the degree of the root is 2. In addition, since the subsequence from 4-th to 9-th entries of $L_D(T)$ corresponding the vertices in $V(T_1)$, we have $L_D(T_1) = (7, 2, 4, 1, 1, 1, 1)$. Similarly, we have $L_D(T_2) = (4, 3, 1, 1)$. In the same manner, we construct the structure of each root-subtree from its descendant representation. Figure 3 (c) show the next step to construct the structure of T, and Figure 3 (d) show the final step to construct the structure of T.

Let $T = (G, r)$ be a rooted tree. There may be more than one ordered tree $T' = (G', r', \pi')$ that are rooted isomorphic to T. An ordered tree $T^* = (G^*, r^*, \pi^*)$ is called the *canonical tree* of T if G^* is a labeled graph satisfying the following two conditions:

(i) for the i-th vertex v_i following the Siblings First DFS in T^*, $\mathrm{lab}(v_i) = i$; and

(ii) the descendant representation $L_D(T^*)$ is the lexicographically maximum among all ordered trees $T' = (G', r', \pi')$ that are rooted isomorphic to T.

For a rooted tree $T = (G, r)$, we define the *canonical representation* $L_D(T)$ to be the descendant representation of the canonical tree of T. Note that two rooted trees $T = (G, r)$ and $T' = (G', r')$ are rooted isomorphic if and only if $L_D(T) = L_D(T')$.

Figure 4 (a) shows an ordered tree $T = (G, r, \pi)$ indexed by Siblings First DFS and its descendant representation $L_D(T)$, and Figure 4 (b) shows the canonical tree T^* of the rooted tree (G, r) and its descendant representation $L_D(T^*)$.

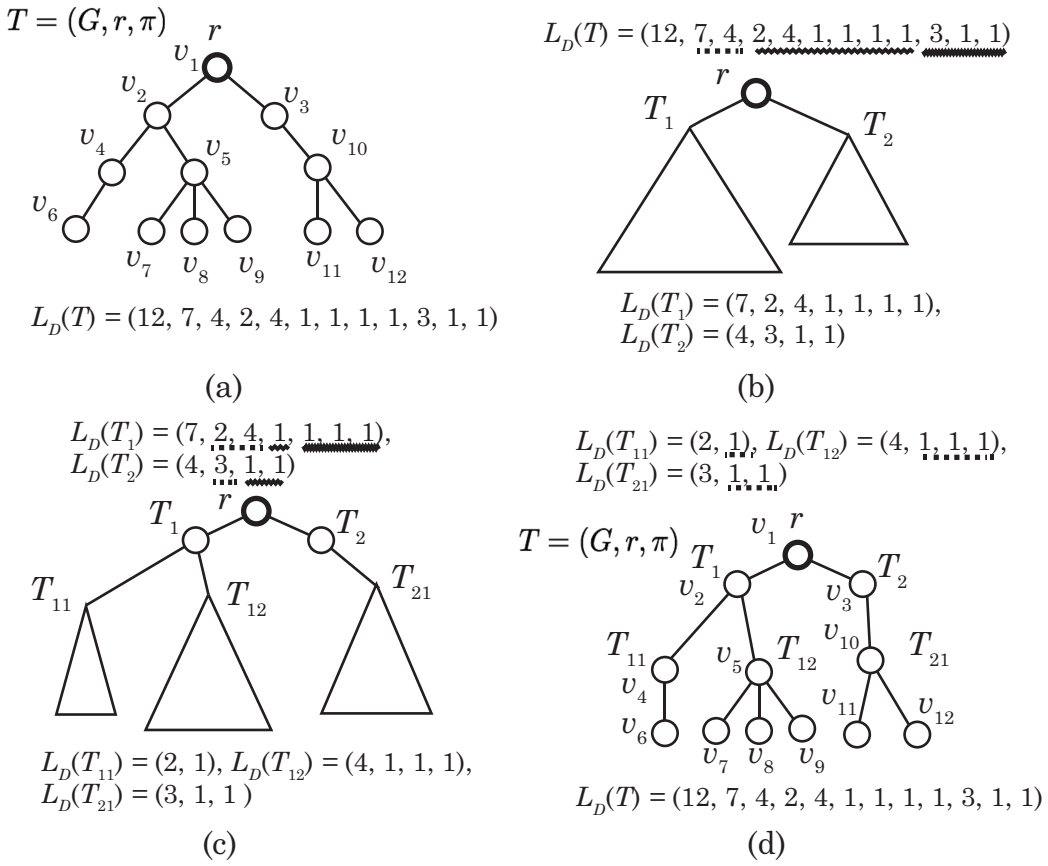

Figure 3: An example of constructing the structure of an ordered tree from its descendant representation: (a) An ordered tree T and its descendant representation; (b) First step of constructing the structure of T; (c) Second step of constructing the structure of T; and (d) Final step of constructing the structure of T.

Now we introduce a set of degree-bounded rooted trees and we define the order of trees in the set. For two positive integers $n \geq 1$ and $\Delta \geq 2$, we define the set $\mathcal{G}_\Delta(n)$ of degree-bounded rooted trees to be the set of canonical trees of all rooted trees with n vertices satisfying:

(i) the degree of the root is at most $\Delta - 1$;

(ii) the degree of vertex except for the root is at most Δ; and

(iii) no two different canonical trees in $\mathcal{G}_\Delta(n)$ are rooted isomorphic.

We let $g_\Delta(n) \triangleq |\mathcal{G}_\Delta(n)|$ for $n \geq 1$ and $\Delta \geq 2$. Note that, if the maximum degree of a tree is at most 1, such a tree is either a single vertex or two vertices with one edge.

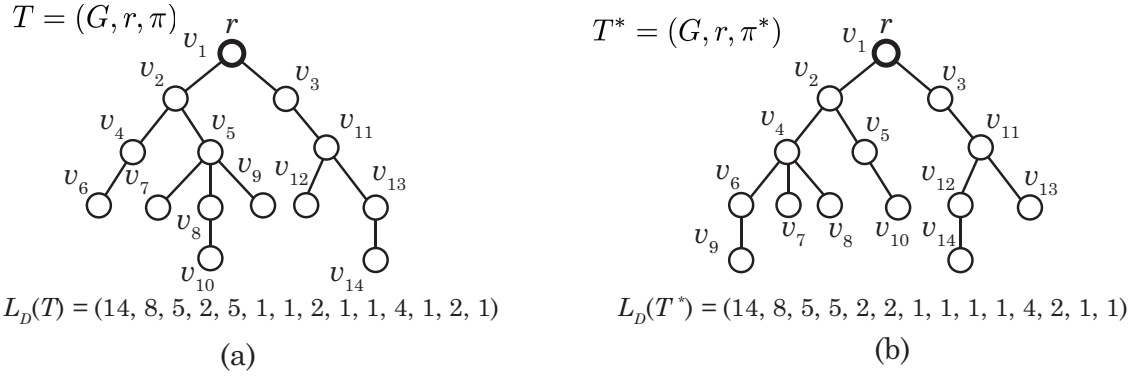

Figure 4: (a) An ordered tree $T = (G, r, \pi)$ indexed by the Siblings First DFS; and (b) The canonical tree $T^* = (G^*, r^*, \pi^*)$ of (G, r) indexed by the Siblings First DFS.

Hence we assume that $n \geq 1$ and $\Delta \geq 2$.

For two positive integers $n \geq 1$ and $\Delta \geq 2$, the tree ordering $\sigma_{\rm des}$ in $\mathcal{G}_\Delta(n)$ is defined to be the lexicographical ascending order of its canonical representation, i.e., for two canonical trees $T = (G, r, \pi)$ and $T' = (G', r', \pi')$ in $\mathcal{G}_\Delta(n)$, $\sigma_{\rm des}(T) \prec \sigma_{\rm des}(T')$ if $\sigma_{\rm lex}(L_D(T)) \prec \sigma_{\rm lex}(L_D(T'))$ holds. For two positive integers $n \geq 1$ and $\Delta \geq 2$, let ${\rm rank}_{\Delta,n} : \mathcal{G}_\Delta(n) \to [1, g_\Delta(n)|]$ denote the function such that for two canonical trees T and T' in $\mathcal{G}_\Delta(n)$, ${\rm rank}_{\Delta,n}(T) < {\rm rank}_{\Delta,n}(T')$ if $\sigma_{\rm des}(T) \prec \sigma_{\rm des}(T')$ holds. We say that a canonical tree $T \in \mathcal{G}_\Delta(n)$ with ${\rm rank}_{\Delta,n}(T) = k$ is the k-th tree in $\mathcal{G}_\Delta(n)$, or the rank of such a canoincal tree T is k in $\mathcal{G}_\Delta(n)$. For a canonical tree $T \in \mathcal{G}_\Delta(n)$ with ${\rm rank}_{\Delta,n}(T) = k > 1$, we say that a canonical tree $T' \in \mathcal{G}_\Delta(n)$ is the predecessor tree of T if ${\rm rank}_{\Delta,n}(T') = k - 1$. It immediately follows that T and T' in $\mathcal{G}_\Delta(n)$ are rooted isomorphic if and only if ${\rm rank}_{\Delta,n}(T) = {\rm rank}_{\Delta,n}(T')$.

For two integers $n \geq 1$ and $\Delta \geq 2$, let T be a canonical tree in $\mathcal{G}_\Delta(n)$. We know that for each $v \in V(T)$, the subtree $T(v)$ is in $\mathcal{G}_\Delta(|V(T(v))|)$. Hence for an integer $\Delta \geq 2$, we define the function ${\rm rank}_\Delta$ to be ${\rm rank}_\Delta(T) \triangleq {\rm rank}_{\Delta,|V(T)|}(T)$. For a canonical tree $T \in \mathcal{G}_\Delta(n)$, we define the *descendant sequence* $DS(T)$ and *rank sequence* $RS(T)$ as follows:

(i) $DS(T) \triangleq (|V(T_1)|, |V(T_2)|, \ldots, |V(T_{\deg(r)})|)$; and

(ii) $RS(T) \triangleq ({\rm rank}_\Delta(T_1), {\rm rank}_\Delta(T_2), \ldots, {\rm rank}_\Delta(T_{\deg(r)}))$.

We show an example of the descendant sequence and the rank sequence of a canonical tree in Figure 5. Figure 5 (a) (resp., (b)) shows the canonical trees in $\mathcal{G}_4(3)$ (resp., $\mathcal{G}_4(4)$) following the tree ordering σ_{des}. Figure 5 (c) shows a rooted tree $T = (G, r)$ with 12 vertices whose maximum degree at most 4, and Figure 5 (d) shows the canonical tree T^* of T. From Figure 5 (b), we see that the first root-subtree T_1^* of T^* has 4 vertices and

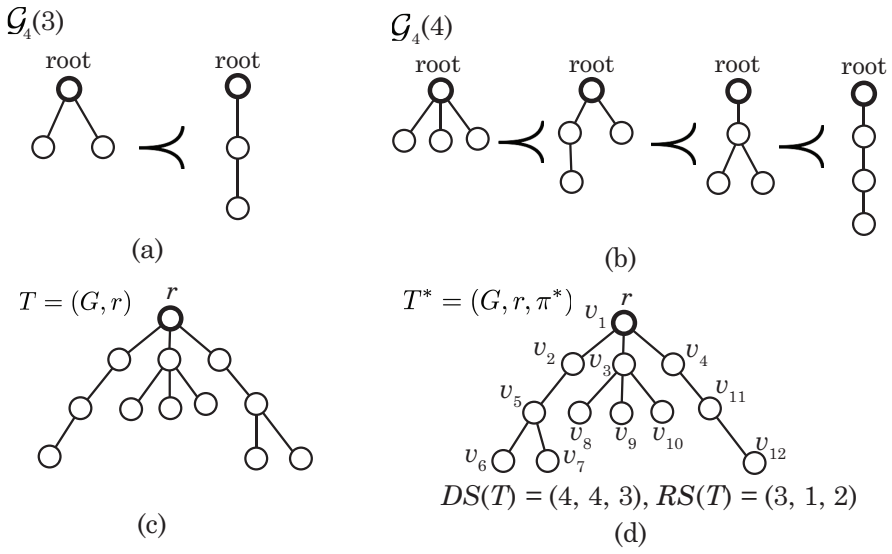

Figure 5: An example of a descendant sequence and a rank sequence of a rooted tree: (a) rooted trees in $\mathcal{G}_4(3)$; (b) rooted trees in $\mathcal{G}_4(4)$; (c) a rooted tree $T = (G, r)$ with 12 vertices whose maximum degree at most 4; and (d) the canonical tree of T indexed by the Siblings First DFS.

T_1^* is the 3rd tree in $\mathcal{G}_4(4)$. Similarly, from Figures 5 (a) and (b), we have $|V(T_2^*)| = 4$, $\mathrm{rank}_\Delta(T_2^*) = 1$, $|V(T_3^*)| = 3$ and $\mathrm{rank}_\Delta(T_3^*) = 2$. Therefore we have the descendant sequence $DS(T^*) = (4, 4, 3)$, and the rank sequence $RS(T^*) = (3, 1, 2)$.

The pair of the descendant sequence and the rank sequence of a given tree can serve as a unique representation of a tree. Hence we have the following lemma.

Lemma 12. *Let $n \geq 1$ and $\Delta \geq 2$ be two positive integers, and let $T = (G, r, \pi)$ and $T' = (G', r', \pi')$ be two canonical trees in $\mathcal{G}_\Delta(n)$. Then $L_D(T) \prec L_D(T')$ holds if and only if either (i) $DS(T) \prec DS(T')$, or (ii) $DS(T) = DS(T')$ and $RS(T) \prec RS(T')$.*

Proof. From the definition of the descendant representation, the first entry of both $L_D(T)$ and $L_D(T')$ is n, and the next $\deg(r)$ (resp., $\deg(r')$) entries of $L_D(T)$ (resp., $L_D(T')$) are the descendant sequence $DS(T)$ (resp., $DS(T')$).

First we show the necessity. Suppose that $L_D(T) \prec L_D(T')$ holds. For an integer $i \in [1, n]$, let x_i (resp., x'_i) denote the i-th entry of $L_D(T)$ (resp., $L_D(T')$). Let k denote the smallest integer such that $x_k < x'_k$. Since the first entry of both $L_D(T)$ and $L_D(T')$ is n, we have $k \geq 2$. If $k \leq 1 + \deg(r)$ holds, then we have $DS(T) \prec DS(T')$. Suppose that $k > 1 + \deg(r)$. In this case, the descendant sequences of T and T' are equivalent. This implies that $\deg(r) = \deg(r')$ and for each $i \in [1, \deg(r)]$, it holds that $|V(T_i)| = |V(T'_i)|$. Let v_k (resp., v'_k) denote the k-th vertex of T (resp., T') following

the Siblings First DFS ordering. For an integer i such that $v_k \in V(T_i)$, since $x_k < x'_k$ holds, we have $L_D(T_i) \prec L_D(T'_i)$, and we obtain $\mathrm{rank}_\Delta(T_i) < \mathrm{rank}_\Delta(T'_i)$ and $RS(T) \prec RS(T')$. Therefore if $L_D(T) \prec L_D(T')$ holds, then either (i) $DS(T) \prec DS(T')$, or (ii) $DS(T) = DS(T')$ and $RS(T) \prec RS(T')$ holds.

Next we show the sufficiency. Since we have $|V(T)| = |V(T')| = n$, if $DS(T) \prec DS(T')$, then $L_D(T) \prec L_D(T')$ holds. Suppose that $DS(T) = DS(T')$ and $RS(T) \prec RS(T')$ holds. Then, following the assumption that $DS(T) = DS(T')$, it holds that $\deg(r) = \deg(r')$ and $|V(T_i)| = |V(T'_i)|$, $1 \leq i \leq \deg(r)$, and there exists an integer k such that for all $1 \leq i < k$, $\mathrm{rank}_\Delta(T_i) = \mathrm{rank}_\Delta(T'_i)$ and $\mathrm{rank}_\Delta(T_k) < \mathrm{rank}_\Delta(T'_k)$. Since the lexicographical order does not change when entries of equal values are removed from or added to each of the sequences with the same position, we delete the first $(1 + \deg(r) + \sum_{i=1}^{k-1}(|V(T_i)| - 1))$ entries of $L_D(T)$ and $L_D(T')$, respectively, and then, we insert $|V(T_k)|$ $(= |V(T'_k)|)$ as the first entry of the modified $L_D(T)$ and $L_D(T')$. The first $|V(T_k)|$ of the obtained sequences shows the descendant representation of the k-th root-subtree $L_D(T_k)$ and $L_D(T'_k)$, respectively. From $\mathrm{rank}_\Delta(T_k) < \mathrm{rank}_\Delta(T'_k)$, we have $L_D(T_k) \prec L_D(T'_k)$. Since we have been only erasing and adding equivalent entries, it holds that $L_D(T) \prec L_D(T')$.

As a result, $L_D(T) \prec L_D(T')$ holds if and only if either (i) $DS(T) \prec DS(T')$, or (ii) $DS(T) = DS(T')$ and $RS(T) \prec RS(T')$. □

We mention the two relations about sequences: the relation between descendant sequences and positive descending sequences; and the relation between the rank sequences and the upper bounded sequences. We have the following lemma.

Lemma 13. *Let $n \geq 1$ and $\Delta \geq 2$ be two positive integers, and let $T = (G, r, \pi)$ be a canonical tree in $\mathcal{G}_\Delta(n)$. Let $(n_1, n_2, \ldots, n_{\deg(r)})$ be the descendant sequence $DS(T)$, let $(r_1, r_2, \ldots, r_{\deg(r)})$ be the rank sequence $RS(T)$, and let K be the sequence $(g_\Delta(n_1), g_\Delta(n_2), \ldots, g_\Delta(n_{\deg(r)}))$ Then it holds that $DS(T) \in \mathcal{D}(\Delta - 1, n - 1)$ and $(r_1 - 1, r_2 - 1, \ldots, r_{\deg(r)} - 1) \in \mathcal{S}(K)$.*

Proof. First we show $DS(T) \in \mathcal{D}(\Delta - 1, n - 1)$. The descendant sequence $DS(T)$ is a positive integer sequence with length $\deg(r) \leq \Delta - 1$, and it holds that $\mathrm{sum}(DS(T)) = n - 1$ since the number of vertices in all root-subtrees are the number of vertices in T except for the root. Suppose that the descendant sequence $DS(T)$ is not a descending sequence. Then there exists two integers $i \in [1, \deg(r)]$ and $j \in [i + 1, \deg(r)]$ such that $|V(T_i)| < |V(T_j)|$. In this case, for the ordered tree $T' = (G, r, \pi')$ obtained by flipping the i-th and j-th root-subtrees of T, we have $L_D(T') \prec L_D(T)$. This, however, contradicts that T is the canonical tree of (G, r), and it holds that the descendant sequence is a descending sequence. As a result, the descendant sequence $DS(T)$ can be regarded as a sequence in the positive descending sequence $\mathcal{D}(\Delta - 1, n - 1)$.

Next we prove that $(r_1 - 1, r_2 - 1, \ldots, r_{\deg(r)} - 1) \in \mathcal{S}(K)$. For two integers i and j with $1 \leq i < j \leq \deg(r)$, if $|V(T_i)| = |V(T_j)|$ holds, then we have $\mathrm{rank}_\Delta(T_i) <$

$\mathrm{rank}_\Delta(T_j)$. Note that this property comes from the definition of the canonical tree. Suppose that, for some $1 \leq i < j \leq \deg(r)$, if $\mathrm{rank}_\Delta(T_i) < \mathrm{rank}_\Delta(T_j)$ with $|V(T_i)| = |V(T_j)|$ holds, then the ordered tree obtained by flipping i-th and j-th root-subtree of T has greater descendant representation than T. This contradicts that T is canonical tree of (G, r). As a result, for the rank sequence $(r_1, r_2, \ldots, r_{\deg(r)})$ of T, it holds that $1 \leq r_i \leq g_\Delta(n_i)$, $1 \leq i \leq \deg(r)$, and $r_j \geq r_{j+1}$, $1 \leq j \leq n-1$ with $k_j = k_{j+1}$. Therefore the sequence $(r_1 - 1, r_2 - 1, \ldots, r_{\deg(r)} - 1)$ is an upper bounded sequence in $\mathcal{S}(K)$. □

For simplicity, in order to treat rank sequences as upper bounded sequence, we redefine each entry of the rank sequence to be rank minus one. For a canonical tree T in $\mathcal{G}_\Delta(n)$, let $(v_1, v_2, \ldots, v_n)$ denote the Siblings First DFS ordering of T, we define the *rank representation* $L_R(T)$ of T to be

$$L_R(T) \triangleq (\mathrm{rank}_\Delta(T(v_1)), \mathrm{rank}_\Delta(T(v_2)), \ldots, \mathrm{rank}_\Delta(T(v_n))).$$

Note that in order to reconstruct the structure of tree from the rank representation, we need some additional information since we do not have the number of vertices in each subtree.

3.3 Counting Degree-Bounded Rooted Trees

In this subsection, we devise an algorithm to obtain the number of degree-bounded rooted trees $g_\Delta(n)$ based on dynamic programming. We investigate some recursive relationships that hold for the number of trees in the family $\mathcal{G}_\Delta(n)$. For any rooted tree $T \in \mathcal{G}_\Delta(n)$ and a vertex $v \in V(T)$, the subtree $T(v)$ is in $\mathcal{G}_\Delta(|V(T(v))|)$. For four integers $n \geq 1$, $\Delta \geq 2$, $m \in [0, n-1]$, and $d \in [0, \min\{\Delta, n-1\}]$, let $\mathcal{G}_\Delta(n, m, d)$ denote the set of rooted trees with n vertices such that the maximum degree of the tree is at most Δ, the maximum number of vertices in a root-subtree is at most m, and the degree of the root is at most d. Let $g_\Delta(n, m, d)$ denote the number of trees in $\mathcal{G}_\Delta(n, m, d)$. We immediately get that, for all $n \geq 1$, it holds that $\mathcal{G}_\Delta(n) = \mathcal{G}_\Delta(n, n-1, \Delta-1)$ by the definition of $\mathcal{G}_\Delta(n)$ and $\mathcal{G}_\Delta(n, m, d)$.

For positive integers n and m, let $\mathrm{C_r}(n, m) \triangleq \binom{n+m-1}{m}$ denote the number of combinations with repetitions of m out of n elements. We give the following observation that follows from the definition of $g_\Delta(n, m, d)$.

Lemma 14. *Let $n \geq 1$, $\Delta \geq 2$, $m \in [0, n-1]$, and $d \in [0, \min\{\Delta, n-1\}]$ be four ineters. Then:*

(i) $g_\Delta(n, m, d) = 0$ *for* $n > md + 1$;

(ii) $g_\Delta(1, 0, 0) = 1$;

(iii) $g_\Delta(m) = g_\Delta(m, m-1, \Delta-1)$;

(iv) *for an integer* $\ell \geq 1$, $\mathrm{C_r}(g_\Delta(m), \ell) = \mathrm{C_r}(g_\Delta(m), \ell-1) \cdot (g_\Delta(m) + \ell - 1)/\ell$; *and*

(v)

$$g_\Delta(n,m,d) = g_\Delta(n,m-1,d) + \sum_{\substack{\max\{1,n+(1-m)d-1\}\le\ell,\\ \ell\le\min\{d,\lfloor(n-1)/m\rfloor\}}} \mathrm{C_r}(g_\Delta(m),\ell)\cdot g_\Delta(n-m\ell,m-1,d-\ell).$$

Proof. (i) For any tree in $\mathcal{G}_\Delta(n,m,d)$, the number of vertices is at most $md+1$. If $n > md+1$ holds, then $\mathcal{G}_\Delta(n,m,d) = \emptyset$ holds.
(ii) Clearly, $\mathcal{G}_\Delta(1,0,0)$ consists of one tree with a single vertex.
(iii) Immediate from $\mathcal{G}_\Delta(n) = \mathcal{G}_\Delta(n,n-1,\Delta-1)$.
(iv) For any positive integers $n \ge 1$ and $m \ge 1$, it holds that $\mathrm{C_r}(n,m) = \mathrm{C_r}(n,m-1)\cdot (n+m-1)/m$.
(v) We consider the number of trees in $\mathcal{G}_\Delta(n,m,d)$ such that there are ℓ root-subtrees with exactly m vertices. Note that such ℓ is at most $\min\{d,\lfloor(n-1)/m\rfloor\}$. If $\ell = 0$, then the number of such trees is $g_\Delta(n,m-1,d)$. For $\ell \ge 1$, the number of choices of ℓ such subtrees is equal to $\mathrm{C_r}(g_\Delta(m),\ell)$ since there are $g_\Delta(m)$ different candidates for each subtree. The remaining part obtained after eliminating the ℓ subtrees must be a rooted tree on $n-m\ell$ vertices such that each root-subtree has at most $m-1$ vertices, and the degree of the root is at most $d-\ell$. The number of such trees is equal to $g_\Delta(n-m\ell,m-1,d-\ell)$. Hence the number of trees in $\mathcal{G}_\Delta(n,m,d)$ such that there are ℓ root-subtrees with exactly m vertices is $\mathrm{C_r}(g_\Delta(m),\ell)\cdot g_\Delta(n-m\ell,m-1,d-\ell)$. From (i), if $n-m\ell > (m-1)(d-\ell)+1$ holds, then there are no such trees, and we have $g_\Delta(n-m\ell,m-1,d-\ell) = 0$. As a result, we obtain

$$g_\Delta(n,m,d) = g_\Delta(n,m-1,d) + \sum_{\substack{\max\{1,n+(1-m)d-1\}\le\ell,\\ \ell\le\min\{d,\lfloor(n-1)/m\rfloor\}}} \mathrm{C_r}(g_\Delta(m),\ell)\cdot g_\Delta(n-m\ell,m-1,d-\ell).$$

□

From Lemma 14, we propose an algorithm based on dynamic programming for counting $g_\Delta(n,m,d)$. Let $n \ge 1$, $\Delta \ge 2$, $m \in [0,n-1]$, and $d \in [0,\min\{\Delta,n-1\}]$ be four integers. For three integers $1 \le i \le n$, $1 \le j \le \min\{m,i-1\}$, and $k < \lceil i/j\rceil$, we have $g_\Delta(i,j,k) = 0$ from Lemma 14-(i). Hence it is enough to calculate only $g_\Delta(i,j,k)$, $1 \le i \le n$, $0 \le j \le \min\{m,i-1\}$, and $\lceil i/j\rceil \le k \le \min\{\Delta,i-1\}$. From Lemma 14-(ii), we initialize $g_\Delta[1,0,0]$ by 1. For three integers $1 \le i \le n$, $1 \le j \le \min\{m,i-1\}$, and $\lceil i/j\rceil \le k \le \min\{\Delta,i-1\}$, we calculate $g_\Delta(i,j,k)$ by Lemma 14-(v). From Lemma 14-(iii), we have $g_\Delta(m) = g_\Delta(m,m-1,\Delta-1)$, and from Lemma 14-(iv), given the value $\mathrm{C_r}(g_\Delta(m),\ell-1)$, we can obtain $\mathrm{C_r}(g_\Delta(m),\ell)$ in constant time. Therefore we obtain the value $g_\Delta(i,j,k)$ in $\mathcal{O}(k)$ time.

A description of the algorithm is shown in Algorithm 6. In this algorithm, $g_\Delta[i,j,k]$ stores the values of $g_\Delta(i,j,k)$, $1 \le i \le n$, $0 \le j \le \min\{m,i-1\}$, and $\lceil i/j\rceil \le k \le$

$\min\{\Delta, i-1\}$, and the variable r contains the value $\mathrm{C_r}(g_\Delta(m), \ell)$. From Lemma 14 and Algorithm 6, we have the following Theorem.

Algorithm 6 DP based counting algorithm for degree-bounded rooted trees

Input: Three integers $n \geq 1$, $\Delta \geq 2$, and $m \in [0, n-1]$.

Output: $g_\Delta(i,j,k)$, $1 \leq i \leq n$, $0 \leq j \leq \min\{m, i-1\}$, and $\lceil i/j \rceil \leq k \leq \min\{\Delta, i-1\}$.

```
1:  g_Δ[1,0,0] := 1;
2:  for i := 2,3,...,n do
3:      for j := 1,...,min{m, i−1} do
4:          for k := ⌈i/j⌉,...,min{Δ, i−1} do
5:              g_Δ[i,j,k] := 0; r := 1;
6:              for ℓ := max{1, n+(1−m)d−1},...,min{k, ⌊(i−1)/j⌋} do
7:                  r := r·(g_Δ[j, j−1, Δ−1] + ℓ − 1)/ℓ;
8:                  i' := i − jℓ; j' := min{j−1, i'−1}; k' := min{k−ℓ, i'−1};
9:                  g_Δ[i,j,k] := g_Δ[i,j,k] + r·g_Δ[i',j',k']
10:             end for;
11:             if i − 1 ≤ (j−1)k then
12:                 g_Δ[i,j,k] := g_Δ[i,j,k] + g_Δ[i,j−1,k]
13:             end if
14:         end for
15:     end for
16: end for;
17: output g_Δ[i,j,k], 1 ≤ i ≤ n, 0 ≤ j ≤ min{m, i−1}, and ⌈i/j⌉ ≤ k ≤ min{Δ, i−1}
```

Theorem 6. *Let $n \geq 1$, $\Delta \geq 2$, and $m \in [0, n-1]$ be three integers. Then the set of numbers $g_\Delta(i,j,k)$, $1 \leq i \leq n$, $0 \leq j \leq \min\{m, i-1\}$, and $\lceil i/j \rceil \leq k \leq \min\{\Delta, i-1\}$ can be obtained in $\mathcal{O}(nm\Delta^2)$ time and $\mathcal{O}(nm\Delta)$ space.*

Proof. Lemma 14 gives us a recursive structure of $g_\Delta(i,j,k)$. These values can be computed by using dynamic programming and storing a table of size $\mathcal{O}(nm\Delta)$ for storing the values of $g_\Delta(i,j,k)$, $1 \leq i \leq n$, $0 \leq j \leq \min\{m, i-1\}$, and $\lceil i/j \rceil \leq k \leq \min\{\Delta, i-1\}$. In the case of Lemma 14-(i) and (ii), we can obtain the value $g_\Delta(i,j,k)$ in $\mathcal{O}(1)$ time. For Lemma 14-(iii), (iv), and (v), computing the entry $g_\Delta(i,j,k)$ can be done in $\mathcal{O}(k)$ time since given the value $\mathrm{C_r}(g_\Delta(m), \ell-1)$, we can obtain $\mathrm{C_r}(g_\Delta(m), \ell)$ in constant time. Therefore the set of numbers $g_\Delta(i,j,k)$, $1 \leq i \leq n$, $0 \leq j \leq \min\{m, i-1\}$, and $\lceil i/j \rceil \leq k \leq \min\{\Delta, i-1\}$ can be obtained in $\mathcal{O}(nm\Delta^2)$ time and $\mathcal{O}(nm\Delta)$ space. □

4 Enumerating Degree-Bounded Trees

In this section, we discuss a sequential enumeration algorithm for degree-bounded rooted/unrooted trees. In subsection 4.1, we propose an algorithm to enumerate all degree-bounded trees in $\mathcal{G}_\Delta(n)$ following the lexicographical ascending order of descendant representation σ_{des}. Given a degree-bounded rooted tree T in $\mathcal{G}_\Delta(n)$, we generate the predecessor tree by updating root-subtrees of T. In subsection 4.2, we introduce a problem setting for enumerating rooted trees which restricts the degree of the root and the number of vertices in a root-subtree. We will show that the enumeration of this problem setting can be done by using the sequential algorithm for $\mathcal{G}_\Delta(n)$. Based on this observation, we propose an algorithm to enumerate all degree-bounded unrooted trees.

4.1 Enumeration for Degree-Bounded Rooted Trees

We enumerate all rooted trees in $\mathcal{G}_\Delta(n)$ by generating the predecessor tree one by one following the lexicographical ascending order of descendant representation σ_{des}. Given the descendant sequence and the rank sequence of a given tree in $\mathcal{G}_\Delta(n)$, we examine how to generate the descendant sequence and the rank sequence of the predecessor tree of the given tree.

Lemma 15. *Let $n \geq 1$ and $\Delta \geq 2$ be two positive integers, and let $T = (G, r, \pi)$ be a canonical tree in $\mathcal{G}_\Delta(n)$. Let $(n_1, n_2, \ldots, n_{\deg(r)})$ denote the descendant sequence of T, and let K denote the sequence $(g_\Delta(n_1), g_\Delta(n_2), \ldots, g_\Delta(n_{\deg(r)}))$. Then:*

(i) *The path P_n with length n rooted at one of its endpoints has the lexicographically maximum canonical representation among $\mathcal{G}_\Delta(n)$, and it holds that $L_D(P_n) = (n, n-1, \ldots, 1)$ and $L_R(P_n) = (g_\Delta(n) - 1, g_\Delta(n-1) - 1, \ldots, g_\Delta(1) - 1)$.*

(ii) *If $DS(T) = \min(\mathcal{D}(\Delta - 1, n - 1))$ and $RS(T) = \min(\mathcal{S}(K))$ holds, then T has no predecessor.*

(iii) *For the predecessor tree $T' = (G', r', \pi')$ of T in $\mathcal{G}_\Delta(n)$, it holds that either*

  (a) *$DS(T') = DS(T)$, and $RS(T')$ is the lexicographic predecessor of $RS(T)$ in $\mathcal{S}(K)$; or*

  (b) *$DS(T')$ is the lexicographic predecessor of $DS(T)$ in $\mathcal{D}(\Delta - 1, n - 1)$, and $RS(T') = \max(\mathcal{S}(K))$.*

(iv) *Testing whether T has a predecessor tree or not can be done in $\mathcal{O}(\Delta)$ time, and the descendant sequence and the rank sequence of the predecessor tree of T if one exists can be obtained in $\mathcal{O}(\Delta)$ time and $\mathcal{O}(\Delta)$ space.*

Proof. (i) For all rooted trees in $\mathcal{G}_\Delta(n)$, let T be the canonical tree with the maximum canonical representation among $\mathcal{G}_\Delta(n)$. The lexicographically maximum descendant sequence among all canonical trees in $\mathcal{G}_\Delta(n)$ is $(n-1)$. This implies that T has only one root-subtree with $n-1$ vertices. We apply this argument recursively, and we see that T is a path with length n rooted at one of its endpoints. Therefore we obtain $L_D(T) = (n, n-1, \ldots, 1)$ and $L_R(T) = (g_\Delta(n)-1, g_\Delta(n-1)-1, \ldots, g_\Delta(1)-1)$.
(ii) From Lemma 12, there exists no canonical trees $T' \in \mathcal{G}_\Delta(n)$ such that $L_D(T') \prec L_D(T)$.
(iii) Immediate from Lemmas 12 and 13.
(iv) From Lemma 15-(ii), in order to determine whether T has a predecessor tree or not, we examine whether $DS(T) = \min(\mathcal{D}(\Delta-1, n-1))$ and $RS(T) = \min(\mathcal{S}(K))$ hold or not. From Theorem 1 and Theorem 5, this can be done in $\mathcal{O}(\Delta)$ time.

From Theorem 5, the predecessor sequence of $DS(T)$ in $\mathcal{D}(\Delta-1, n-1)$ if one exists can be calculated in $\mathcal{O}(\Delta)$ time and $\mathcal{O}(\Delta)$ space. From Theorem 1, the lexicographic predecessor of $RS(T)$ in $\mathcal{S}(K)$ if one exists can be obtained in $\mathcal{O}(\Delta)$ time and $\mathcal{O}(\Delta)$ space. In addition, $\max(\mathcal{S}(K))$ is $(g_\Delta(n_1)-1, g_\Delta(n_2)-1, \ldots, g_\Delta(n_{\deg(r)})-1)$, and this can be obtained in $\mathcal{O}(\Delta)$ time. Therefore from Lemma 15-(iii), the descendant sequence and the rank sequence of the predecessor tree of T can be obtained in $\mathcal{O}(\Delta)$ time and $\mathcal{O}(\Delta)$ space. □

Now we devise an algorithm to construct the predecessor tree T' of a given canonical tree T in $\mathcal{G}_\Delta(n)$ by updating root-subtrees of T if necessary. An overview of our algorithm as follows. If $DS(T) = \min(\mathcal{D}(\Delta-1, n-1))$ and $RS(T) = \min(\mathcal{S}(K))$ holds, then by Lemma 15-(ii), we see that T has no predecessor tree in $\mathcal{G}_\Delta(n)$. Otherwise, we construct the descendant sequence and the rank sequence of the predecessor tree T' of T by Lemma 15-(iii). Based on these sequences, we modify the root-subtrees of T if the number of vertices in the root-subtree or the rank will change. Note that, if both descendant sequence of T and T' are the same, and the rank of i-th root subtree of T and T' are also equivalent, then we see that the i-th root subtree of T and T' are rooted isomorphic, and there is no need to update the i-th root subtree in order to construct T'.

Suppose that all entries of $RS(T)$ are zero but $DS(T) \neq \min(\mathcal{D}(\Delta-1, n-1))$ holds. In this case, the descendant sequence $(n'_1, n'_2, \ldots, n'_{\ell'})$ of the predecessor tree is the lexicographic predecessor of $DS(T)$ in $\mathcal{D}(\Delta-1, n-1)$, and the rank sequence $(r_1, r_2, \ldots, r_\ell)$ of the predecessor tree satisfies $r_i = g_\Delta(n'_i) - 1$, $1 \leq i \leq \ell$. From Lemma 15-(i), for each $i \in [1, \ell']$, the i-th root-subtree of the predecessor tree is a path with length n'_i rooted at one of its endpoints. Since the descendant representation and the rank representation of the path are trivial, we obtain the descendant representation and the rank representation of the predecessor tree of T.

Suppose that there exists a non-zero entry in $RS(T)$, then we have $DS(T) = DS(T')$, and $RS(T)$ is the lexicographic predecessor of $RS(T)$ of the upper bounded sequences. We update each root-subtree of T as follows. Let $(n'_1, n'_2, \ldots, n'_{\deg(r')})$ denote the descendant sequence of T', and let $(r_1, r_2, \ldots, r_{\deg(r)})$ (resp., $(r'_1, r'_2, \ldots, r'_{\deg(r')})$) denote the rank sequence of T (resp., T'). From Lemmas 5 and 15, for each $i \in [1, \deg(r')]$, r'_i is the following four possible values: (i) $r'_i = g_\Delta(n'_i) - 1$, (ii) $r'_i = r'_{i-1}$, (iii) $r'_i = r_i - 1$, or (iv) $r'_i = r_i$. For case (i), from Lemma 15-(i), the i-th root-subtree T'_i of T' is the path with length n'_i rooted at one of its endpoints, and we have $L_D(T'_i) = (n'_i, n'_i - 1, \ldots, 1)$ and $L_R(T) = (g_\Delta(n'_i) - 1, g_\Delta(n'_i - 1) - 1, \ldots, g_\Delta(1) - 1)$. In case (ii) $r'_i = r'_{i-1}$, the i-th and $(i-1)$-th root-subtrees of T' are rooted isomorphic. Hence the update for i-th root-subtree can be done by copying the $(i-1)$-th root-subtree in T'. In case (iii) $r'_i = r_i - 1$, this update can be done by generating the predecessor tree of T_i. This update will be repeated recursively for each of the root-subtrees. In case (iv) $r'_i = r_i$, we see that the i-th root-subtree of T and T' are rooted isomorphic, and there is no need to update the i-th root-subtree.

We show a description of the algorithm to generate the predecessor tree of a given tree in $\mathcal{G}_\Delta(n)$ if one exists in Algorithm 7, and we have the following theorem.

Theorem 7. *Let $n \geq 1$ and $\Delta \geq 2$ be two positive integers, and let L_D and L_R be the descendant representation and the rank representation of a canonical tree $T = (G, r, \pi) \in \mathcal{G}_\Delta(n)$. Assume that the values of $g_\Delta(i), 1 \leq i \leq n$ are already obtained. Then testing whether T has a predecessor tree or not can be done in $\mathcal{O}(\Delta)$ time, and the descendant representation and the rank representation of the predecessor tree of T in $\mathcal{G}_\Delta(n)$ if one exists can be generated in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space.*

Proof. From the descendant representation and the rank representation, the descendant sequence $DS(T)$ and the rank sequence $RS(T)$ can be obtained in $\mathcal{O}(\Delta)$ time since $DS(T)$ and $RS(T)$ are the subsequence of L_D and L_R from 2nd entry to $(1+\deg(r))$-th entry, respectively. From Lemma 15-(iv), testing whether T has a predecessor tree or not can be done in $\mathcal{O}(\Delta)$.

Suppose that T has a predecessor tree in $\mathcal{G}_\Delta(n)$. From Lemma 15-(iv), the descendant sequence and the rank sequence of the predecessor tree of T can be constructed in $\mathcal{O}(\Delta)$ time.

We estimate the time complexity of updating the root-subtrees for each cases. Let $T' = (G', r', \pi')$ be the predecessor tree of T in $\mathcal{G}_\Delta(n)$. For $1 \leq i \leq \deg(r')$, let n_i (resp., r_i) denote the i-th entry of the descendant sequence (resp., rank sequence) of T'. The first case is $r_i = g_\Delta(n_i) - 1$. In this case, the corresponding subtree is the path with length n_i. This modification can be done in $\mathcal{O}(n_i)$ time. In the second case $r_i = r_{i-1}$, the i-th root-subtree will be the same as the $(i-1)$-th root-subtree in T'. This modification can be done in $\mathcal{O}(n_i)$ time by copying the left subtree. In the last case, we construct

Algorithm 7 Generate the predecessor of a degree-bounded rooted tree in $\mathcal{G}_\Delta(n)$

Input: Two positive integers $n \geq 1$ and $\Delta \geq 2$, the values of $g_\Delta(i), 1 \leq i \leq n$, the descendant representation $L_D(T)$, and the rank representation $L_R(T)$ of a canonical tree T in $\mathcal{G}_\Delta(n)$ rooted at a vertex r.

Output: The descendant representation $L_D(T')$ and the rank representation $L_R(T')$ of the predecessor tree T' of T in $\mathcal{G}_\Delta(n)$, if T has no predecessor tree in $\mathcal{G}_\Delta(n)$, then the output is a message, "T is minimum."

$k :=$ the first entry of $L_R(T)$;
$(n_1, n_2, \ldots, n_{\deg(r)}) := DS(T)$ obtained from $L_D(T)$;
$(r_1, r_2, \ldots, r_{\deg(r)}) := RS(T)$ obtained from $L_R(T)$;
if $r_i = 0$ for all $1 \leq i \leq \deg(r)$ **then**
    if $DS(T) = \min(\mathcal{D}(\Delta - 1, n - 1))$ **then**
        output "T is minimum"
    else
        $(n'_1, n'_2, \ldots, n'_\ell) :=$ the lexicographic predecessor of $DS(T)$ in $\mathcal{D}(\Delta - 1, n - 1)$;
        $L_D[i] := (n'_i - 1, n'_i - 2, \ldots, 1)$ for $1 \leq i \leq \ell$;
        $L_R[i] := (g_\Delta(n'_i - 1) - 1, g_\Delta(n'_i - 2) - 1, \ldots, g_\Delta(1) - 1)$ for $1 \leq i \leq \ell$;
        $L_D[T'] := (n, n'_1, \ldots, n'_\ell, L_D[1], L_D[2], \ldots, L_D[\ell])$;
        $L_R[T'] := (k - 1, t'_1, \ldots, t'_\ell, L_R[1], L_R[2], \ldots, L_R[\ell])$;
        output $L_D[T']$ and $L_R[T']$ as $L_D(T')$ and $L_R(T')$, respectively.
    end if
else /* $K = (g_\Delta(n_1), g_\Delta(n_2), \ldots, g_\Delta(n_{\deg(r)}))$ */
    $(r'_1, r'_2, \ldots, r'_{\deg(r)}) :=$ the lexicographic predecessor of $RS(T)$ in $\mathcal{S}(K)$;
    for $i := 1, 2, \ldots, \deg(r)$ **do**
        if $i > 1$ and $n_i = n_{i-1}$ and $r'_i = r'_{i-1}$ **then**
            /* i-th root-subtree is rooted isomorphic to the left root-subtree */
            $L_D[i] := L_D[i - 1]$; $L_R[i] := L_R[i - 1]$
        else if $r'_i = g_\Delta(n_i) - 1$ **then**
            /* i-th root-subtree is the path of length n'_i */
            $L_D[i] := (n'_i - 1, n'_i - 2, \ldots, 1)$;
            $L_R[i] := (g_\Delta(n'_i - 1) - 1, g_\Delta(n'_i - 2) - 1, \ldots, g_\Delta(1) - 1)$
        else if $r'_i = r_i - 1$ **then**
            /* i-th root-subtree is the predecessor tree in $\mathcal{G}_\Delta(n_i)$ */
            $L_D[i], L_R[i] :=$ **Algorithm7**$(n_i, \Delta, g_\Delta, L_D(T_i), L_R(T_i))$;
            Delete the first entry of $L_D[i]$ and $L_R[i]$
        end if
    end for;
    $L_D[T'] := (n, n_1, \ldots, n_{\deg(r)}, L_D[1], L_D[2], \ldots, L_D[\deg(r)])$;
    $L_R[T'] := (k - 1, r'_1, \ldots, r'_{\deg(r)}, L_R[1], L_R[2], \ldots, L_R[\deg(r)])$;
    output $L_D[T']$ and $L_R[T']$ as $L_D(T')$ and $L_R(T')$, respectively
end if

the predecessor tree of the root-subtree, and the update should be repeated recursively. However, this operation will repeat at most the number of vertices of the subtree. As a result, the update for each subtree can be done in $\mathcal{O}(n_i)$ time. Since we have $\Delta < n$ and $\sum_{i=1}^{\deg(r')} n_i = n - 1$, the descendant representation and the rank representation of the predecessor tree of T in $\mathcal{G}_\Delta(n)$ can be obtained in $\mathcal{O}(\Delta + \Delta + \sum_{i=1}^{\deg(r')} n_i) = \mathcal{O}(n)$ time.

Since it is sufficient to store only the values of $g_\Delta(i), 0 \leq i \leq n$ of n elements, the descendant representation, and the rank representation, the space complexity is $\mathcal{O}(n)$. □

4.2 Enumerating Degree Bounded Unrooted Trees

Based on the proposed algorithm for the sequential enumeration of degree-bounded rooted trees, we devise an algorithm for another problem setting for enumerating rooted trees, which restricts the degree of the root and the number of vertices in a root-subtree. For four positive integers $n \geq 1$, $\Delta \geq 2$, $d \leq \Delta$, and $m < n$, let $\mathcal{G}_\Delta(n, d, m)$ denote a maximal set of rooted trees with n vertices satisfying:

(i) the maximum degree is at most Δ;

(ii) the degree of the root is at most d;

(iii) the number of vertices in a root-subtree is at most m; and

(iv) no two graphs in $\mathcal{F}_\Delta(n, d, m)$ are rooted isomorphic.

Note that a root-subtree T' of $T \in \mathcal{G}_\Delta(n, d, m)$ is in $\mathcal{G}_\Delta(|V(T)|)$. The enumeration for $\mathcal{G}_\Delta(n, d, m)$ can be done as follows. In a similar way as in the case of $\mathcal{G}_\Delta(n)$, for a rooted tree $T \in \mathcal{G}_\Delta(n, d, m)$, the number of vertices in each root-subtree of T can be represented as a sequence in $\mathcal{D}(d, n-1, m)$ since the number of root-subtree is bounded by d, the number of vertices in all root-subtree is $n - 1$, and the number of vertices in a root-subtree is bounded by m. For the sequence $(n_1, n_2, \ldots, n_\ell)$ in $\mathcal{D}(d, n-1, m)$, the sequence $K = (g_\Delta(n_1), g_\Delta(n_2), \ldots, g_\Delta(n_\ell))$, and an upper bounded sequence $R = (r_1, r_2, \ldots, r_\ell)$ in $\mathcal{S}(K)$, for each $1 \leq i \leq \ell$, we construct the r_i-th tree T_i in $\mathcal{G}_\Delta(n_i)$, and we combine a root vertex and each T_i with an edge, then we obtain a rooted tree $T \in \mathcal{G}_\Delta(n, d, m)$. Therefore the enumeration for $\mathcal{G}_\Delta(n, d, m)$ can be done by the similar technique of the sequential enumeration for $\mathcal{G}_\Delta(n)$.

From these observation, we propose an algorithm to enumerate all degree-bounded unrooted trees. For positive integers $n \geq 1$ and $\Delta \geq 2$, let $\mathcal{F}_\Delta(n)$ denote a maximal set of unrooted trees with n vertices whose maximum degree is at most Δ such that no two graphs in $\mathcal{F}_\Delta(n)$ are isomorphic. We devise an algorithm for enumerating all unrooted trees in $\mathcal{F}_\Delta(n)$ by using the prescribed problem setting. Let T be an

unrooted tree in $\mathcal{F}_\Delta(n)$. If T has a bicentroid, then we can regard T as an edge with two canonical trees in $\mathcal{G}_\Delta(n/2)$. Hence the enumeration of degree-bounded trees with the bicentroid is equivalent to the enumeration of unordered pairs of canonical trees in $\mathcal{G}_\Delta(n/2)$. On the other hand, if T has the unicentroid, then the number of vertices in the root-subtrees of T rooted at its centroid can be bounded by $\lfloor (n-1)/2 \rfloor$. Therefore the enumeration of degree-bounded trees with the unicentroid is equivalent to the enumeration of $\mathcal{G}_\Delta(n, \Delta, \lfloor (n-1)/2 \rfloor)$.

We propose a two-phase scheme for enumerating all trees in $\mathcal{F}_\Delta(n)$. A description of an algorithm is shonw in Algorithm 10. In the first phase, we pre-compute the values of $g_\Delta(i), 0 \leq i \leq \lfloor n/2 \rfloor$, according to Theorem 6. In the second phase, we generate bicentroid-rooted trees and unicentroid-rooted trees, respectively. We show an algorithm for enumeration all trees in $\mathcal{F}_\Delta(n)$ rooted at their bicentroid and unicentroid separately. In the bicentroid case, first we construct two paths with $n/2$ vertices rooted at one of its end-points, and we generate all unordered pairs of canonical trees in $\mathcal{G}_\Delta(n/2)$ by using the sequential enumeration for rooted trees. A description of the algorithm is shown in Algorithm 8. In the unicentroid case, we enumerate all trees in $\mathcal{G}_\Delta(n, \Delta, \lfloor (n-1)/2 \rfloor)$ sequentialy. A description of the algorithm is shown in Algorithm 9. Finally we have Theorem 8, and from Theorems 6 and 8, we obtain Theorem 9.

Algorithm 8 Generate all trees in $\mathcal{F}_\Delta(n)$ rooted at their bicentroid

Input: Two positive integers $n \geq 1$ and $\Delta \geq 2$, and the values of $g_\Delta(i), 1 \leq i \leq n$.
Output: All trees in $\mathcal{F}_\Delta(n)$ rooted at their bicentroid.

1: **if** n is even **then**
2: Initialize the rank sequence $R = (r_1, r_2) := (g_\Delta(n/2) - 1, g_\Delta(n/2) - 1)$;
3: Initialize the previous rank sequence $R' = (r'_1, r'_2) := (-1, -1)$;
4: /*Let $L_D[1]$ and $L_D[2]$ be the path with length $n/2$ */
5: $L_D[1] := (n/2, n/2 - 1, \ldots, 1)$; $L_D[2] := L_D[1]$;
6: $L_R[1] := (g_\Delta(n/2) - 1, g_\Delta(n/2 - 1) - 1, \ldots, g_\Delta(1) - 1)$; $L_R[2] := L_R[1]$;
7: /* $K = (g_\Delta(n/2), g_\Delta(n/2))$ */
8: **while** $R \neq \min(\mathcal{S}(K))$ **do**
9: **for each** $i := 1, 2$ with $r_i \neq r'_i$ **do**
10: Update $L_D[i]$ and $L_R[i]$ to the r_i-th tree in $\mathcal{G}_\Delta(n_i)$
11: **end for**;
12: **output** $(L_D[1], L_D[2])$;
13: $R' := R$; R:= the lexicographic predecessor sequence of R in $\mathcal{S}(K)$
14: **end while**
15: **end if**

Algorithm 9 Generate all trees in $\mathcal{F}_\Delta(n)$ rooted at their unicentroid

Input: Two positive integers $n \geq 1$ and $\Delta \geq 2$, and the values of $g_\Delta(i), 1 \leq i \leq \lfloor n/2 \rfloor$.
Output: All trees in $\mathcal{F}_\Delta(n)$ rooted at their unicentroid.

$D := \max(\mathcal{D}(\Delta, n-1, \lfloor (n-1)/2 \rfloor))$;
while $D \neq \min(\mathcal{D}(\Delta, n-1, \lfloor (n-1)/2 \rfloor))$ **do**
    /* $D = (n_1, n_2, \ldots, n_{|D|})$ */
    /* $K = (g_\Delta(n_1), g_\Delta(n_2), \ldots, g_\Delta(n_{|D|}))$ */
    $R = (r_1, r_2, \ldots, r_{|D|}) := \max(\mathcal{S}(K))$;
    $R' = (r'_1, r'_2, \ldots, r'_{|D|}) := (-1, -1, \ldots, -1)$;
    while $R \neq \min(\mathcal{S}(K))$ **do**
        for each $i := 1, 2, \ldots, |D|$ with $r_i \neq r'_i$ **do**
            $L_D[i] :=$ the descendant representation of r_i-th tree in $\mathcal{G}_\Delta(n_i)$;
            Delete the first entry of $L_D[i]$
        end for;
        $L_D := (n, n_1, \ldots, n_{|D|}, L_D[1], L_D[2], \ldots, L_D[|D|])$;
        output L_D;
        $R' := R$; $R :=$ the lexicographic predecessor sequence of R in $\mathcal{S}(K)$
    end while;
    $D' := D$;
    $D :=$ the lexicographic predecessor sequence of D in $\mathcal{D}(\Delta, n-1)$
end while

Algorithm 10 Enumerate all trees in $\mathcal{F}_\Delta(n)$

Input: Two positive integers $n \geq 1$ and $\Delta \geq 2$.
Output: All trees in $\mathcal{F}_\Delta(n)$.

/*Phase 1: Counting*/
$S := \{g_\Delta(i) \mid 0 \leq i \leq \lfloor n/2 \rfloor\}$ by **Algorithm 6**;
/*Phase 2: Enumeration*/
Algorithm8(n, Δ, S); /* Enumeration for Trees with Bicentroid */
Algorithm9(n, Δ, S) /* Enumeration for Trees with Unicentroid */

Theorem 8. *Let $n \geq 1$ and $\Delta \geq 2$ be two positive integers. Assume that the values of $g_\Delta(i), 1 \leq i \leq \lfloor n/2 \rfloor$ are already obtained. Then all trees in $\mathcal{F}_\Delta(n)$ can be generated in $\mathcal{O}(n)$ time per tree and $\mathcal{O}(n)$ space.*

Proof. We generate all trees with bicentroid and all trees with unicentroid in $\mathcal{F}_\Delta(n)$, respectively. In the bicentroid case, the descendant representation and the rank representation of two paths with $n/2$ vertices rooted at one of its end-points can be obtained in $\mathcal{O}(n)$ time by Lemma 15-(i). From Theorem 7, all unordered pairs of canonical trees

in $\mathcal{G}_\Delta(n/2)$ can be obtained in $\mathcal{O}(n)$ time per unordered pair by using the sequential enumeration for rooted trees.

In the unicentroid case, we enumerate trees in $\mathcal{G}_\Delta(n, \Delta, \lfloor (n-1)/2 \rfloor)$ sequentialy. Similarly, by Theorem 7, even if all root-subtrees are updated, all trees can be generated in $\mathcal{O}(n)$ time per tree. As a result, all trees in $\mathcal{F}_\Delta(n)$ can be generated in $\mathcal{O}(n)$ time per tree.

The values of $g_\Delta(i), 1 \leq i \leq \lfloor n/2 \rfloor$, contain $\lfloor n/2 \rfloor$ integers, and the amount of space of the descendant representation and the rank representation of a tree with n vertices is also bounded by n. Thus the space complexity is $\mathcal{O}(n)$. □

Theorem 9. *Given the number of vertices $n \geq 1$ and the maximum degree bound $\Delta \geq 2$, after $\mathcal{O}(n^2\Delta^2)$ time pre-processing, all trees in $\mathcal{F}_\Delta(n)$ can be enumerated in $\mathcal{O}(n)$ time per tree and $\mathcal{O}(n^2\Delta)$ space in total.*

5 Ranking for Degree Bounded Rooted Trees

In this section, for two positive integers $n \geq 1$ and $\Delta \geq 2$, given a rooted tree $T = (G, r)$ with n vertices such that the maximum degree of T is at most Δ, and the degree of root is at most $\Delta - 1$, we devise an algorithm to identify the rank of the canonical tree of T in $\mathcal{G}_\Delta(n)$. We design a bottom-up algorithm to calculate the rank of a given rooted tree. Suppose that we can derive the rank of a tree from its descendant sequence and the rank sequence. We apply the rank calculation in BFS Post-order, since the descendant sequence and the rank sequence of the leaves are obvious. Note that, given the number of vertices and the rank for each root-subtree of a rooted tree, the descendant sequence and the rank sequence can be obtained by sorting.

Given positive integers $n \geq 1$ and $\Delta \geq 2$, and a rooted tree $T = (G, r)$ with n vertices such that the maximum degree of T is at most Δ, and the degree of root is at most $\Delta - 1$, the *ranking problem* is to determine the rank of the canonical tree of T in $\mathcal{G}_\Delta(n)$ in the lexicographical ascending order of descendant representation σ_{des}. Now we show how to obtain the rank of a tree from its descendant sequence and the rank sequence.

For positive integers $n \geq 1$ and $\Delta \geq 2$, let $T = (G, r)$ be a rooted tree with n vertices such that the maximum degree of T is at most Δ, and the degree of root is at most $\Delta - 1$, and let T^* be the canonical tree of T. For a canonical tree $T' \in \mathcal{G}_\Delta(n)$, we say that T' is *descendant sequence predecessor* of T if $DS(T') \prec DS(T^*)$ holds, and we say that T' is *rank sequence predecessor* of T if $DS(T') = DS(T^*)$ and $RS(T') \prec RS(T^*)$ hold. Let $\mathcal{D}_\Delta(T)$ denote the set of all descendant sequence predecessors of T, and let $\mathcal{R}_\Delta(T)$ denote the set of all rank sequence predecessors of T. From the definition of the rank of degree-bounded rooted tree, the rank of the canonical tree of T is $|\mathcal{D}_\Delta(T)| + |\mathcal{R}_\Delta(T)| + 1$.

In subsection 5.1, we mention how to obtain the number of descendant sequence predecessors, and in subsection 5.2, we devise how to obtain the number of rank sequence predecessors.

5.1 The Number of Descendant Sequence Predecessors

In order to obtain the number of the descendant sequence predecessors, we show the following lemma.

Lemma 16. *Let $n \geq 1$, $\Delta \geq 2$, $m \in [0, n-1]$ and $d \in [0, \min\{\Delta, n-1\}]$ be four integers, and let $T = (G, r)$ be a rooted tree such that its canonical tree is in $\mathcal{G}_\Delta(n, m, d)$. Let $(n_1, n_2, \ldots, n_{\deg(r)})$ denote the descendant sequence of T, and let $\ell \in [1, \deg(r)]$ denote the smallest integer such that $n_\ell > n_{\ell+1}$. Then the following recursive equation holds:*

$$|\{T' \in \mathcal{G}_\Delta(n, m, d) \mid DS(T') \prec DS(T)\}|$$

$$= g_\Delta(n, n_1 - 1, d) + \sum_{i=1}^{\ell-1} \mathrm{C_r}(g_\Delta(n_1), i) \cdot g_\Delta(n - in_1, n_1 - 1, d - i)$$
$$+\mathrm{C_r}(g_\Delta(n_1), \ell) \cdot |\{T' \in \mathcal{G}_\Delta(n - \ell n_1, n_1 - 1, d - \ell) \mid DS(T') \prec (n_{\ell+1}, \ldots, n_{\deg(r)})\}|.$$

Proof. Let $T' = (G', r', \pi')$ be a canonical tree in $\mathcal{G}_\Delta(n, m, d)$, and let $(n'_1, n'_2, \ldots, n'_{\deg(r')})$ denote the descendant sequence of T'. From Lemma 14, we have the following three equations.

$$|\{T' \in \mathcal{G}_\Delta(n, m, d) \mid n'_1 < n_1\}| = g_\Delta(n, n_1 - 1, d). \tag{1}$$

$$|\{T' \in \mathcal{G}_\Delta(n, m, d) \mid n'_1 = n_1, (n'_2, \ldots, n'_\ell) \prec (n_2, \ldots, n_\ell)\}|$$
$$= \sum_{i=1}^{\ell-1} \mathrm{C_r}(g_\Delta(n_1), i) \cdot g_\Delta(n - in_1, n_1 - 1, d - i). \tag{2}$$

$$|\{T' \in \mathcal{G}_\Delta(n, m, d) \mid (n'_1, \ldots, n'_\ell) = (n_1, \ldots, n_\ell), DS(T') \prec DS(T)\}|$$
$$= \mathrm{C_r}(g_\Delta(n_1), \ell) \cdot |\{T' \in \mathcal{G}_\Delta(n - \ell n_1, n_1 - 1, d - \ell) \mid DS(T') \prec (n_{\ell+1}, \ldots, n_{\deg(r)})\}|. \tag{3}$$

Therefore we have

$$|\{T' \in \mathcal{G}_\Delta(n, m, d) \mid DS(T') \prec DS(T)\}|$$
$$= |\{T' \in \mathcal{G}_\Delta(n, m, d) \mid n'_1 < n_1\}|$$
$$+|\{T' \in \mathcal{G}_\Delta(n, m, d) \mid n'_1 = n_1, (n'_2, \ldots, n'_\ell) \prec (n_2, \ldots, n_\ell)\}|$$
$$+|\{T' \in \mathcal{G}_\Delta(n, m, d) \mid (n'_1, \ldots, n'_\ell) = (n_1, \ldots, n_\ell), DS(T') \prec DS(T)\}|$$
$$= g_\Delta(n, n_1 - 1, d) + \sum_{i=1}^{\ell-1} \mathrm{C_r}(g_\Delta(n_1), i) \cdot g_\Delta(n - in_1, n_1 - 1, d - i)$$
$$+\mathrm{C_r}(g_\Delta(n_1), \ell) \cdot |\{T' \in \mathcal{G}_\Delta(n - \ell n_1, n_1 - 1, d - \ell) \mid DS(T') \prec (n_{\ell+1}, \ldots, n_{\deg(r)})\}|.$$

□

Based on Lemma 16, we propose an algorithm to obtain the number of the descendant sequence predecessors of a given rooted tree. We assume that the numbers $g_\Delta(i, j, k)$, $1 \leq i \leq n$, $0 \leq j \leq m$, and $0 \leq k \leq \Delta$ are given. For four integers $n \geq 1$, $\Delta \geq 2$, $m \in [0, n-1]$ and $d \in [0, \min\{\Delta, n-1\}]$, let $T = (G, r)$ be a rooted tree such that its canonical tree is in $\mathcal{G}_\Delta(n, m, d)$, and let $(n_1, n_2, \ldots, n_{\deg(r)})$ denote the descendant sequence of T. In order to obtain the number of the descendant sequence predecessors of T, we calculate the terms in Lemma 16 one by one. The number $g_\Delta(n, n_1 - 1, d)$ in Equations 1 is given as input. We can find the smallest integer ℓ such that $n_\ell > n_{\ell+1}$ in $\mathcal{O}(\deg(r))$ time, and we can obtain the number

$\sum_{i=1}^{\ell-1} \mathrm{C_r}(g_\Delta(n_1), i) \cdot g_\Delta(n - in_1, n_1 - 1, d - i)$ in $\mathcal{O}(\ell)$ time. Then we calculate the number $|\{T' \in \mathcal{G}_\Delta(n - \ell n_1, n_1 - 1, d - \ell) \mid DS(T') \prec (n_{\ell+1}, \ldots, n_{\deg(r)})\}|$, recursively. Then we calculate the sum of them, and we have the number of descendant sequence predecessors of T. We show a description of the algorithm to obtain the number of the descendant sequence predecessors of a given tree in Algorithm 11, and we have the following lemma.

Algorithm 11 Calculate the number of the descendant sequence predecessors

Input: Four integers $n \geq 1$, $\Delta \geq 2$, $m \in [0, n-1]$ and $d \in [0, \min\{\Delta, n-1\}]$, a descendant sequence $DS(T) = (n_1, n_2, \ldots, n_{\deg(r)})$ of a rooted tree $T = (G, r)$ such that its canonical tree is in $\mathcal{G}_\Delta(n, m, d)$, and the values of $g_\Delta(i, j, k)$, $1 \leq i \leq n$, $0 \leq j \leq m$, and $0 \leq k \leq \Delta$.

Output: The number of descendant sequence predecessors of T.

1: $r := g_\Delta(n, n_1 - 1, d)$; /* Eq. (1) */
2: $\ell :=$ the smallest integer such that $n_\ell > n_{\ell+1}$;
3: $r := r + \sum_{i=1}^{\ell-1} \mathrm{C_r}(g_\Delta(n_1), i) \cdot g_\Delta(n - in_1, n_1 - 1, d - i)$; /* Eq. (2) */
4: **if** $\ell < \deg(r)$ **then**
5: $\quad D := (n_{\ell+1}, n_{\ell+2}, \ldots, n_{\deg(r)})$;
6: $\quad r := r + \mathrm{C_r}(g_\Delta(n_1), \ell) \cdot$ **Algorithm11**$(n - \ell n_1, \Delta, n_1 - 1, d - \ell, D, g_\Delta)$;/* Eq. (3) */
7: **end if**;
8: **return** r

Lemma 17. *Let $n \geq 1$, $\Delta \geq 2$, $m \in [0, n-1]$ and $d \in [0, \min\{\Delta, n-1\}]$ be four integers, and let T be a rooted tree such that its canonical tree is in $\mathcal{G}_\Delta(n, m, d)$. Assume that the descendant sequence $DS(T)$ of T and the values of $g_\Delta(i, j, k)$, $1 \leq i \leq n$, $0 \leq j \leq m$, and $0 \leq k \leq \Delta$ are given. Then the number of descendant predecessors of T can be obtained in $\mathcal{O}(d)$ time.*

Proof. Let $f(d)$ denote the total time complexity to calculate the equation in Lemma 16. Equations 1 and 2 of Lemma 16 can be calculated in $\mathcal{O}(1)$ and $\mathcal{O}(\ell)$ time respectively. Equation 3 of Lemma 16 can be calculated in $f(d - \ell)$ time. Consequently, for some constant c_1 and c_2, we have $f(d) = c_1 + c_2 \cdot \ell + f(d - \ell)$. We show $f(d)$ can be bounded by $\mathcal{O}(d)$ by induction. Assume that for any $d' < d$, $f(d') = c \cdot d'$ holds with some constant c with $c_1 + (c_2 - c) \cdot \ell < 0$. Then, we have

$$\begin{aligned} f(d) &= c_1 + c_2 \cdot \ell + f(d - \ell) \\ &= c_1 + c_2 \cdot \ell + c \cdot (d - \ell) \\ &= c_1 + (c_2 - c) \cdot \ell + c \cdot d \end{aligned}$$

$$< \quad cd.$$

Therefore we have $\mathcal{O}(f(d)) = \mathcal{O}(d)$. □

5.2 The Number of Rank Sequence Predecessors

In this subsection, we devise a method to obtain the number of rank sequence predecessors. Since the rank sequence can be regarded as an upper bounded sequence, the problem to find the number of the rank sequence predecessors is equivalent to the ranking of upper bounded sequences. Therefore from Theorem 3, we obtain the following lemma.

Lemma 18. *Let $n \geq 1$ and $\Delta \geq 2$ be two positive integers, and let $T = (G, r)$ be a rooted tree such that its canonical tree is in $\mathcal{G}_\Delta(n)$. Assume that the descendant sequence $DS(T)$ and the rank sequence $RS(T)$, and the values of $g_\Delta(i), 1 \leq i \leq n$ are given. Then the number of rank sequence predecessors of T can be obtained in $\mathcal{O}(\deg(r)^2)$ time.*

Proof. Let $(n_1, n_2, \ldots, n_{\deg(r)})$ denote the descendant sequence of T, and let K denote the sequence $(g_\Delta(n_1), g_\Delta(n_2), \ldots, g_\Delta(n_{\deg(r)}))$. Let $I_1, I_2, \ldots, I_\ell$ be the uniform decomposition of K, and ℓ_i denote the length of I_i for each $i \in [1, \ell]$. From Theorem 3, the rank of the sequence $RS(T)$ in $\mathcal{S}(K)$ can be done in $\mathcal{O}(\sum_{i=1}^{\ell} \ell_i^2)$. From the property of uniform decomposition, we have $\sum_{i=1}^{\ell} \ell_i = |DS(T)| = \deg(r)$. Since $\sum_{i=1}^{\ell} \ell_i^2 < (\sum_{i=1}^{\ell} \ell_i)^2 = \deg(r)^2$ holds, the number of rank sequence predecessors of T can be obtained in $\mathcal{O}(\deg(r)^2)$ time. □

Now we give an algorithm to obtain the rank of a rooted tree in Algorithm 12. Let $n \geq 1$ and $\Delta \geq 2$ be two positive integers, and let T be a rooted tree such that its canonical tree is in $\mathcal{G}_\Delta(n)$. The vertex v is traversed according to the BFS Post-order. The descendant sequence and the rank sequence of the subtree $T(v)$ can be obtained by sorting the pair of $(|V(T(v))|, \mathrm{rank}_\Delta(T(v)))$ for all subtrees rooted at a child of v. The number of the descendant sequence predecessors and the rank sequence predecessors can be found by Algorithms 11 and 3, and we obtain the rank of $T(v)$ by adding $|\mathcal{D}_\Delta(T)|$ and $|\mathcal{R}_\Delta(T)| + 1$. In this algorithm, the variable rank[v] store the rank of subtree $T(v)$ in $\mathcal{G}_\Delta(|V(T(v))|)$. As a result, we have the following theorem.

Theorem 10. *Let $n \geq 1$ and $\Delta \geq 2$ be two positive integers, and let $T = (G, r)$ be a rooted tree such that its canonical tree is in $\mathcal{G}_\Delta(n)$. Assume that the values of $g_\Delta(i, j, k)$, $1 \leq i \leq n$, $0 \leq j \leq \min\{m, i-1\}$, and $0 \leq k \leq \min\{\Delta, i-1\}$ are given. Then the rank of the canonical tree of T in $\mathcal{G}_\Delta(n)$ can be obtained in $\mathcal{O}(\min\{n\Delta^2, n^2\})$ time.*

Proof. For a vertex v in $V(T)$, given the number of descendant (resp., rank) for each subtree rooted at a child of v, the descendant sequence and the rank sequence of the

Algorithm 12 Ranking Algorithm of $\mathcal{G}_\Delta(n)$

Input: Two positive integers $n \geq 1$ and $\Delta \geq 2$, a rooted tree $T = (G, r)$ such that its canonical tree is in $\mathcal{G}_\Delta(n)$, and the values of $g_\Delta(i, j, k)$, $1 \leq i \leq n$, $0 \leq j \leq \min\{m, i-1\}$, and $0 \leq k \leq \min\{\Delta, i-1\}$.

Output: The rank of T in $\mathcal{G}_\Delta(n)$.

1: **for** $v \in V(T)$ in BFS Post-order **do**
2: **if** v is a leaf of T **then**
3: rank$[v] := 1$
4: **else**
5: Calculate $DS(T(v))$ by sorting the number of vertices in all subtrees rooted at a child of v;
6: Calculate $RS(T(v))$ by sorting the rank of all subtrees rooted at a child of v;
7: $d :=$ **Algorithm 11**$(n, \Delta, n-1, \Delta-1, DS(T(v)), g_\Delta)$;
8: /* $(n_1, n_2, \ldots, n_\ell) := DS(T(v)); K := (g_\Delta(n_1), g_\Delta(n_2), \ldots, g_\Delta(n_\ell))$ */
9: $r :=$ **Algorithm 3**$(K, RS(T(v)))$;
10: rank$[v] := d + r + 1$
11: **end if**
12: **end for**;
13: output rank$[r]$

subtree $T(v)$ can be obtained by sorting the pair of $(|V(T(v))|, \mathrm{rank}_\Delta(T(v)))$ for all subtrees rooted at a child of v. Hence the descendant sequence and rank sequence of $V(T)$ can be obtained in $\mathcal{O}(\deg(v) \log \deg(v))$ time if the number of descendant and the rank for each subtree rooted at a child of v are given. Since the descendant and the rank of leaves is obvious, by Lemmas 17 and 18, for each $v \in V(T)$, the rank of subtree $T(v)$ can be identified in $\mathcal{O}(\deg(v) \log \deg(v) + \deg(v) + \deg(v)^2) = \mathcal{O}(\deg(v)^2)$ by processing the BFS Post-order. Therefore the rank of a tree in $\mathcal{G}_\Delta(n)$ can be obtained in $\mathcal{O}(\sum_{v \in V(T)} \deg(v)^2) = \mathcal{O}(\min\{n\Delta^2, n^2\})$ time since we have $\sum_{v \in V(T)} \deg(v)^2 \leq (\sum_{v \in V(T)} \deg(r))^2 = (2(n-1))^2$. □

In order to identify the rank of a given rooted tree, we showed a bottom-up algorithm in Algorithm 12. In this algorithm, we calculate the number of vertices and the rank for all subtrees. This implies that the algorithm can identify the descendant representation and the rank representation of the canonical tree of a given rooted tree from the structure of the given rooted tree. Therefore we have the following corollary.

Corollary 1. *Let $n \geq 1$ and $\Delta \geq 2$ be two positive integers, and let $T = (G, r)$ be a rooted tree with n vertices such that the maximum degree is at most Δ, and the degree of r is at most $\Delta - 1$. Assume that the values of $g_\Delta(i, j, k)$, $1 \leq i \leq n$, $0 \leq j \leq$*

$\min\{m, i-1\}$, *and* $0 \leq k \leq \min\{\Delta, i-1\}$ *are given. Then the canonical tree of* T *can be obtained in* $\mathcal{O}(\min\{n\Delta^2, n^2\})$ *time.*

6 Unranking for Degree Bounded Rooted Trees

In this section, for three positive integers $n \geq 1$, $\Delta \geq 2$, and $k \in [1, g_\Delta(n)]$, we devise an algorithm to generate the k-th canonical tree in $\mathcal{G}_\Delta(n)$ following the lexicographical ascending order of descendant representation σ_{des}. We say that the problem to generate the k-th canonical tree in $\mathcal{G}_\Delta(n)$ is unranking problem. In the ranking problem, we proposed a bottom-up algorithm which calculats the rank of a tree from the descendant sequence and the rank sequence. Conversely, in the unranking problem, we propose a top-down algorithm, that is, we derive the descendant sequence and the rank sequence from it's rank. From the descendant sequence and the rank sequence, we know the rank of each root-subtree. In order to obtain the whole structure of k-th canonical tree, we apply the unranking algorithm to each subtree recursively.

In subsection 6.1, we explain how to find the descendant sequence of the k-th canonical tree in $\mathcal{G}_\Delta(n)$ from the integer k. Similarly, in subsection 6.2, we show how to obtain the rank sequence of the k-th canonical tree in $\mathcal{G}_\Delta(n)$ from the integer k.

6.1 Unranking for Descendant Sequence

For five integers $n \geq 1$, $\Delta \geq 2$, $m \in [0, n-1]$, $d \in [0, \min\{\Delta, n-1\}]$, and $k \in [1, g_\Delta(n, m, d)]$, we devise an algorithm to identify the descendant sequence of the k-th canonical tree in $\mathcal{G}_\Delta(n, m, d)$ following the lexicographical ascending order of descendant representation. From Lemma 16, we have the following Lemma.

Lemma 19. *Let $n \geq 1$, $\Delta \geq 2$, $m \in [0, n-1]$ and $d \in [0, \min\{\Delta, n-1\}]$ be four integers, and let T be a canonical tree in $\mathcal{G}_\Delta(n, m, d)$. Then:*

(i) *If $g_\Delta(n, j-1, d) < \mathrm{rank}_\Delta(T) \leq g_\Delta(n, j, d)$ holds, then the maximum root-subtree has exactly j vertices;*

(ii) *If $g_\Delta(n, j-1, d) + \sum_{k=1}^{\ell-1} \mathrm{C_r}(g_\Delta(j), k) \cdot g_\Delta(n-jk, j-1, d-k) < \mathrm{rank}_\Delta(T) \leq g_\Delta(n, j-1, d) + \sum_{k=1}^{\ell} \mathrm{C_r}(g_\Delta(j), k) \cdot g_\Delta(n-jk, j-1, d-k)$ holds, then the rooted tree T has ℓ root-subtrees whose number of vertices is exactly j; and*

(iii) *If the maximum root-subtree of T has exactly j vertices, and T has ℓ root-subtrees whose number of vertices is j, then let $k = \mathrm{rank}_\Delta(T) - g_\Delta(n, j-1, d) - \sum_{k=1}^{\ell-1} \mathrm{C_r}(g_\Delta(j), k) \cdot g_\Delta(n-jk, j-1, d-k)$, $k' = \lfloor k / \mathrm{C_r}(g_\Delta(j), \ell) \rfloor$, let T_{res} be the rooted tree obtained by removing all j-vertices subtrees in T, and the descendant sequence of T_{res} is equal to the descendant sequence of the k'-th tree in $\mathcal{G}_\Delta(n-j\ell, j-1, d-\ell)$.*

Proof. (i) Given two canonical trees T and T' in $\mathcal{G}_\Delta(n)$, if $DS(T) \prec DS(T')$ holds, then we have $L_D(T) \prec L_D(T')$. Hence for any trees $T \in \mathcal{G}_\Delta(n, j-1, d)$ and a tree $T' \in \mathcal{G}_\Delta(n, j, d) \setminus \mathcal{G}_\Delta(n, j-1, d)$, it holds that $L_D(T) \prec L_D(T')$ since the first entry of

$DS(T)$ is less than j and the first entry of $DS(T')$ is j. Therefore $g_\Delta(n, j-1, d) < \mathrm{rank}_\Delta(T) \leq g_\Delta(n, j, d)$ implies that the maximum root-subtree of T has exactly j vertices.

(ii) From the equation 2 in Lemma 16, the number of canonical trees in $\mathcal{G}_\Delta(n, m, d)$ such that at most ℓ root-subtrees whose vertices are exactly j is $\sum_{k=1}^{\ell} \mathrm{C_r}(g_\Delta(j), k) \cdot g_\Delta(n - jk, j-1, d-k)$. Let T (resp., T') be a canonical tree in $\mathcal{G}_\Delta(n, m, d)$ such that there are k (resp., k') root-subtrees whose vertices are exactly j respectively. If $k < k'$ holds, then we have $L_D(T) \prec L_D(T')$ since the first k (resp., k') entries of the descendant sequence of T (resp., T') is j, respectively.

From Lemma 19-(i), T has at least one exactly j vertices root-subtree since $g_\Delta(n, j-1, d) < \mathrm{rank}_\Delta(T) \leq g_\Delta(n, j, d)$ holds. Let T (resp., T') be a canonical tree in $\mathcal{G}_\Delta(n, j, d)$ and ℓ (resp., ℓ') be the number of j vertices root-subtree on T (resp., T'). If $\ell < \ell'$ holds, then we obtain $DS(T) \prec DS(T')$ and $L_D(T) \prec L_D(T')$ since the first ℓ (resp., ℓ') entries of the descendant sequence of T (resp., T') is j.

From Lemma 14, the number of rooted trees such that the number of exactly j vertices root-subtrees is ℓ is $\mathrm{C_r}(g_\Delta(j), \ell) \cdot g_\Delta(n - jk, j-1, d-\ell)$. Therefore if $g_\Delta(n, j-1, d) + \sum_{k=1}^{\ell-1} \mathrm{C_r}(g_\Delta(j), k) \cdot g_\Delta(n - jk, j-1, d-k) < \mathrm{rank}_\Delta(T) \leq g_\Delta(n, j-1, d) + \sum_{k=1}^{\ell} \mathrm{C_r}(g_\Delta(j), k) \cdot g_\Delta(n - jk, j-1, d-k)$ holds, then T has ℓ root-subtrees whose vertices are exactly j.

(iii) The number of vertices in T_{res} is $n - j\ell$ and the degree of root must be at most $d - \ell$. In addition, each root-subtrees in T_{res} has at most $j-1$ vertices. Therefore T_{res} belongs to $\mathcal{G}_\Delta(n-j\ell, j-1, d-\ell)$. From Lemma 14, for each tree in $\mathcal{G}_\Delta(n-j\ell, j-1, d-\ell)$, we obtain a rooted tree by adding ℓ repetitions of exactly j vertices root-subtrees. The number of variation of attaching such ℓ root-subtrees is $\mathrm{C_r}(g_\Delta(j), \ell)$. Let T'_{res} and $\hat{T}_{\mathrm{res}}$ be canonical trees in $\mathcal{G}_\Delta(n - j\ell, j-1, d-\ell)$ such that $L_D(T'_{\mathrm{res}}) \prec L_D(\hat{T}_{\mathrm{res}})$. The trees T' and $\hat{T}$ obtained by attached any ℓ repetitions of exactly j vertices subtree to T'_{res} and $\hat{T}_{\mathrm{res}}$ respectively satisfy $L_D(T') \prec L_D(\hat{T})$. Note that, if T'_{res} and $\hat{T}_{\mathrm{res}}$ has the same descendant sequence, then this may not hold. Consequently, $k' = \lfloor k / \mathrm{C_r}(g_\Delta(j), \ell) \rfloor$ indicates the rank of a canonical tree in $\mathcal{G}_\Delta(n - j\ell, j-1, d-\ell)$, which has the same descendant sequence as T_{res}. □

From Lemma 19, we show an algorithm to construct the descendant sequence of k-th tree in $\mathcal{G}_\Delta(n, m, d)$. First we find the maximum number j of vertices in a root-subtree by Lemma 19-(i), then we find the number of repetitions of the root-subtrees with j vertices by Lemma 19-(ii). After that, we find the descendant sequence D' of the tree obtained by deleting all root-subtrees with j vertices by Lemma 19-(iii). The concatenation of the sequence $(j, j, \ldots, j)$ with length ℓ and D' is the descendant sequence of k-th tree in $\mathcal{G}_\Delta(n, m, d)$. We show a description of the algorithm in Algorithm 13 and we have the following lemma.

Algorithm 13 Constraction of the Descendant Sequence of k-th canonical tree in $\mathcal{G}_\Delta(n,m,d)$

Input: Five positive integers $n \geq 1$, $\Delta \geq 2$, $m \in [0, n-1]$, $d \in [0, \min\{\Delta, n-1\}]$, and $1 \leq k \leq g_\Delta(n,m,d)$, and the values of $g_\Delta(i,j,k)$, $1 \leq i \leq n$, $0 \leq j \leq \min\{m, i-1\}$, and $0 \leq k \leq \min\{\Delta, i-1\}$.

Output: The descendant sequence of the k-th canonical tree in $g_\Delta(n,m,d)$.

1: Calculate the maximum number j of vertices in a root-subtree by Lemma 19-(i);
2: Calculate the number ℓ of repetitions of the root-subtrees with j vertices by Lemma 19-(ii);
3: $k := k - g_\Delta(n, j-1, d) - \sum_{i=1}^{\ell-1} \mathrm{C_r}(g_\Delta(j), i) \cdot g_\Delta(n-ji, j-1, d-i)$;
4: $k' := \lfloor k / \mathrm{C_r}(g_\Delta(j), \ell) \rfloor$;
5: $D' :=$**Algorithm 13**$(n - j\ell, j-1, d-\ell, k', g_\Delta)$;
6: /*$I(\ell, j)$ is the uniform sequence with length ℓ and value j */
7: **output** the concatenation sequence of $I(\ell, j)$ and D'

Lemma 20. *Let $n \geq 1$, $\Delta \geq 2$, $m \in [0, n-1]$, $d \in [0, \min\{\Delta, n-1\}]$, and $k \in [1, g_\Delta(n,m,d)]$ be five integers. Assume that the values of $g_\Delta(i,j,k)$, $1 \leq i \leq n$, $0 \leq j \leq \min\{m, i-1\}$, and $0 \leq k \leq \min\{\Delta, i-1\}$ are given. Then the descendant sequence of the k-th canonical tree in $\mathcal{G}_\Delta(n,m,d)$ can be obtained in $\mathcal{O}(d \log n)$ time.*

Proof. From Lemma 19-(i), we find the maximum number j of vertices in a root-subtree in k-th canonical tree by using binary search, and it can be done in $\mathcal{O}(\log n)$ since $m < n$ holds. From Lemma 19-(ii), the number of repetition of root-subtrees with j vertices can be obtained in $\mathcal{O}(d)$ time. Let $f(d)$ denote the time complexity of Algorithm 13. We have $f(d) = \mathcal{O}(\log n) + \mathcal{O}(\ell) + f(d-\ell) = \mathcal{O}(d \log n)$. □

6.2 Unranking for Rank Sequence

Given three positive integers $n \geq 1$, $\Delta \geq 2$, and $k \in [1, g_\Delta(n)]$, we derive the rank sequence of the k-th canonical tree in $\mathcal{G}_\Delta(n)$. From Lemma 20, we obtain the descendant sequence $(n_1, n_2, \ldots, n_{\deg(r)})$ of the k-th canonical tree in $\mathcal{G}_\Delta(n)$. For the sequence $K = (g_\Delta(n_1), g_\Delta(n_2), \ldots, g_\Delta(n_{\deg(r)}))$, the rank sequence of the k-th canonical tree T in $\mathcal{G}_\Delta(n)$ is the $(k - |\mathcal{D}_\Delta(T)|)$-th upper bounded sequence in $\mathcal{S}(K)$. Hence the rank sequence identification problem is equivalent to the unranking problem for upper bounded sequences. By Algorithm 4, the rank sequence of k-th canonical tree can be obtained.

Finally, we propose an algorithm for the unranking of $\mathcal{G}_\Delta(n)$. Given three positive integers $n \geq 1$, $\Delta \geq 2$, and $k \in [1, g_\Delta(n)]$, we find the descendant sequence of the k-th canonical tree in $\mathcal{G}_\Delta(n)$ by Lemma 19, then we find the rank sequence of the k-th tree $\mathcal{G}_\Delta(n)$ by unranking algorithm for upper bounded sequences. After that, we have the

number of vertices and the rank of each root-subtree of the k-th canonical tree, and we construct the structure of each root-subtree, recursively. We show a description of the unranking algorithm for $\mathcal{G}_\Delta(n)$ in Algorithm 14, and we have the following theorem.

Algorithm 14 Unranking Algorithm for $\mathcal{G}_\Delta(n)$

Input: Three positive integers $n \geq 1$, $\Delta \geq 2$, and $k \in [1, g_\Delta(n)]$, the values of $g_\Delta(i,j,k)$, $1 \leq i \leq n$, $0 \leq j \leq \min\{m, i-1\}$, and $0 \leq k \leq \min\{\Delta, i-1\}$.

Output: The descendant representation of k-th tree T in $\mathcal{G}_\Delta(n)$.

1: Calculate the descendant sequence $D = (n_1, n_2, \ldots, n_\ell)$ of k-th tree;
2: Calculate the rank sequence $R = (r_1, r_2, \ldots, r_\ell)$ of k-th tree;
3: **for** $i = 1, 2, \ldots, \ell$ **do**
4: $\quad c_i :=$**Algorithm 14**$(n_i, \Delta, r_i, g_\Delta)$;
5: $\quad$ Delete the first entry from c_i
6: **end for**
7: **return** $(n, D, c_1, c_2, \ldots, c_\ell)$

Theorem 11. *Let $n \geq 1$, $\Delta \geq 2$, and $k \in [1, g_\Delta(n)]$ be three positive integers. Assume that the values of $g_\Delta(i,j,k)$, $1 \leq i \leq n$, $0 \leq j \leq \min\{m, i-1\}$, and $0 \leq k \leq \min\{\Delta, i-1\}$ are given. Then the k-th canonical tree in $\mathcal{G}_\Delta(n)$ can be constructed in $\mathcal{O}(\min\{n^2\Delta^2 \log \Delta, n^3 \log \Delta\})$ time.*

Proof. Let T be the k-th canonical tree in $\mathcal{G}_\Delta(n)$ rooted at a vertex r. From Lemma 20, the descendant sequence $(n_1, n_2, \ldots, n_{\deg(r)})$ of T can be obtained in $\mathcal{O}(\deg(r) \log n)$ time from only the integer k.

Let K denote the sequence $(g_\Delta(n_1), g_\Delta(n_2), \ldots, g_\Delta(n_{\deg(r)}))$. From Lemma 13 and the equation $k = |\mathcal{D}_\Delta(T)| + |\mathcal{R}_\Delta(T)| + 1$, we see that the rank sequence of k-th canonical tree in $\mathcal{G}_\Delta(n)$ is the $(k - |\mathcal{D}_\Delta(T)|)$-th sequence in $\mathcal{S}(K)$. From Lemma 17, the number of descendant predecessors $|\mathcal{D}_\Delta(T)|$ of T can be obtained in $\mathcal{O}(\deg(r))$. The rank sequence of T can be derived from the integer k by the unranking of $\mathcal{S}(K)$.

Let $I_1, I_2, \ldots, I_\ell$ denote the uniform decomposition of K, and let ℓ_i (resp., m_i) denote the length (resp., value) of I_i for each $i \in [1, \ell]$, respectively. Note that an ordered tree with n vertices and the maximum degree at most Δ can be uniquely represented by a sequence with length n such that each entry is in $[1, \Delta]$ (i.e. for an ordered tree, a sequence obtained by arranging degree for each vertex in DFS-ordering can be a unique representation of ordered trees, and if the maximum degree of the ordered tree can be bounded by Δ, then each entry of the sequence is bounded by Δ.) As a result, the number of rooted trees in $g_\Delta(n_i)$ can be bounded by Δ^{n_i}, and we see that m_i can be bounded by Δ^{n_i}. Hence from Theorem 4, the rank sequence of k-th tree can be obtained in $\mathcal{O}(\sum_{i=1}^{\ell} \ell_i^2 \log(\ell_i + m_i)) = \mathcal{O}(n \deg(r)^2 \log \Delta)$ time.

In order to obtain whole structure of k-th canonical tree in $\mathcal{G}_\Delta(n)$, we apply the identification of the descendant sequence and the rank sequence for each root-subtrees, recursively. From Lemma 20 and Theorem 4, for each $v \in V(T)$, the identification of the descendant sequence and the rank sequence for each subtree can be done in $\mathcal{O}(\deg(v) \log n + n \deg(v)^2 \log \Delta)$. Since we have $\deg(v) \leq \Delta$ and $\sum_{v \in V(T)} \deg(v) = 2(n-1)$, it holds that $\sum_{v \in V(T)} \deg(v) \log n = n \log n$ and $\sum_{v \in V(T)} n \deg(v)^2 \log \Delta \leq \min\{n^2 \Delta^2 \log \Delta, n^3 \log \Delta\}$. Hence we have

$$\begin{aligned} & \mathcal{O}(\sum_{v \in V(T)} \deg(v) \log n + \sum_{v \in V(T)} n \deg(v)^2 \log \deg(v)) \\ = \; & \mathcal{O}(n \log n + \min\{n^2 \Delta^2 \log \Delta, n^3 \log \Delta\}) \\ = \; & \mathcal{O}(\min\{n^2 \Delta^2 \log \Delta, n^3 \log \Delta\}). \end{aligned} \tag{4}$$

As a result, the unranking of $\mathcal{G}_\Delta(n)$ can be done in $\mathcal{O}(\min\{n^2 \Delta^2 \log \Delta, n^3 \log \Delta\})$ time. □

7 Experimental Results

In order to examine the practical performance of our enumeration algorithm, we implemented our sequential enumeration algorithm, and we evaluated the practical time to enumerate all non-isomorphic unlabeled trees with/without the degree bound. Our algorithm is implemented in the C++ language, and compiled using `gcc` version 4.8.4, with `-O3` optimization. All experiments were run on a PC with Intel(R) Xeon(R) CPU E5-1660 v3 @ 3.00GHz and 32 GB memory.

First we conduct an experiment for enumerating all non-isomorphic unlabeled trees without degree bound ($\Delta = n-1$). We show the number of trees, the time for counting phase, the time for enumerating phase, the total time, and the average time to generate one tree in Table 1. We set the time limit to 600 seconds. If the computation time is over 600 seconds, then we indicate by T.O. (for "time out"). From Table 1, we observe that our implementation of the algorithm can generate all trees with up to 27 vertices in 600 seconds. We see that the average time to generate one tree will be increasing as the number of vertices increase. This result implies that the time complexity of this algorithm is not constant time per tree.

There are two production programs Molgen [3] and Enumol to enumerate chemical graphs. Both Molgen and Enumol can enumerate all alkanes with n carbons, that is, the non-isomorphic unlabeled unrooted trees with n vertices and the maximum degree at most 4. In order to confirm an advantage of computation time, we compare the computation time to generate all trees in $\mathcal{F}_4(n)$ by Molgen, Enumol, and our proposed algorithm. The summary of the computation times is shown in Table 2. The time limit is again set to be 600 seconds. From Table 2, Molgen can enumerate all alkanes up to 23 carbons, Enumol can generate all alkanes up to 27 carbons, and proposed algorithm can enumerate all alkanes up to 30 carbons in 600 seconds. We observe that our algorithm has the shortest running time. In addition, we observe that if the number of vertices is even, then the time for generating the predecessor tree is in general slightly larger than in the odd case. This can be attributed to the fact that if n is even, then some trees have a bicentroid root, and the algorithm does some extra processing in this case.

Table 1: Computation time for enumerating all trees on n vertices without a degree bound.

n	$\lvert\mathcal{F}_{n-1}(n)\rvert$	counting time[s]	enumerating time[s]	total time[s]	average time[s]
21	2144505	1.25×10^{-4}	0.68	0.68	3.19×10^{-7}
22	5623756	1.31×10^{-4}	2.05	2.05	3.64×10^{-7}
23	14828074	1.54×10^{-4}	5.52	5.52	3.72×10^{-7}
24	39299897	1.81×10^{-4}	16.28	16.28	4.14×10^{-7}
25	104636890	1.84×10^{-4}	44.78	44.78	4.28×10^{-7}
26	279793450	2.37×10^{-4}	129.84	129.84	4.64×10^{-7}
27	751065460	2.49×10^{-4}	356.25	356.25	4.74×10^{-7}
28	2023443032	2.94×10^{-4}	T.O.*	T.O.	T.O.

*T.O. stands for "Time Out," set to be 600 seconds in our experiments.

Table 2: Computation time for enumerating all trees on n vertices with maximum degree bound $\Delta = 4$.

n	$\lvert\mathcal{F}_4(n)\rvert$	Molgen[s]	Enumol[s]	Proposed			
				counting time[s]	enumerating time[s]	total time[s]	average time[s]
21	910726	44.69	0.73	0.85×10^{-4}	0.07	0.07	8.55×10^{-8}
22	2278658	110.51	1.99	0.67×10^{-4}	0.19	0.19	8.61×10^{-8}
23	5731580	281.35	4.91	0.88×10^{-4}	0.47	0.47	8.20×10^{-8}
24	14490245	T.O.*	13.28	0.81×10^{-4}	1.31	1.31	9.06×10^{-8}
25	36797588	T.O.	33.19	0.87×10^{-4}	3.38	3.38	9.19×10^{-8}
26	93839412	T.O.	91.07	0.97×10^{-4}	8.83	8.83	9.41×10^{-8}
27	240215803	T.O.	229.95	1.15×10^{-4}	21.48	21.48	8.94×10^{-8}
28	617105614	T.O.	T.O.	1.07×10^{-4}	58.48	58.48	9.47×10^{-8}
29	1590507121	T.O.	T.O.	1.52×10^{-4}	144.19	144.19	9.06×10^{-8}
30	4111846763	T.O.	T.O.	1.52×10^{-4}	387.37	387.37	9.42×10^{-8}
31	10660307791	T.O.	T.O.	1.72×10^{-4}	T.O.	T.O.	T.O.

*T.O. stands for "Time Out," set to be 600 seconds in our experiments.

8 Conclusion

In this paper, we defined an order of trees and we designed a scheme for generating all trees according to this order. This scheme consists of two phases, a counting phase and an enumerating phase. The counting phase calculates the number of rooted trees with n vertices such that the maximum degree bound is Δ and the degree of the root vertex is at most $\Delta - 1$ by dynamic programming. The enumerate phase generates all trees according to the tree ordering by using the information from the counting phase. After $\mathcal{O}(n^2\Delta^2)$ time pre-processing, the proposed algorithm can enumerate all non-rooted isomorphic trees on n vertices whose maximum degree at most Δ in $\mathcal{O}(n)$ time per tree and $\mathcal{O}(n^2\Delta)$ space in total.

In addition, we propose a ranking algorithm and an unranking algorithm. After $\mathcal{O}(n^2\Delta^2)$ time pre-processing, our ranking algorithm can obtain the rank of a tree in $\mathcal{G}_\Delta(n)$ in $\mathcal{O}(\min\{n\Delta^2, n^2\})$ time, and our unranking algorithm can generate the tree of any given rank in $\mathcal{O}(\min\{n^2\Delta^2 \log \Delta, n^3 \log \Delta\}$ time.

For the future work, the following three problems are open. First we are interested in an enumeration algorithm with constant time per tree. There exists a family-tree based algorithm for enumeration unlabeled trees in constant time per tree [20]. However, we only proposed an algorithm for enumeration trees in $\mathcal{O}(n)$ time per tree. There may exist an algorithm for enumeration trees in the lexicographical ascending order of descendant representation in constant time per tree.

Second the time complexities of sequential enumeration, ranking and unranking depend on the order of trees. For a tree order which is different from the lexicographical ascending order of descendant representation, we construct algorithms for sequential enumeration, ranking, and unranking of trees. Then the time complexity of the algorithms following a different tree order may be different from the time complexity of the algorithms we proposed. Therefore finding an order of trees to achieve better time complexity for sequential enumeration, ranking, and unranking is one of the open problems.

Third we would like to design a sequential enumeration, ranking, and unranking algorithms for more general graphs. Enumeration of chemical compounds is one of the applications of graph enumeration. We can represent a chemical compound as a graph with the vertex color and the edge multiplicity. Hence the enumeration, ranking, and unranking for trees with the vertex color and the multiple edges is an important problem. In addition, we would like to generate graphs with cycles. For example, we devise the enumeration for graphs with exactly one cycle. We regard a graph with one cycle as a cycle and rooted trees connected to the cycle. If we obtain the rank of trees connected to the cycle, then we can generate the graph by using unranking algorithm for trees. Hence all we have to do is to enumerate all possible combination of the rank

of trees connected to the cycle. This technique will be applied for not only graphs with one cycle but also graphs consisting of a special structure and rooted trees connected to the special structure. For instance, one of the special structures is a connected graph with three cycles such that the degree of each vertex is at least 2. Therefore based on our sequential enumeration and unranking, designing an algorithm for enumerating more general graphs is one of the open problems.

References

[1] Amani, M. and Nowzari-Dalini, A.: "Generation, ranking and unranking of ordered trees with degree bounds," arXiv preprint, arXiv:1603.00977 (2016).

[2] Aringhieri, R., Hansen, P., and Malucelli, F.: "Chemical trees enumeration algorithms," 4OR: A Quarterly Journal of Operations Research 1(1), pp. 67–83. (2003).

[3] Benecke, C., Grund, R., Hohberger, R., Kerber, A., Laue, R., and Wieland, T.: "Molgen+, a generator of connectivity isomers and stereoisomers for molecular structure elucidation," Analytica Chimica Acta 314(3), pp. 141–147 (1995).

[4] Beyer, T. and Hedetniemi, S.M.: "Constant time generation of rooted trees," SIAM Journal on Computing 9(4), pp. 706–712 (1980).

[5] Blum, L. C. and Reymond, J. L.: "970 million druglike small molecules for virtual screening in the chemical universe database gdb-13," Journal of the American Chemical Society 131(25), pp. 8732–8733 (2009).

[6] Fujiwara, H., Wang, J., Zhao, L., Nagamochi, H., and Akutsu, T.: "Enumerating treelike chemical graphs with given path frequency," Journal of Chemical Information and Modeling 48(7), pp. 1345–1357 (2008).

[7] Ishida, Y., Kato, Y., Zhao, L., Nagamochi, H., and Akutsu, T.: "Branch-and-bound algorithms for enumerating treelike chemical graphs with given path frequency using detachment-cut," Journal of chemical information and modeling 50(5), pp. 934–946 (2010).

[8] Jordan, C.: "Sur les assemblages de lignes," J. Reine Angew. Math 70(185), 81 (1869).

[9] Knuth, D. E.: "Generating All Combinations and Partitions," The Art of Computer Programming, Vol. 4, Fascicle 3, Addison-Wesley Professional, (2005).

[10] Liebehenschel, J.: "Ranking and unranking of lexicographically ordered words: An average-case analysis," Journal of Automata, Languages and Combinatorics, 2(4), pp. 227–268 (1997).

[11] Meringer, M. and Schymanski, E.L.: "Small molecule identification with molgen and mass spectrometry," Metabolites 3(2), pp. 440–462 (2013).

[12] Myrvold, W. and Ruskey, F.: "Ranking and unranking permutations in linear time," Information Processing Letters, 79(6), pp. 281–284 (2001).

[13] Nakano, S.I.: "Efficient generation of plane trees," Information Processing Letters 84(3), pp. 167–172 (2002).

[14] Peironcely, J.E., Rojas-Chertó, M., Fichera, D., Reijmers, T., Coulier, L., Faulon, J.L., and Hankemeier, T.: "OMG: Open Molecule Generator," Journal of cheminformatics 4(1), 21 (2012).

[15] Prüfer, H.: "Neuer Beweis eines Satzes über Permutationen," Archiv für Mathematik und Physik 27, pp. 142–144 (1918).

[16] Shimizu, M., Nagamochi, H., and Akutsu, T.: "Enumerating tree-like chemical graphs with given upper and lower bounds on path frequencies," BMC bioinformatics 12(14), S3 (2011).

[17] Shimizu, T., Fukunaga, T., and Nagamochi, H.: "Unranking of small combinations from large sets," Journal of Discrete Algorithms 29, pp. 8–20 (2014).

[18] Vogt, M. and Bajorath, J.: "Chemoinformatics: a view of the field and current trends in method development," Bioorganic & medicinal chemistry 20(18), pp. 5317–5323 (2012).

[19] Wright, R.A., Richmond, B., Odlyzko, A., and McKay, B.D.: "Constant time generation of free trees," SIAM Journal on Computing 15(2), pp. 540–548 (1986).

[20] Zhuang, B. and Nagamochi, H.: "Constant time generation of trees with degree bounds," 9th International Symposium on Operations Research and Its Applications, pp. 183–194 (2010).