Enumeration of Ordered Trees with Leaf Restrictions

Yasuaki Kobayashi ✉ 🆔
Faculty of Information Science and Technology, Hokkaido University, Sapporo, Japan

Dominik Köppl[1] ✉ 🏠 🆔
Department of Informatics, Yamanashi University, Japan

Yasuko Matsui ✉ 🆔
Department of Mathematical Sciences, Tokai University, Japan

Hirotaka Ono[1] ✉ 🏠 🆔
Department of Mathematical Informatics, Nagoya University, Japan

Toshiki Saitoh ✉ 🆔
School of Computer Science and Systems Engineering, Kyushu Institute of Technology, Fukuoka, Japan

Yushi Uno ✉
Graduate School of Informatics, Osaka Metropolitan University, Japan

──── **Abstract** ────

An α-ary tree for a constant $\alpha \geq 2$ is a rooted tree in which each node has at most α children. A node having no children is called a leaf. For a given rooted tree and a node v, the number of edges from the root to v is called the depth of v. We call a vector $\boldsymbol{w} = (w_1, w_2, \ldots, w_d)$ of nonnegative integers an (α-ary) distribution if there is an α-ary tree T such that the number of leaves at each depth $i \in [1..d]$ in T is w_i. Although not every vector of nonnegative integers is a distribution, a distribution can be associated with many α-ary trees. In this paper, we present an algorithm to enumerate all α-ary trees for a given distribution. Our algorithm reports the first tree in $O(d + \sum_{i=1}^{d} w_i)$ time, and then each subsequent α-ary tree in $O(\max_{i=1}^{d} w_i)$ time by representing each tree as the difference from the previous one. The algorithm can be restricted to computing all trees that are *full*, i.e., trees whose nodes have exactly α or no children.

---

[1] Corresponding author

1   Introduction

It is our pleasure to dedicate this work to Roberto Grossi, whose broad enthusiasm for algorithms and data structure has significantly shaped the landscape of enumeration algorithms [9]. Over the past decades, his contributions made significant impact on structural and algorithmic aspects of trees, strings, and graphs, inspiring both theoretical advances and practical applications.

In this spirit, we turn our attention to the enumeration of ordered trees, a topic rooted in combinatorics with strong connections to data structures, formal languages, and algorithmic analysis. Ordered trees play a fundamental role in various domains, hierarchical data representations, and the analysis of recursive algorithms are major examples. Enumerating ordered trees uncovers rich combinatorial patterns, with which we hope to offer a tiny tribute to Roberto's enduring influence, exploring a topic in line with his passion for enumerating combinatorial structures of mathematical beauty.

We here tackle a problem whose restricted version on binary trees has been posed in 2023 as an open problem presented at IWOCA 2023: The question is to efficiently enumerate all rooted full binary trees whose number of leaves at each depth is given as an input. This problem spawned from the research of prefix-free codes in coding theory.

Here, we tackle this enumeration problem in even more general settings: binary (not necessarily full) trees and α-ary trees of a fixed depth d for a constant $\alpha \geq 2$. For any of these tree types, our algorithm outputs the first solution in $O(d + n)$ time. Subsequently, it outputs the difference between two successive solutions in $O(m)$ time with constant delay, where n is the number of leaves and m is the maximum number of leaves at any depth. We here define *delay* as the time between the end of the previous output and the start of the subsequent output.

2   Related Work

With respect to the enumeration of rooted ordered trees, we are aware of an algorithm with constant delay to output all trees [3], or with restrictions on a specific diameter [12] or on a fixed number of internal nodes and leaves [17]. Given the number m of internal nodes, Zaks [18] gave an enumeration algorithm for all α-ary trees having m internal nodes. This algorithm reports the trees in a lexicographical order based on a bit encoding of the trees. Recently, an algorithm [5] enumerating AVL trees has been proposed. In addition, closed formulas have been studied for counting various properties in rooted ordered trees of unbounded degree [6, 7].

For the enumeration of full binary trees with n leaves, we propose a $2n$-bit encoding of a full binary tree. We use this encoding in the same way as Ruskey and Proskurowski [13] for the enumeration of binary trees by using Gray codes for enumerating bit encodings. Another bit encoding variant has been proposed by Baba et al. [1], who however used a depth-first encoding based on DFUDS [2], while our encoding is level-wise in the spirit of LOUDS [10].

3   Preliminaries

Given a fixed integer $\alpha \geq 2$, an α-*ary tree* is an ordered *rooted tree* whose nodes each have at most α children, which are ordered. In this paper, we only consider trees that have at least two nodes. Except for the root, the parent of each node is uniquely determined. There is a left-to-right ordering of the child nodes with respect to the parent. A node is called a *leaf* if it has no child, otherwise it is called an *internal* node. For a given rooted tree and a node v, the number of edges from the root to v is called the *depth* of v, which we denote by $\mathrm{depth}(v)$.

In particular, the depth of the root is zero. The *height* of the tree is the depth of the leaf with the largest depth. A *full α-ary tree* is an α-ary tree whose nodes are either leaves or internal nodes with α children. A special case is $\alpha = 2$, for which we name a 2-ary tree a *binary tree*.

In what follows, we start with the enumeration of full binary trees, which is the easiest case. Then we extend our techniques to arbitrary binary trees, and finally consider α-ary trees.

4 Properties of Binary Trees

Our input is a so-called *binary (tree leaf) distribution*, a term coined by Buro [4, Lemma 2.3]. It is defined as follows. We call an integer vector $\boldsymbol{w} = (w_1, \ldots, w_d)$ a *binary distribution* if there is a binary tree T with depth d such that T has w_i leaves with depth i for every $i \in [1..d]$. We say that the *binary (tree leaf) distribution* of T is $\boldsymbol{w}$ – there can be many trees that have the same binary distribution. A binary distribution is also called a *full-binary distribution* if there is a *full* binary tree with the same property. Buro [4, Thm. 2.4] showed that the full-binary distribution of a full binary tree with maximum path length is unique. Here we study the connection of arbitrary binary distributions and their corresponding trees. In detail, the first problem we want to tackle in this paper is as follows.

▶ **Problem 1.** *Given a (full-)binary distribution $\boldsymbol{w}$, enumerate all (full) binary trees whose (full-)binary distribution is $\boldsymbol{w}$.*

We first characterize the necessary conditions for an integer vector to be a (full) binary distribution. For that, we make use of Kraft's inequality [11].

▶ **Lemma 2** (Kraft's inequality)**.** *For a binary tree with leaves $v_1, \ldots, v_n$, it holds that $\sum_{i=1}^{n} 1/2^{\mathrm{depth}(v_i)} \leq 1$.*

Let $f(\boldsymbol{w}) = \sum_{i=1}^{d} w_i/2^i$ be the *weight* of a vector $\boldsymbol{w}$ of nonnegative integers. Then $f(\boldsymbol{w}) \leq 1$ is a necessary condition according to Kraft's inequality. We show that it is also sufficient.

▶ **Lemma 3.** *A vector of nonnegative integers $\boldsymbol{w}$ is a binary distribution if and only if $f(\boldsymbol{w}) \leq 1$.*

Proof by induction on the depth d. For $d = 1$, the binary tree can have only one or two leaves, i.e., $\boldsymbol{w} = (1)$ or $\boldsymbol{w} = (2)$, and in any case $w_1 \leq 2$ holds. For the induction hypothesis, assume that our claim holds for a depth d. Given a vector of nonnegative integers $\boldsymbol{w} = (w_1, \ldots, w_{d+1})$ with $f(\boldsymbol{w}) \leq 1$, we show that there exists a binary tree whose binary distribution is $\boldsymbol{w}$. For that, we define a vector $\boldsymbol{w}' = (w_1, \ldots, w_{d-1}, w'_d)$ and distinguish the following two cases.

- If w_{d+1} is even, we set $w'_d = w_d + w_{d+1}/2$, and thus $f(\boldsymbol{w}) = f(\boldsymbol{w}')$. By the induction hypothesis, there is a binary tree T' whose binary distribution is $\boldsymbol{w}'$. If we expand $w_{d+1}/2$ leaves with depth d in T to internal nodes with two leaves, we obtain a binary tree whose binary distribution in $\boldsymbol{w}$.

- If w_{d+1} is odd, we rewrite the inequality $1 \geq f(\boldsymbol{w}) = \sum_{i=1}^{d+1} w_i 2^{-i}$ as $2^{d+1} \geq \sum_{i=1}^{d+1} w_i 2^{d+1-i}$. Observing that the right summation only consists of even terms except for w_{d+1}, the left-hand side is even while the right-hand side is odd. This allows us to increment the right-hand side by one while still retaining validity, i.e., as $2^{d+1} \geq \sum_{i=1}^{d} w_i 2^{d+1-i} + w_{d+1} + 1$. We therefore can apply the even case analysis for the vector $(w_1, \ldots, w_d, w_{d+1} + 1)$. ◀

▶ **Lemma 4.** *A vector $\boldsymbol{w} = (w_1, \ldots, w_d)$ with nonnegative integers and $w_d > 0$ is the full-binary distribution of a full binary tree if and only if $f(\boldsymbol{w}) = 1$.*

Proof. We prove both directions by induction, for which we use $d = 1$ as the starting point. For depth 1, on the one hand, there is only one full binary tree with the root having two leaves as children. On the other hand, there is only one vector $\boldsymbol{w} = (w_1) = (2)$ with $f(\boldsymbol{w}) = 1$, and therefore the claim holds. For the induction hypothesis, let us assume that the claim holds for d.

Direction $\Rightarrow$. Consider a full binary tree T whose full-binary distribution is $\boldsymbol{w} = (w_1, \ldots, w_{d+1})$. We can group the w_{d+1} leaves into $w_{d+1}/2$ pairs, each pair of leaves sharing the same parent at depth w_d. We therefore can contract these pairs with their parents to a contracted tree T' having $w_d + w_{d+1}/2$ leaves at depth w_d. Since T' is a full binary tree and $w_d + w_{d+1}/2 > 0$, according to the induction hypothesis, the full-binary distribution $\boldsymbol{w}' = (w_1, \ldots, w_d + w_{d+1}/2)$ of T' satisfies $f(\boldsymbol{w}') = 1$. This yields $f(\boldsymbol{w}') = f(\boldsymbol{w})$ because

$$1 = \sum_{i=1}^{d-1} \frac{w_i}{2^i} + \frac{w_d + w_{d+1}/2}{2^d} = \sum_{i=1}^{d+1} \frac{w_i}{2^i} \, .$$

Direction $\Leftarrow$. Given a vector $\boldsymbol{w} = (w_1, \ldots, w_{d+1})$ with nonnegative integers such that $f(\boldsymbol{w}) = 1$, i.e., $\sum_{i=1}^{d+1} \frac{w_i}{2^i} = 1$. Multiplying this equation on both sides by 2^{d+1} gives $2^d w_1 + 2^{d-1} w_2 + \cdots + 2 w_d + w_{d+1} = 2^{d+1}$, and hence w_{d+1} must be even since all other terms are even. Therefore, $w'_d = w_d + w_{d+1}/2$ is integer and $\boldsymbol{w}' = (w_1, \ldots, w_{d-1}, w'_d)$ is a vector with $f(\boldsymbol{w}') = 1$ because

$$\sum_{i=1}^{d-1} \frac{w_i}{2^i} + \frac{w'_d}{2^d} = \sum_{i=1}^{d-1} \frac{w_i}{2^i} + \frac{w_d}{2^d} + \frac{w_{d+1}}{2^{d+1}} = 1.$$

By the induction hypothesis, $\boldsymbol{w}'$ is a full-binary distribution of a full binary tree T'. We now expand $w_{d+1}/2$ leaves of T' at depth d to internal nodes, each having two leaf children. This gives us a full binary tree whose full-binary distribution is $\boldsymbol{w}$. ◄

Let $g(i, \boldsymbol{w}) = \sum_{j=i}^{d} w_j / 2^{j-i}$ for $i \in [1..d]$.

▶ **Corollary 5.** *Given a full-binary tree distribution $\boldsymbol{w} = (w_1, \ldots, w_d)$, then $g(i, \boldsymbol{w})$ is the number of nodes with depth j, which is even, for every $i \in [1..d]$.*

Proof. Following the proof of Lemma 4, we iteratively merge the leaves on depth j with their parent node for j from d to $i + 1$. At each step, we obtain a full binary tree such that the number of leaves with the highest depth is always even. ◄

▶ **Theorem 6.** *The number of full binary trees having a full-binary distribution $\boldsymbol{w} = (w_1, \ldots, w_d)$ with $w_d > 0$ and $f(\boldsymbol{w}) = 1$ is*

$$\prod_{i=1}^{d-1} \binom{g(i, \boldsymbol{w})}{w_i} \, .$$

Proof. We show the claim by induction on d. The base case is $d = 1$, for which we have seen that $\boldsymbol{w} = (2)$ is uniquely defined. For the induction hypothesis, let us assume that the claim holds for d.

Consider a full-binary distribution $\boldsymbol{w} = (w_1, \ldots, w_{d+1})$ with $w_{d+1} > 0$ and $f(\boldsymbol{w}) = 1$. Then $g(i, \boldsymbol{w})$ is even for all i according to Corollary 5. In particular, the vector $\boldsymbol{w}' := (w_1', \ldots, w_d')$ defined by $w_i' = w_i$ for $i \in [1..d-1]$ and $w_d' = w_d + w_{d+1}/2$ is an integer vector with $f(\boldsymbol{w}') = 1$. With the induction hypothesis for $\boldsymbol{w}'$ we obtain

$$\prod_{i=1}^{d-1} \binom{\sum_{j=i}^{d} \frac{w_j'}{2^{j-i}}}{w_i'} = \prod_{i=1}^{d-1} \binom{w_i + \frac{w_{i+1}}{2} + \cdots + \frac{w_{d-1}}{2^{(d-1)-i}} + \frac{w_d + w_{d+1}/2}{2^{d-i}}}{w_i} = \prod_{i=1}^{d-1} \binom{\sum_{j=i}^{d+1} \frac{w_j}{2^{j-i}}}{w_i}.$$

Each of these trees has $w_d + w_{d+1}/2$ leaves with depth d. Fix one of them, which we modify to obtain a tree whose full-binary distribution is $\boldsymbol{w}$. For that, we select $w_{d+1}/2$ of those leaves, for which we have $\binom{w_d + w_{d+1}/2}{w_{d+1}/2} = \binom{w_d + w_{d+1}/2}{w_d}$ possibilities. By modifying each of the trees whose full-binary distribution is $\boldsymbol{w}'$ in all possible ways, we obtain

$$\prod_{i=1}^{d-1} \binom{\sum_{j=i}^{d+1} \frac{w_j}{2^{j-i}}}{w_i} \cdot \binom{w_d + w_{d+1}/2}{w_d} = \prod_{i=1}^{d} \binom{\sum_{j=i}^{d+1} \frac{w_j}{2^{j-i}}}{w_i}.$$

By construction, all modifications are distinct by using only expansions. If two expansions are equal, they must have originated from the same tree we expanded.   ◀

▶ **Example 7.** For $d = 2$, there are two full-binary distributions $\boldsymbol{w}_1 = (0, 4)$ and $\boldsymbol{w}_2 = (1, 2)$. For $\boldsymbol{w}_1$, $\binom{0 + \frac{4}{2}}{0} = 1$, and there is only the perfect full binary tree having $\boldsymbol{w}_1$ as full-binary distribution. For $\boldsymbol{w}_2$, $\binom{1 + \frac{2}{2}}{1} = 2$, there are two full binary trees, as shown in Figure 1.
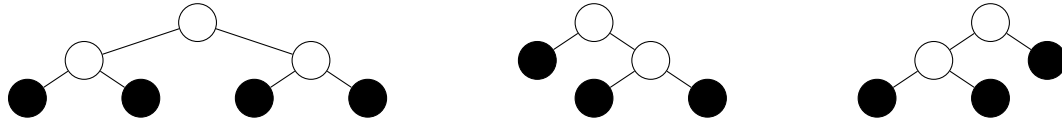

Figure 1 Illustration of Example 7, where the left tree has the full-binary distribution $(0, 4)$ and the other trees $(1, 2)$. Black nodes are leaves.

5   Enumeration of Full Binary Trees

In what follows, we present an enumeration algorithm for Problem 1 in the case of full binary trees. Our main idea is to process the trees in a linear order, which we define by a binary encoding of the trees.

For that, we recall the result of Corollary 5 that $g(i, \boldsymbol{w})$ gives the number of nodes with depth $i \in [1..d]$, which is common to all full binary trees having the full-binary distribution $\boldsymbol{w}$. In short, we define the vector $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_d)$ with $\eta_i := g(i, \boldsymbol{w})$ for $i \in [1..d]$. Then $\sum_{i=1}^{d} \eta_i$ is the number of all nodes. In particular, we can use an η_i-length bit vector to specify which nodes on depth i are internal nodes or leaves. The list of bit vectors on all depths is sufficient to represent a full binary tree. Then $\sum_{i=1}^{d} \eta_i = \sum_{i=1}^{d} \sum_{j=i}^{d} \frac{w_j}{2^{j-i}} \leq 2 \sum_{i=1}^{d} w_i$. For $n = \sum_{i=1}^{d} w_i$, this means that we can represent each full binary tree whose full-binary distribution is $\boldsymbol{w}$ in $2n$ bits. We use this bit representation to enumerate these trees. For that, we build on the algorithm of [14].

▶ **Lemma 8** ([14]). *There is an algorithm that enumerates all subsets with $m \in [1..n]$ integers of the integer range $[1..n]$ in ascending lexicographic order with constant delay.*
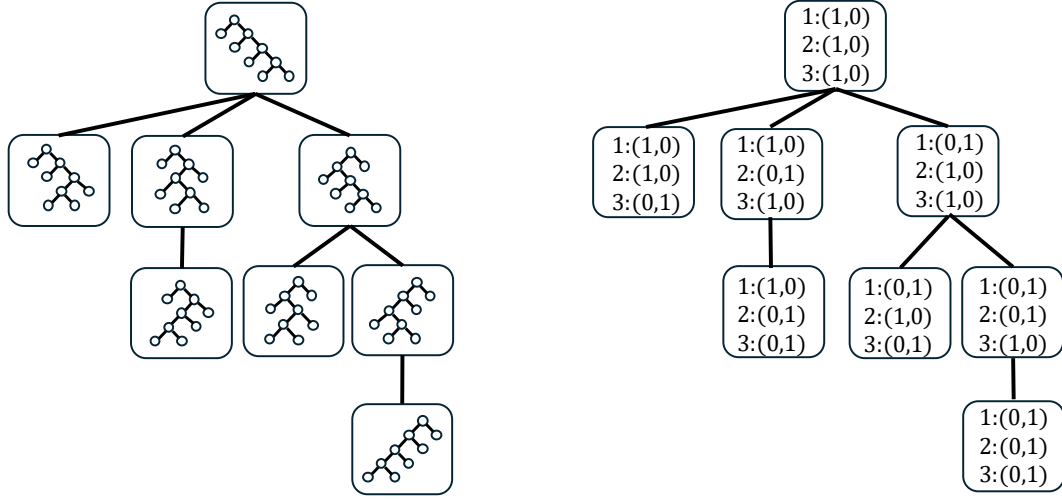
■ **Figure 2** Enumerating all full binary trees whose distribution is $\boldsymbol{w} = (1, 1, 1, 2)$ with $d = 4$. Both trees on the left and right depict the same enumeration tree of $\boldsymbol{w}$ explained in Section 5, but with a different representation. A node on the left depicts a full binary tree T whose encoding $\boldsymbol{B}_T$ is given by a node on the right at the same position in the respective tree. We partition each $\boldsymbol{B}_T$ by the depths $[1..d-1]$ (depth d contains only zeros). For instance, the root node of the right enumeration tree E specifies with 10 that there is a left leaf and a right internal node on each depth in $[1..3]$, while the deepest leaf of E specifies a tree with opposite characteristics.

The connection to our problem is that a bit vector of length n with m "1"s exactly represents such a subset. Since we can impose the natural lexicographic order on the $2n$-length bit vectors, the missing step is to find, given a $2n$-length bit vector representing a full binary tree, the lexicographic succeeding bit vector that represents a tree we want to output.

To facilitate our starting, we assume that $w_i \geq 1$ for every $i \in [1..d]$. Suppose that we have a full-binary tree T whose distribution is $\boldsymbol{w}$. We deduce from T a bit vector $\boldsymbol{B}_T$ representing T. For explanation, we assume that the least significant bits are on the left. We start with a bit vector $\boldsymbol{B}_T$ of length $\sum_{i=1}^{d} \eta_i \leq 2n$ to represent the order of the internal nodes and leaves on each depth. To do this, we partition $\boldsymbol{B}_T$ by depths $\boldsymbol{B}_T = (\boldsymbol{b}^{(1)} \cdots \boldsymbol{b}^{(d)})$ such that $\boldsymbol{b}^{(i)}$ is a bit vector of length η_i with $b_j^{(i)} = 1$ if and only if the j-th node with depth i is a leaf. In particular, $\boldsymbol{b}^{(i)}$ has w_i zeros. As an example, the values of $\boldsymbol{B}$ for the trees depicted in Figure 1 are 001111 (left), 1011 (middle), and 0111 (right). (Recall that $\boldsymbol{B}_T$ does not encode the root node, for which we assume it is always an internal node.) As a practical optimization, we can omit the highest depth d since it always contains leaves, i.e., $\boldsymbol{b}^{(d)}$ contains only zeros. Since we can reconstruct T from $\boldsymbol{B}_T$ uniquely, there is a one-to-one mapping from full binary trees with at least two nodes and a subset of bit strings defined by the above tree encoding. Consequently, each full binary tree has exactly one specific bit vector representation.

In what follows, we represent each full binary tree T having the distribution $\boldsymbol{w}$ with the bit vector $\boldsymbol{B}_T$, which is stored as a node in a so-called enumeration tree E. The root of E represents the full binary tree T obtained by grouping all internal nodes to the left side, which we expressed by the lexicographic least bit vector, obtained by shifting all "1"s in every $\boldsymbol{b}^{(i)}$ of $\boldsymbol{B}_T$ to the left. An E-node is augmented by a bit vector $\boldsymbol{B}_T$ representing a full binary tree T and an integer, which we call the *change level*. The change level of an E-node v states the depth at which the bit vector of v differs from its parent; as an exception, we
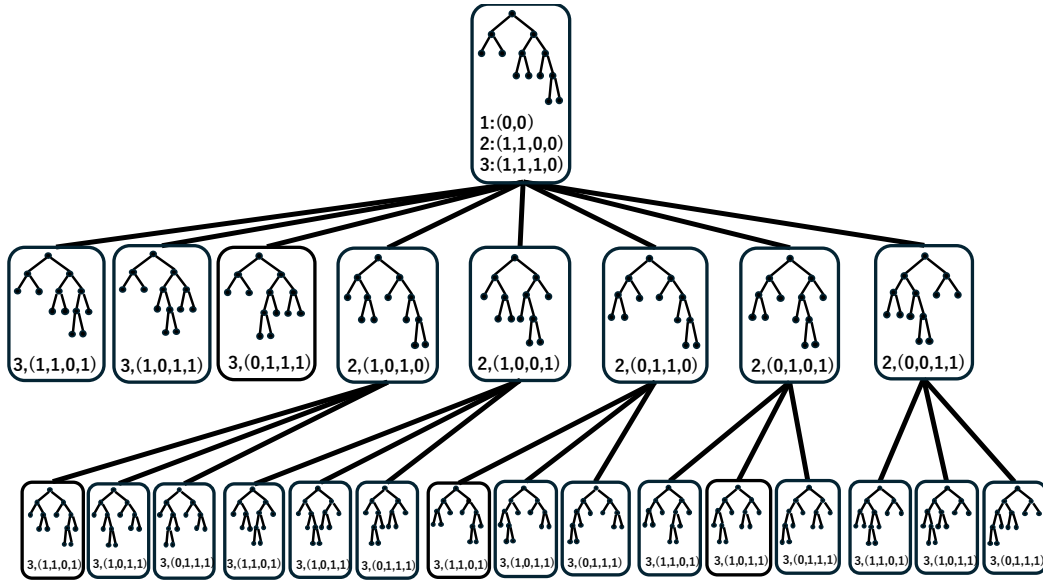
Figure 3 An enumeration tree E on an extended example compared to Figure 2, with distribution $\boldsymbol{w} = (0, 2, 3, 2)$. We here provide a combined view of both trees as illustrated in Figure 2. While the root node shows its full bit vector, we only depict the bit vector at the change level of each other node.

stipulate that the change level of the root node is 0. The change level is an invariant we impose on E in the sense that, when traversing from a node u to of its children v, the tree represented by u and by v can and must differ only at the change level of v. A consequence is that E-nodes with change level d are leaves.

We organize the children of an E-node as follows. Fix one E-node v with bit vector $\boldsymbol{B}_T = (\boldsymbol{b}^{(1)} \cdots \boldsymbol{b}^{(d)})$ and change level $\delta \in [0..d-1]$. Then the children of v correspond to all possible rearrangements of the bit patterns in one of the vectors $\boldsymbol{b}^{(\delta+1)}, \ldots, \boldsymbol{b}^{(d)}$. We start with the first children of v that enumerate all possible bit patterns of $\boldsymbol{b}^{(d)}$ (resulting by permutations). So they have change level d with a bit vector that differs from v's bit vector only at depth d. (In particular, these nodes are leaves by the definition of the change level.) Next, we continue with the group of children for $\boldsymbol{b}^{(d-1)}$ and change level $d-1$, and so on, up until $\boldsymbol{b}^{(\delta+1)}$ and change level $\delta + 1$. In particular, we arrange each group of children with the same change level i in lexicographic order of their corresponding bit vector $\boldsymbol{b}^{(i)}$. This order of the children and the order of their groups allows us to enumerate all full binary trees by a pre-order traversal of E in lexicographic order. (In particular, the lexicographic order imposes the invariant that we never enumerate a full binary tree twice.) This structure of E warrants the following – see Figure 2 for an example and Figure 3 for a more elaborated example involving a non-binary choice at one level:

- Any subtree rooted at a node of E is an enumeration tree itself. That is because an E-node with change level $\delta \in [1..d]$ representing a tree T is the root node of an enumeration tree that considers only the suffix $\boldsymbol{b}^{(\delta+1)} \cdots \boldsymbol{b}^{(d)}$ of $\boldsymbol{B}_T$ for the enumeration. In detail, for an E-node with change level δ it holds that $\boldsymbol{b}^{(i)} = (\overbrace{1 \cdots 1}^{w_i} \overbrace{0 \cdots 0}^{\eta_i - w_i})$ for every $i \in [\delta + 1..d-1]$ by the construction E – recall that a node with change level δ only makes a change on $\boldsymbol{b}^{(\delta+1)}$ compared to its parent node, and reading the change levels upwards to the root

gives a monotone decreasing sequence of change levels (so there is no way that any deeper level has changed with respect to the initial bit vector of the root.) In particular, an E-node is the lexicographically least among all its descendants.

- Since deeper recursion levels only touch shorter suffixes of the bit vectors, the rightmost leaf of an E-node v is lexicographically smaller than v's right sibling (if it exists). Therefore, all E-nodes represent different vectors, and the sum of all E-nodes is the size of the output we want to compute.

If we now run the algorithm of Lemma 8 on d nested loops we can traverse the constructed enumeration tree E in a pre-order traversal and output each node upon visiting it. The maximum delay happens when moving to the next value in the lowest $d-1$ loops. This delay is $O(d)$, but $O(1)$ amortized because to reach a depth of d' we have already output d' nodes.

It is left to argue that we can transform the sequence of outputted E-nodes into the full binary trees we want to output. For that, we initially build the full binary tree represented by the root of E from scratch, taking $O(n)$ time. Subsequently, for each E-node visit, it suffices to change the order of the nodes of the previously computed full binary tree at a specific depth, costing $O(\max_i g(i, \boldsymbol{w}))$ time.

▶ **Lemma 9.** *Given a full-binary distribution $\boldsymbol{w}$ with $w_i \geq 1$ for all $i \in [1..d]$, we can enumerate all full binary trees whose full-binary distribution is $\boldsymbol{w}$ in lexicographic order with constant amortized delay or $O(d)$ worst-case delay. The precomputation takes $O(n)$ time.*

We can generalize this approach to full-binary distributions having zero values. The obstacle is that we cannot guarantee constant delay since we might traverse long paths with zero weights. As a remedy, we want to skip each depth i with $w_i = 0$. To do that, let us denote by $I = \{i \in [1..d] \mid w_i > 0\}$ all depths with leaves. Then it suffices to restrict the values of a change level of a node to values of $I \cup \{0\}$. By construction, the root node has change level 0, and a node with change level δ has children whose augmented value is the successor of δ in I.

▶ **Lemma 10.** *Given a full-binary distribution $\boldsymbol{w}$ with $w_d \geq 1$, we can enumerate all full binary trees whose full-binary distribution is $\boldsymbol{w}$ in lexicographic order with constant amortized delay or $O(d)$ worst-case delay. The precomputation takes $O(n)$ time.*

To obtain worst-case constant delay time, we can drop the requirement to output in lexicographic order by using a folklore technique, such as the alternating output technique [15]. The idea is to output nodes on odd depths when visiting them the first time, but postpone the output of nodes on even depths until we backtrack and move outside their respective subtrees.

▶ **Theorem 11.** *Given a full-binary distribution $\boldsymbol{w} = (w_1, \ldots, w_d)$, the corresponding full binary tree can be output in $O(d+n)$ time for the first tree and then output all trees in the form of differences with a size of $O(m)$ with a delay of $O(1)$ in $O(m)$ time, where $n = \sum_{i=1}^{d} w_i$ and $m = \max\{w_i \mid i \in [1..d]\}$.*

6   Enumeration of Binary Trees

In what follows, we show how to generalize the enumeration of full binary trees to general binary trees, where internal nodes are allowed to have one or two children.

From Lemmas 3 and 4, the binary distribution $\boldsymbol{w} = (w_1, \ldots, w_d)$ of a non-full binary tree also satisfies $f(\boldsymbol{w}) = \sum_{i=1}^{d} w_i / 2^i < 1$. We make the connection to full binary trees by adding dummy leaves to all internal nodes that have only one child. To this end, we denote the

number of dummy leaves added at depth i by x_i, and construct a vector $\boldsymbol{x} = (x_1, \ldots, x_d)$. The sum $\boldsymbol{x} + \boldsymbol{w}$ gives a vector that is the binary distribution of a full binary tree because $\sum_{i=1}^{d}(w_i + x_i)/2^i = 1$. In what follows, we call $\boldsymbol{x}$ the *complement vector* of $\boldsymbol{w}$, and $\boldsymbol{w} + \boldsymbol{x}$ the *completion* of $\boldsymbol{w}$. For such $\boldsymbol{x}$, a nonnegative integer vector $\boldsymbol{y}$ that satisfies $\boldsymbol{y} \leq \boldsymbol{x}$ is called the *subcomplement vector* of $\boldsymbol{w}$.

▶ **Lemma 12.** *For a complement vector $\boldsymbol{x}$ of $\boldsymbol{w}$ we have that $x_i \leq g(i, \boldsymbol{w})$.*

Proof. We can only attach a dummy leaf at depth i to internal nodes that have exactly one child on depth i. By the pigeonhole principle, there are at most $g(i, \boldsymbol{w})$ many such children on depth i. ◀

From Lemmas 3 and 4, the following immediately holds.

▶ **Lemma 13.** *If the binary distribution $\boldsymbol{w}$ satisfies $f(\boldsymbol{w}) < 1$, then there exists a complement vector for $\boldsymbol{w}$.*

However, the complement vector is not unique, in general. For example, $\boldsymbol{w} := (0, 2, 2)$ is not a full-binary distribution since $2/4 + 2/8 = 3/4$, while $(0, 1, 0)$ and $(0, 0, 2)$ are both complement vectors of $\boldsymbol{w}$. See Figure 4 for an example.

■ **Figure 4** Examples for Lemma 13 with the binary distribution $\boldsymbol{w} = (0, 2, 2)$, where two binary trees exist, which can be complemented by adding dummy leaves. The left tree and the right tree can be complemented by the vectors $(0, 0, 2)$ and $(0, 1, 0)$ to full binary trees, respectively. Black nodes are leaves, and dummy nodes are rectangles.

Also, even if $\boldsymbol{x}$ satisfies $f(\boldsymbol{w} + \boldsymbol{x}) = 1$, it does not necessarily mean that $\boldsymbol{x}$ is the complement vector of $\boldsymbol{w}$. For example, $\boldsymbol{w} := (0, 2, 0, 2)$ has $(0, 0, 2, 2)$ as a complement vector, but $\boldsymbol{x} = (0, 0, 0, 6)$ is not a complement vector: A tree having the binary distribution $\boldsymbol{w}$ cannot have six dummy leaves on depth 4. However, a necessary condition is given below.

▶ **Lemma 14.** *A nonnegative integer vector $\boldsymbol{x} = (0, \ldots, 0, y_d)$ is a subcomplement vector of a complement vector $\boldsymbol{y} = (y_1, \ldots, y_d)$ of a binary distribution $\boldsymbol{w} = (w_1, \ldots, w_d)$ if and only if $y_d \equiv w_d \pmod{2}$, $y_d \in [0, w_d]$, and $f(\boldsymbol{w}) + y_d/2^d \leq 1$.*

Proof. Let $\boldsymbol{x} = (0, \ldots, 0, y_d)$.

Direction $\Rightarrow$. In order for $\boldsymbol{x}$ to be a subcomplement vector of $\boldsymbol{w}$, a binary tree with $y_d + w_d$ leaves, including dummy leaves, must exist. Therefore, it follows from Corollary 5 that $y_d + w_d$ is even, so $y_d \equiv w_d \pmod{2}$. Also, since dummy leaves can only be attached to internal nodes that have exactly one child (in this case, a leaf), $y_d \leq w_d$ by Lemma 12. Furthermore, in order for $\boldsymbol{x} + \boldsymbol{w}$ to be a binary tree (with dummy leaves), $f(\boldsymbol{w} + \boldsymbol{x}) = \sum_{i=1}^{d}(w_i + x_i)/2^i = \sum_{i=1}^{d} w_i/2^i + x_d/2^d \leq 1$ must hold.

Direction $\Leftarrow$. We show that there exists a complement vector $\boldsymbol{y}$ for $\boldsymbol{w}$ such that $\boldsymbol{x} \leq \boldsymbol{y}$.

For depth $d = 1$, $\boldsymbol{w}$ must be either (1) or (2). In the case of $\boldsymbol{w} = (2)$, $\boldsymbol{w}$ represents a full binary tree, so the complement vector of $\boldsymbol{w}$ is $\boldsymbol{y} = (0)$, and thus $\boldsymbol{x} = (0)$. In the case of $\boldsymbol{w} = (1)$, the only complement vector is $\boldsymbol{y} = (1)$, and hence $\boldsymbol{x} = (1)$ is the unique subcomplement vector of $\boldsymbol{y} = (1)$ satisfying $x_d = y_d$. Thus, in both cases the subcomplement vectors are uniquely defined by the right-hand side conditions.

In what follows, we assume that $d \geq 2$. Given a binary distribution $\boldsymbol{w}$ of a binary tree T, by assumption $w_d + y_d$ and $w_d - y_d$ are even numbers. Hence, we can apply the contraction trick like in previous proofs. For that, consider the vector $\boldsymbol{w}' := (w_1, \ldots, w_{d-2}, w_{d-1} + (w_d + y_d)/2)$. The vector $\boldsymbol{w}'$ is well-defined: Since $w_d \equiv y_d \mod 2$ and $y_d \leq w_d$, $(w_d + y_d)/2 = (w_d - y_d)/2 + y_d$ is a positive integer. Since

$$f(\boldsymbol{w}') = \sum_{i=1}^{d-2} w_i/2^i + (w_{d-1} + (w_d + y_d)/2)/2^{d-1} = f(\boldsymbol{w}) + y_d/2^d \leq 1,$$

by Lemmas 3 and 13, $\boldsymbol{w}'$ is a binary distribution of a binary tree T'. The normal leaves (i.e., excluding dummy leaves) of depth $d - 1$ in T' are $w_{d-1} + (w_d + y_d)/2$ in total, of which

- w_{d-1} of them are former leaves of T unaffected by the contraction,
- $(w_d + y_d)/2$ internal nodes of T are changed into leaves on depth $d - 1$ by contracting them with their two child leaves (where one may be a dummy leaf).

The resulting binary tree T' therefore has $w_{d-1} + (w_d + y_d)/2$ leaves on depth $d - 1$ in total. The contraction of T to T' by removing depth d does not affect the number of leaves at depth $d - 2$ or less in T. By definition, there is a complement vector $\boldsymbol{z} = (z_1, \ldots, z_{d-1})$ of $\boldsymbol{w}'$ that can turn T' into a full binary tree. Finally, we turn y_d leaves of T' on depth d into internal nodes, each having one child. This restores T and induces the complement vector $\boldsymbol{z} = (z_1, \ldots, z_{d-1}, y_d)$ of $\boldsymbol{w}$. ◀

For every binary tree, there is only one way to add dummy leaves to internal nodes to make it a full binary tree. Hence, the complement vector for each binary tree, whose completion is the binary distribution of the full binary tree, is uniquely determined. Conversely, given a full binary tree and a configuration classifying leaves in dummy leaves and non-dummy leaves, we can obtain the original binary tree by removing dummy leaves according to the configuration to obtain the original binary tree. Based on this observation, we first find the completion, i.e., the corresponding full binary tree, for $\boldsymbol{w}$, and then enumerate the binary trees by considering all ways of adding dummy leaves while enumerating the corresponding full binary completions. That is, by enumerating all possible complement vectors for $\boldsymbol{w}$, and then considering the process of enumerating the different ways of adding dummy leaves while considering the corresponding full binary completion, we can enumerate all binary trees. In summarizing the above, the enumeration algorithm is designed in a three-layer structure:

1. enumeration of complement vectors (Section 6.1),
2. enumeration of the way of adding dummy leaves to each complement vector (Section 6.2), and
3. enumeration of binary trees for each way of adding dummy leaves (Section 6.3).

6.1 Enumeration of Complement Vectors

We enumerate the complement vectors with the insights of Lemmas 13 and 14. In detail, Lemma 14 gives us a strategy to enumerate the complement vectors by enumerating subcomplement vectors. We show the algorithmic steps in Algorithm 1, which is a recursive algorithm
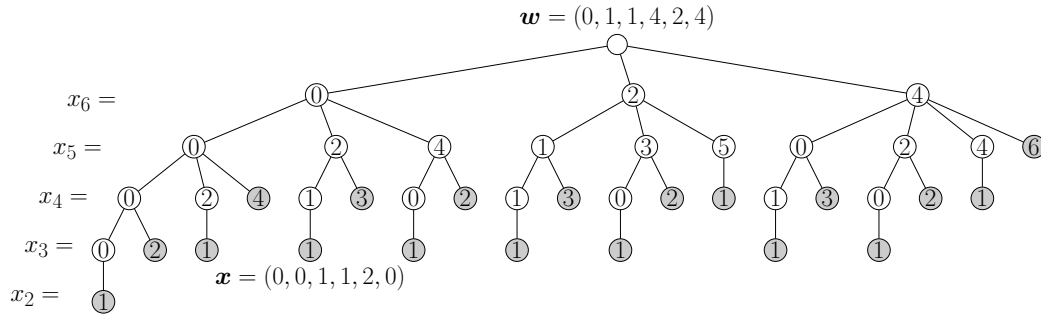
Figure 5 Enumeration tree of complement vectors for the binary distribution $\boldsymbol{w} = (0, 1, 1, 4, 2, 4)$, where leaves correspond to the complement vectors. A node on depth i determines the value $x[d - i + 1]$ of the complement vector. The details are described in Section 6.1.

Algorithm 1 Subroutine for enumerating complement vectors for $\boldsymbol{w}$, cf. Section 6.1.

1: call EnumCompVector with $W \leftarrow f(\boldsymbol{w})$, $\boldsymbol{x} = (0, \ldots, 0)$, $i = d$ and $c = 0$
2: **procedure** ENUMCOMPVECTOR$(W, \boldsymbol{x}, i, c)$ $\triangleright$ $\boldsymbol{x}$: subcomplement vector, $i \in [1..d]$:
   depth, c: number of conceptional leaves on depth i.
3:                $\triangleright$ determine all dummy nodes on depth i and recursively on lower depths.
4:    $x_i \leftarrow (w_i + c) \bmod 2$ $\triangleright$ initialization. By Lemma 14 the parity must be the same.
5:    **while** $x_i \leq w_i + c$ **and** $W + x_i/2^i \leq 1$ **do**
6:      $\boldsymbol{y} \leftarrow \boldsymbol{x} + (0, \ldots, 0, x_i, 0, \ldots)$ $\triangleright$ $\boldsymbol{y}$ meets the requirements of Lemma 14.
7:      **if** $W + x_i/2^i = 1$ **then**
8:        **Output $\boldsymbol{y}$** $\triangleright$ output a complement vector
9:        **return**
10:     **else**
11:       ENUMCOMPVECTOR$(W + x_i/2^i, \boldsymbol{y}, i - 1, (x_i + w_i)/2)$ $\triangleright$ recurse on depth $i - 1$
12:     **end if**
13:     $x_i \leftarrow x_i + 2$
14:   **end while**
15:   **return**
16: **end procedure**

---

that determines the values of x_i in descending order $x_d, x_{d-1}, \ldots$ in a depth-first manner. We initially call ENUMCOMPVECTOR with the arguments $(f(\boldsymbol{w}), \boldsymbol{0}, d, 0)$, for the weight $f(\boldsymbol{w})$ of the binary distribution $\boldsymbol{w}$, $\boldsymbol{0}$ the d-dimensional vector with 0 entries, the depth d and the number of conceptional leaves on depth d. The last argument needs a definition: This algorithm recursively contracts leaves and dummy leaves at depth $i + 1$ to the so-called *conceptional* leaves on depth i. If we have x_{i+1} dummy leaves and w_{i+1} ordinary leaves on depth $i + 1$, then the number of conceptional leaves at depth i is $c_i = (x_{i+1} + w_{i+1})/2$, and we consider from then on the binary distribution $(w_1, w_2, \ldots, w_{i-1}, w_i + c_i)$ for a tree of depth i. Now, in the function ENUMCOMPVECTOR, we enumerate all x_i satisfying the conditions of Lemma 14 at Line 5. We proceed with each such x_i at Line 6 with a subcomplement vector $\boldsymbol{y} = \boldsymbol{x} + (0, \ldots, 0, x_i)$. At Line 6, we check if we have obtained a completion with $\boldsymbol{y}$ at that point. If $\boldsymbol{y}$ is a complement vector of $\boldsymbol{w}$, we output $\boldsymbol{y}$ at Line 7 and return. If we can still increase a value in $\boldsymbol{y}$ to add dummy leaves (Line 10), we recursively call the procedure for the preceding depth $i - 1$. By Lemma 14, if we recurse, we either continue recursing or
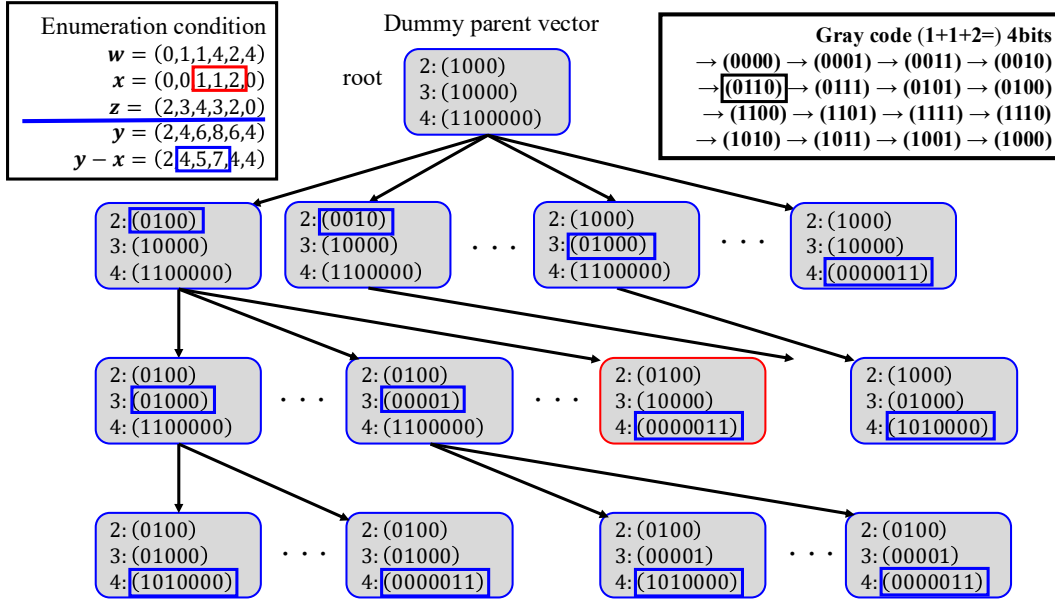
Figure 6 Enumeration of dummy parent configurations and Gray code, cf. Section 6.2. We start with $\boldsymbol{w}$ and $\boldsymbol{x}$ defined in Figure 5. The third row in the left upper box is a vector $\boldsymbol{z}$ with $z_i = \sum_{j=i+1}^{d} (w_j + x_j)/2^{j-i}$ such that $\nu_i = w_i + x_i + z_i$. We only have dummy leaves on the depths $3, 4, 5$ (those depths i for which $x_i > 0$ marked by a red rectangle around the entries of $\boldsymbol{x}$). Therefore, our dummy parent configuration considers only the depths $2, 3, 4$, for which we generate bit vectors for each node in the depicted enumeration tree. The lengths of the bit vectors are determined by $\vec{\nu} - \vec{x}$ by the entries i with $x_{i+1} > 0$, which are 4,5,7 in our case (blue rectangle in the upper left box). For each node in the enumeration tree we visit, we enumerate all Gray codes on the upper right box by visiting linearly all depicted nodes in this linked list. The Gray code has length $\sum_{i=1}^{d} x_i$ and assigns a dummy parent the role whether it has a left or a right dummy leaf.

turn a subcomplement vector into a complement vector. We can write each recursive call in the form of a tree, where each node is a call of ENUMCOMPVECTOR: the root is the initial call, and each leaf is a call that returns a complement vector. See Figure 5 for an example. Finally, since we can evaluate the checks in the pseudocode in $O(1)$ time, the delay between the output of two complement vectors is $O(d)$ time.

6.2   Enumeration of Dummy Leaves

Given the complement vector $\boldsymbol{x}$ for $\boldsymbol{w}$, there can be multiple ways to add the dummy leaves specified by $\boldsymbol{x}$ to a binary tree whose binary distribution is $\boldsymbol{w}$. Our task is to enumerate all these ways. For that, we follow the strategy of Section 5, where we enumerate the leaf/internal node configurations at depth i for full binary trees. However, unlike normal leaves, the sibling of a dummy leaf must be a normal leaf or an internal node. To facilitate checking this condition, we enumerate the internal node one level above the dummy leaf itself, rather than at the configuration of the dummy leaf. We call such an internal node the *dummy parent* of a dummy leaf. For each dummy parent, it suffices to enumerate the two possibilities of having its dummy leaf as its left or right child.

First, as in Section 5, suppose that we have a full-binary distribution $\boldsymbol{w}+\boldsymbol{x}$. From the proof of Corollary 5, the number of nodes at depth i is $\nu_i := g(i, \boldsymbol{w} + \boldsymbol{x}) = \sum_{j=i}^{d} (w_j + x_j)/2^{j-i}$.

Suppose δ is the smallest depth at which a dummy leaf appears for the first time. At depth δ, there are x_δ dummy leaves. By definition, there are x_δ dummy parents at depth $\delta - 1$. We specify these dummy parents by a bit vector of length $\nu_{\delta-1}$ with "1"s at x_δ positions.

In doing so, at depth $\delta - 1$, we have already determined the dummy parents of the dummy leaves at depth δ, regardless of whether they have a dummy leaf as their left or right child, so the number of nodes that we can freely choose at depth δ is $\nu_\delta - x_\delta$. We again specify the dummy parents at depth δ and recurse. Therefore, we can express our selection of dummy parents by a $(\nu_\delta - x_\delta)$-length bit vector with "1"s in $x_{\delta+1}$ positions. By recursing on deeper levels, we can apply this technique on all depths, generalizing from δ to any $i \in [\delta..d]$. In what follows, we want to express the configuration of dummy parents on each level by a bit vector, for which we want to bound the size of ν_i.

▶ **Lemma 15.** *Each ν_i is at most $2 \sum_{j=i}^{d} w_j$.*

Proof. From Lemma 14, $\nu_d = w_d + x_d \leq 2w_d$. So there is an $i \in [0..d-1]$ such that the induction hypothesis $\nu_{d-k} \leq 2\sum_{j=d-k}^{d} w_j$ holds for every $k \in [0..i]$. Then we obtain

$$\nu_{d-(i+1)} = w_{d-(i+1)} + \nu_{d-i}/2 + x_{d-(i+1)} \leq 2w_{d-(i+1)} + \nu_{d-i} \leq 2 \sum_{j=d-(i+1)}^{d} w_j \,,$$

where we used Lemma 12 for $x_{d-(i+1)} \leq \nu_{d-i}/2 + w_{d-(i+1)}$.   ◀

Therefore, we can specify dummy parents with an $O(\sum_{j=i}^{d} w_j)$-length bit vector at each depth. We denote the sequence of bit vectors for specifying the dummy parent configuration by $\boldsymbol{D}$.

Furthermore, we express whether a dummy leaf at depth i is the left or right child of its parent by a bit vector of length x_i (for each configuration of the dummy parents at depth $i - 1$, there are 2^{x_i} ways for the x_i dummy leaves at depth i to be attached). In total, we can express the dummy leaf configuration by a $(\sum_{i=1}^{d} x_i)$-length bit vector. Since $x_i \leq \nu_i \leq 2\sum_{j=i}^{d} w_j$ by the above analysis, this bit vector has length $O(d\sum_{i=1}^{d} w_i)$.

An enumeration of bit vectors are Gray codes [8], which have $O(1)$ delay because two subsequently output bit vectors differ by one bit. Here we use Gray codes to enumerate the dummy leaf configuration for each dummy parent configuration with constant delay.

To conclude, we can enumerate the configuration of dummy leaves in the following manner:

- Enumerate the configuration of the dummy parents with constant time delay, and output this configuration in $O(\sum_{i=1}^{d} w_i)$.
- Generate a Gray code of $\sum_{i=1}^{d} x_i = O(d\sum_{i=1}^{d} w_i)$ bits for each configuration of dummy parents, and enumerate the configuration of dummy leaves.

Figure 6 shows an example of a dummy parent enumeration tree and a Gray code for a binary distribution $\boldsymbol{w} = (0, 1, 1, 4, 2, 4)$ and its complement vector $\boldsymbol{x} = (0, 0, 1, 1, 2, 0)$.

6.3 Enumeration of the Leaf Configuration

Based on the previous discussions, at this point, we have determined the configuration of dummy parents $\boldsymbol{D}$ with x_i dummy leaves and x_{i+1} dummy parents of ν_i nodes at each depth i of the binary tree. Here, we assign roles to the remaining $\nu_i - x_i - x_{i+1}$ nodes as leaves and internal nodes having no dummy leaves as children. As before, we can specify the roles in
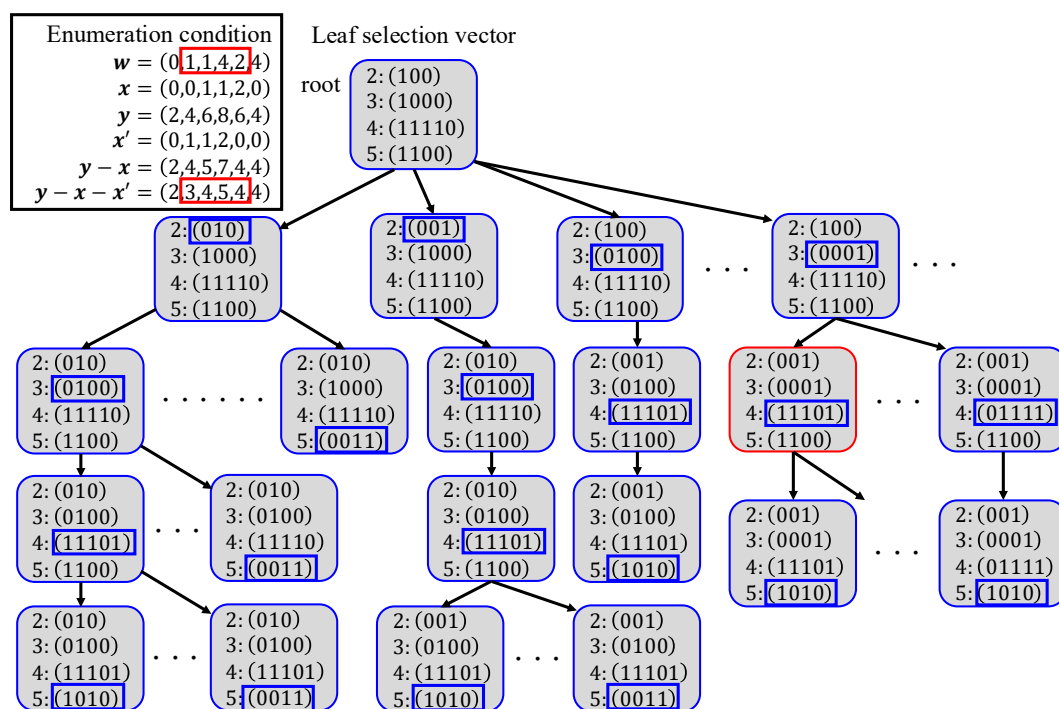
Figure 7 Enumeration of all leaf configurations, cf. Section 6.3. We follow the example of Figure 6. Here, $\boldsymbol{x}'$ denotes the vector $x_i' = x_{i+1}$. The number of leaves and internal nodes having no dummy leaves at depth i is $\nu_i - x_i - x_{i+1} = \nu_i - x_i - x_i'$. Since all nodes on depth 1 are internal nodes ($w_1 = 0$) and all nodes on depth d are leaves ($\nu_d = w_i$), the leaf selection vector only addresses the depths 2,3,4 (red rectangles in the left upper box).

the form of a bit vector of length $\nu_i - x_i - x_{i+1}$ with w_i "1"s. We call such a vector sequence a *leaf configuration vector sequence* and denote it by $\boldsymbol{L}$. Fixing one configuration of dummy leaves, we enumerate $\prod_{i=1}^{d} \binom{\nu_i - x_i - x_{i+1}}{w_i}$ combinations in the same way as in Section 5 to obtain all binary trees having that configuration of dummy leaves. We proceed in this order here despite that once $\boldsymbol{x}$ is determined, the number of dummy parents and leaves at each depth is fixed, so we can determine the leaf configuration vector independently of the dummy parent configuration. Figure 7 gives an example of a leaf configuration enumeration tree for a binary distribution $\boldsymbol{w} = (0, 1, 1, 4, 2, 4)$ and its complement vector $\boldsymbol{x} = (0, 0, 1, 1, 2, 0)$.

6.4   Overview of the Algorithm

We combine the above to enumerate binary trees using enumeration trees and Gray codes. For the sake of explanation, we enumerate the configuration of dummy leaves not directly after enumerating the configuration of dummy parents – we can take care of the dummy leaves at any time after we have determined the dummy parents. This also allows us to postpone the computation of the configuration of dummy leaves to the end.

We visualize our pipeline that ends at the generation of Gray codes following pointers from three nested enumeration tree traversals in Figure 8. There, we start with a top-down traversal of the leftmost tree T. Each leaf of T determines a complement vector $\boldsymbol{x}$. We perform the actual traversal by executing ENUMCOMPVECTOR. Fixing $\boldsymbol{x}$, we move to the neighboring trees $T^D(\boldsymbol{x})$ and $T^L(\boldsymbol{x})$ to enumerate all dummy parent configurations and all leaf configurations $\boldsymbol{D}$ and $\boldsymbol{L}$, respectively. First, we enter the enumeration tree $T^D(\boldsymbol{x})$. Each time we visit a new node in $T^D(\boldsymbol{x})$, we jump into $T^L(\boldsymbol{x})$ and traverse this tree. For each
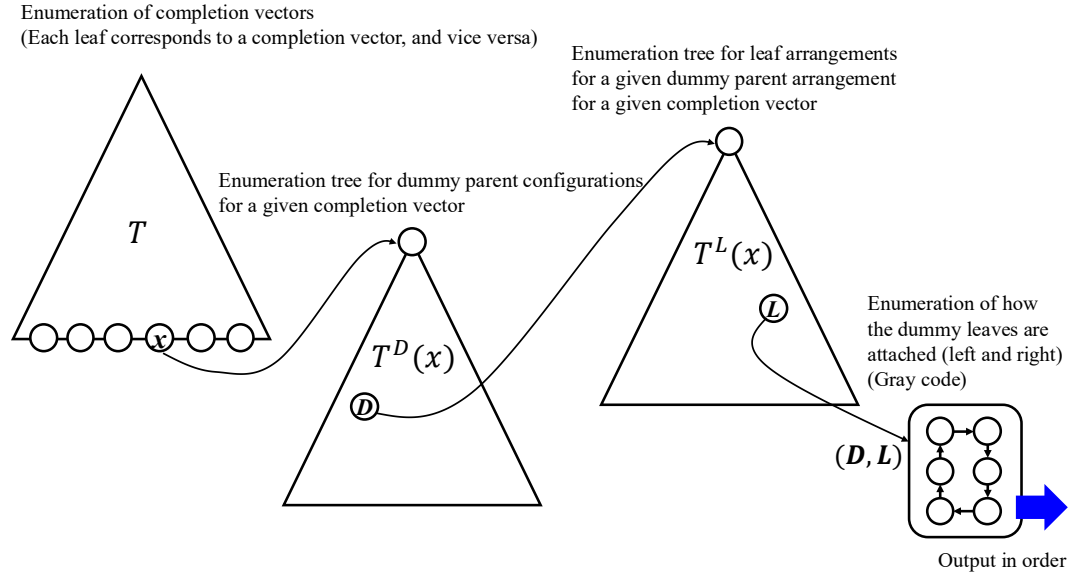
Figure 8 Layout of the algorithm enumerating all binary trees for a given binary distribution, cf. Section 6.4. An example of the leftmost tree T is given by Figure 5, and for the subsequent trees T^D and T^L by Figure 6 and Figure 7, respectively.

$T^L(\boldsymbol{x})$ node we visit, we have determined $(\boldsymbol{D}, \boldsymbol{L})$. Given $|x| = \sum_{i=1}^{d} x_i$, for each Gray code $\boldsymbol{g} \in \{0, 1\}^{|x|}$ we enumerate, we finally output $(\boldsymbol{D}, \boldsymbol{L}, \boldsymbol{g})$. From the discussion in Section 5, we can determine and output the pair $(\boldsymbol{D}, \boldsymbol{L})$ with constant delay. Since we can also determine $\boldsymbol{g}$ with constant delay, we can output $(\boldsymbol{D}, \boldsymbol{L}, \boldsymbol{g})$ for a fixed $\boldsymbol{x}$ with constant delay. However, the delay in outputting $\boldsymbol{x}$ itself is $O(d)$, but this time bound is dominated by the time bound to output the first $(\boldsymbol{D}, \boldsymbol{L}, \boldsymbol{g})$ for $\boldsymbol{x}$. This leads to the following complexities.

▶ **Theorem 16.** *Given a binary distribution $\boldsymbol{w} = (w_1, \ldots, w_d)$, the corresponding binary tree can be output in $O(d + n)$ time for the first tree and then output all trees in the form of differences with a size of $O(\sum_{i=1}^{d} w_i)$ with a delay of $O(1)$.*

While the complexities stated in Theorem 16 are in terms of $\boldsymbol{w}$, we create a full-binary tree whose binary distribution is $\boldsymbol{w} + \boldsymbol{x}$ for each completion vector $\boldsymbol{x}$. Nevertheless, $x_i \le w_i$ for all $i \in [1..d]$, and therefore the number of nodes of this full-binary tree is asymptotically equal to the number of leaves of the tree we output.

Finally, an example of the binary tree corresponding our running example is given in Figure 9.

7 Extension to α-ary trees

In this section, we extend the results of the previous section to general α-ary trees for a constant $\alpha \ge 2$. Like for binary trees ($\alpha = 2$), we first define the notion of an α-ary distribution.

7.1 Properties of full α-ary trees and their enumeration

We call a vector of integers $\boldsymbol{w} = (w_1, \ldots, w_d)$ a *(full) α-ary distribution* if there is a (full) α-ary tree that has w_i leaves on depth i for every $i \in [1..d]$. For a nonnegative integer vector $\boldsymbol{w} = (w_1, \ldots, w_d)$, we define the function $f_\alpha(\boldsymbol{w}) = \sum_{i=1}^{d} w_i / \alpha^i$, which is a generalization of f with $f_2 = f$. The problem we want to tackle in this section can be stated as follows.
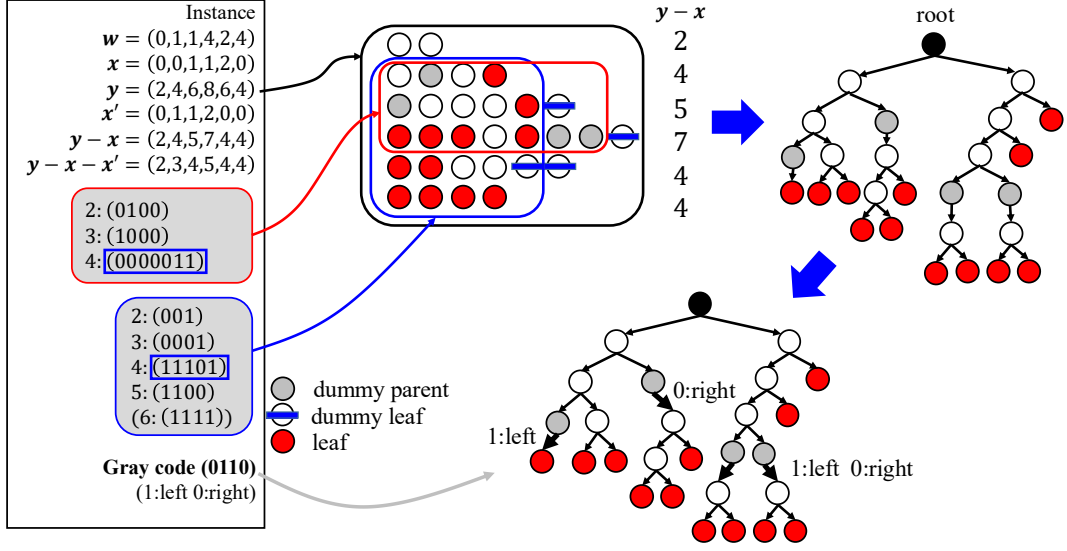
Figure 9 Binary tree corresponding to a vector representation returned by the algorithm whose complexities are stated in Theorem 16. The left box depicts a node of $T^D(\boldsymbol{x})$ (red, cf. Figure 6) and $T^L(\boldsymbol{x})$ (blue, cf. Figure 7). Informally, the red node specifies with a "1" the placement of a dummy parent, and the blue node with a "1" the placement of a (normal) leaf. Finally, a Gray code determines for each dummy parent whether its normal leaf is a left child or a right child.

▶ **Problem 17.** *Given a (full) α-ary distribution $\boldsymbol{w}$, enumerate all (full) α-ary trees whose α-ary distribution is $\boldsymbol{w}$.*

Like in the case of binary trees, the following holds.

▶ **Lemma 18** (Kraft's inequality). *Let $\alpha \geq 2$. A nonnegative integer vector $\boldsymbol{w} = (w_1, \ldots, w_d)$ that satisfies $w_d > 0$ is a α-ary distribution if and only if $f_\alpha(\boldsymbol{w}) \leq 1$.*

▶ **Corollary 19.** *Let $\boldsymbol{w} = (w_1, \ldots, w_d)$ be a nonnegative integer vector satisfying $w_d > 0$ and is a full α-ary distribution. Then, $g_\alpha(j, \boldsymbol{w}) := \sum_{i=j}^{d} w_i / \alpha^{i-j}$ is a multiple of α, and represents the number of nodes at depth j of the α-ary tree having $\boldsymbol{w}$ as its α-ary distribution.*

▶ **Theorem 20.** *Let $\alpha \geq 2$. Given a full α-ary distribution $\boldsymbol{w} = (w_1, \ldots, w_d)$ satisfying $w_d > 0$, the number of all full α-ary trees that have $\boldsymbol{w}$ as their α-ary distribution is*

$$\prod_{i=1}^{d-1} \binom{\sum_{j=i}^{d} \frac{w_j}{\alpha^{j-i}}}{w_i}.$$

From these properties, the enumeration of full binary trees can be extended to the enumeration of full α-ary trees. For that, we observe that the approach for enumerating full binary trees analyzed in Section 5 is independent of the degree of the tree, since the representation only considers the configuration of internal nodes and leaves per depth. Therefore, we can generalize this approach in a straightforward manner by replacing f and g with f_α and g_α, respectively, and updating the values of $\eta_j := g_\alpha(j, \boldsymbol{w})$. We obtain the following result based on an extension of Theorem 11:

▶ **Theorem 21.** *Given a full α-ary distribution $\boldsymbol{w} = (w_1, \ldots, w_d)$, the corresponding full α-ary tree can be output in $O(n)$ time for the first tree and then output all trees in the form of differences with a size of $O(m)$ with a delay of $O(1)$ in $O(m)$ time, where $n = \sum_{i=1}^{d} w_i$ and $m = \max\{w_i \mid i \in [1..d]\}$.*

7.2 Enumeration of α-ary trees

We can carry out the enumeration of α-ary trees in the same way as binary trees, for which we had three levels of enumeration: for the complement vectors, the dummy leaves, and the actual leaves. Here, we proceed in the same way.

7.2.1 Enumeration of full α-ary tree complement vectors

Suppose we have an α-ary distribution $\boldsymbol{w}$ with $f_\alpha(\boldsymbol{w}) < 1$. We fix one α-ary tree corresponding to $\boldsymbol{w}$ and add dummy leaves to all internal nodes with fewer than α children to make it a full α-ary tree. In this case, let the number of dummy leaves added at depth i be x_i, and consider the vector $\boldsymbol{x} = (x_1, \ldots, x_d)$. By construction, we obtain that $\sum_{i=1}^d (w_i + x_i)/\alpha^i = 1$. We call the vector $\boldsymbol{x}$ constructed in this way the *complement vector* of $\boldsymbol{w}$, and we call $\boldsymbol{w} + \boldsymbol{x}$ the *full α-ary tree complement* of $\boldsymbol{w}$. For such $\boldsymbol{x}$, a nonnegative integer vector $\boldsymbol{x}'$ is called a *subcomplement vector* of $\boldsymbol{w}$ if $\boldsymbol{x} \geq \boldsymbol{x}'$. Similarly to binary trees in Lemma 13, the following holds.

▶ **Lemma 22.** *Given an α-ary distribution $\boldsymbol{w}$ with $f_\alpha(\boldsymbol{w}) < 1$, there exists a complement vector of $\boldsymbol{w}$.*

Generalizing Lemma 14, the following is true.

▶ **Lemma 23.** *A nonnegative integer vector $(0, \ldots, 0, y_d)$ is a subcomplement vector of the complement vector $\boldsymbol{y} = (y_1, \ldots, y_d)$ of an α-ary distribution $\boldsymbol{w} = (w_1, \ldots, w_d)$ if and only if $y_d \equiv w_d \pmod{\alpha}$, $y_d \in [0..(\alpha-1)w_d]$, and $f_\alpha(\boldsymbol{w}) + y_d/\alpha^d \leq 1$.*

Using these lemmas, we can generalize Algorithm 1 to Algorithm 2 to enumerate the complement vector of an α-ary distribution.

Algorithm 2 Subroutine for enumerating full α-ary tree complement vectors for $\boldsymbol{w}$.

1: **procedure** ENUMCOMPVECTOR-$\alpha(W, \boldsymbol{x}, i, c)$ ▷ W: value of f_α, $\boldsymbol{x}$: subcomplement vector, i: depth i
2:                 ▷ Determine the number of dummy leaves to add below depth i ($\boldsymbol{w}$ is a global variable)
3:     $x_i \leftarrow w_i + c \bmod \alpha$ ▷ Initialization. Align with α according to Lemma 23
4:     **while** $x_i \leq w_i + c$ **and** $W + x_i/\alpha^i \leq 1$ **do**   ▷ For all x_i satisfying $x_i \leq w_i + c$
5:         $\boldsymbol{y} \leftarrow \boldsymbol{x} + (0, \ldots, 0, x_i, 0, \ldots)$ ▷ $\boldsymbol{x}$ meets the requirements of Lemma 23
6:         **if** $W + x_i/\alpha^i = 1$ **then**
7:             **Output $\boldsymbol{y}$**   ▷ Output as a full α-ary tree complement
8:             **return**
9:         **else**
10:             ENUMCOMPVECTOR$(W + x_i/\alpha^i, \boldsymbol{y}, i-1, (x_i + w_i)/\alpha)$ ▷ recurse on depth $i-1$
11:         **end if**
12:         $x_i \leftarrow x_i + \alpha$
13:     **end while**
14:     **return**
15: **end procedure**

7.2.2 Enumeration of the configuration of dummy leaves

Suppose that we have determined the complement vector $\boldsymbol{x}$ of $\boldsymbol{w}$. The next step is to determine the configuration of dummy leaves, which is (like for binary trees) not unique, in general. However, in the case of α-ary trees, among all siblings of a dummy leaf there must be at least one *normal node*, i.e., an internal node or a leaf that is not dummy. Therefore, we cannot infer from a dummy parent its number of dummy leaves, and thus specifying the configuration of dummy nodes with a Gray code like for binary trees seems not possible. Instead, we enumerate the number of dummy leaves that each dummy parent has from 0 to $\alpha - 1$, and then enumerate the configuration of dummy leaves corresponding to that number. This strategy is slightly more complicated than in the case of binary trees because of this additional intermediate step, but the calculation time is asymptotically absorbed by the time spent for determining one complement vector. Having computed the complement vector, the following steps are analogous to the case of binary trees, for which we obtain constant time delay per output.

We enumerate all possible ways to place the dummy leaves at each depth $\delta \in [1..d]$ by choosing a combination of the configuration of the dummy leaves at depth δ. For that, we first specify how many dummy leaves each parent has at depth $\delta - 1$. According to Corollary 19, in the complemented tree corresponding to the binary distribution $\boldsymbol{w} + \boldsymbol{x}$ with the added dummy leaves, there are $\eta_\delta = g_\alpha(\delta, \boldsymbol{w})$ nodes at depth δ (i.e., there are η_δ/α parents at depth $\delta - 1$). We assign each parent a rank from $[1..\eta_\delta/\alpha]$ in order from left to right, and we say that $p_i^{(\delta)}$ is the number of dummy leaves that the i-th parent has. Since each parent must contain at least one normal node, $p_i^{(\delta)} \in [0..\alpha - 1]$, and the number of dummy leaves at depth δ is x_δ, so $\boldsymbol{p}^{(\delta)} = (p_1^{(\delta)}, \dots, p_{\eta_\delta/\alpha}^{(\delta)})$ satisfies

$$\sum_{i=1}^{\eta_\delta/\alpha} p_i^{(\delta)} = x_i \,. \tag{1}$$

We use a known result for enumerating all distinct integer tuples $(p_1^{(\delta)}, \dots, p_{\eta_\delta/\alpha}^{(\delta)})$ with $p_i^{(\delta)} \in [0..\alpha - 1]$ satisfying Equation (1) with constant delay [16]. For each such integer tuple, we enumerate the configuration of dummy leaves corresponding to each $\boldsymbol{P}$. After that, all α-ary trees can be enumerated with constant time delay as in the case of binary trees.

8 Conclusion

We have addressed the problem to efficiently enumerate trees whose numbers of leaves on each depth are given by a query vector. Specifically, for any leaf distribution $\boldsymbol{w} = (w_1, \dots, w_d)$ of (full) binary or (full) α-ary trees, we obtain the following time complexities. First, we can output the first tree in $O(d + n)$ time. We can output every subsequent tree, after constant delay, in $O(m)$ time by encoding this tree as a difference to a previous one using $O(m)$ words. Here, $n = \sum_{i=1}^d w_i$ is the total number of nodes and $m = \max\{w_i \mid i \in [1..d]\}$ is the maximal number of nodes at any depth. These results hold for both full binary trees and full α-ary trees with $\alpha \geq 2$, and extend to general binary trees with analogous enumeration guarantees.

As future work, we want to determine the exact number of full binary trees and full α-ary trees with distribution $\boldsymbol{w}$ by a closed-form product formulas involving binomial coefficients. We further strive to give practical implementations of our proposed enumeration algorithms.

───── **References** ─────

1   Masahiro Baba, Hirotaka Ono, Kunihiko Sadakane, and Masafumi Yamashita. A succinct representation of a full binary tree. Technical Report 0, Record of Joint Conference of Electrical and Electronics Engineers in Kyushu, 2009. `doi:10.11527/jceeek.2009.0.59.0`.

2 David Benoit, Erik D. Demaine, J. Ian Munro, Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Representing trees of higher degree. *Algorithmica*, 43(4):275–292, 2005. `doi:10.1007/s00453-004-1146-6`.

3 Terry Beyer and Sandra Mitchell Hedetniemi. Constant time generation of rooted trees. *SIAM J. Comput.*, 9(4):706–712, 1980. `doi:10.1137/0209055`.

4 Michael Buro. On the maximum length of Huffman codes. *Inf. Process. Lett.*, 45(5):219–223, 1993. `doi:10.1016/0020-0190(93)90207-P`.

5 Jeremy Chizewer, Stephen Melczer, J. Ian Munro, and Ava Pun. Enumeration and succinct encoding of AVL trees. In *Proc. AofA*, volume 302 of *LIPIcs*, pages 2:1–2:12, 2024. `doi:10.4230/LIPICS.AOFA.2024.2`.

6 Nachum Dershowitz and Shmuel Zaks. Enumerations of ordered trees. *Discret. Math.*, 31(1):9–28, 1980. `doi:10.1016/0012-365X(80)90168-5`.

7 Sen-Peng Eu, Seunghyun Seo, and Heesung Shin. Enumerations of vertices among all rooted ordered trees with levels and degrees. *Discret. Math.*, 340(9):2123–2129, 2017. `doi:10.1016/J.DISC.2017.04.007`.

8 Frank Gray. Pulse code communication. US Patent 2632058, 1953.

9 Roberto Grossi. Enumeration of paths, cycles, and spanning trees. In *Encyclopedia of Algorithms*, pages 640–645. Springer, 2016. `doi:10.1007/978-1-4939-2864-4_728`.

10 Guy Jacobson. Space-efficient static trees and graphs. In *Proc. FOCS*, pages 549–554, 1989. `doi:10.1109/SFCS.1989.63533`.

11 Leon Gordon Kraft. A device for quantizing, grouping, and coding amplitude-modulated pulses. Master's thesis, Massachusetts Institute of Technology, 1949.

12 Shin-Ichi Nakano and Takeaki Uno. Constant time generation of trees with specified diameter. In *Proc. WG*, volume 3353 of *LNCS*, pages 33–45, 2004. `doi:10.1007/978-3-540-30559-0_3`.

13 Frank Ruskey and Andrzej Proskurowski. Generating binary trees by transpositions. *J. Algorithms*, 11(1):68–84, 1990. `doi:10.1016/0196-6774(90)90030-I`.

14 Ivan Stojmenovic. A simple systolic algorithm for generating combinations in lexicographic order. *Computers & Mathematics with Applications*, 24(4):61–64, 1992.

15 Takeaki Uno. Two general methods to reduce delay and change of enumeration algorithms. Technical Report NII-2003-004E, NII Technical Report, 2003. `doi:10.20736/0000000385`.

16 Timothy R. Walsh. Loop-free sequencing of bounded integer compositions. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 33:323–345, 2000.

17 Katsuhisa Yamanaka, Yota Otachi, and Shin-Ichi Nakano. Efficient enumeration of ordered trees with k leaves. *Theor. Comput. Sci.*, 442:22–27, 2012. `doi:10.1016/j.tcs.2011.01.017`.

18 Shmuel Zaks. Lexicographic generation of ordered trees. *Theor. Comput. Sci.*, 10:63–82, 1980. `doi:10.1016/0304-3975(80)90073-0`.

Appendix

Table 1 Symbols used in this article.

symbol	meaning
$\boldsymbol{w}$	distribution
d	maximal depth and dimension of $\boldsymbol{w}$
n	number of leaves
m	maximum value in $\boldsymbol{w}$
$\boldsymbol{x}$	complement vector
$\boldsymbol{y}$	subcomplement vector
$f(\boldsymbol{w})$	$\sum_{i=1}^{d} w_i/2^i$ weight function
$\boldsymbol{\eta}$	$\eta_i := g(i, \boldsymbol{w})$
ν_i	$g(i, \boldsymbol{w} + \boldsymbol{x})$
$g(i, \boldsymbol{w})$	$\sum_{j=i}^{d} w_i/2^{j-i}$ for $i \in [1..d]$