Faster Path Queries in Colored Trees via Sparse Matrix Multiplication and Min-Plus Product

Younan Gao
Faculty of Computer Science, Dalhousie University, Halifax, Canada

Meng He
Faculty of Computer Science, Dalhousie University, Halifax, Canada

Abstract

Let T be an ordinal tree on n nodes in which each node is assigned a color. We consider the batched colored path counting problem and the batched path mode/least frequent element query problem, in which given n query paths, each identified by a pair of nodes in T, one is asked to answer queries of the following forms: How many distinct colors are there on each query path (i.e. the colored path counting problem); what is the color on each query path that occurs at least/most as frequently as any other colors (i.e. the path mode/least frequent element query problem). By reducing the batched colored path counting problem to sparse matrix multiplication, we design a solution that answers n colored path counting queries in $\tilde{O}(n^{\frac{2\omega}{\omega+1}}) = O(n^{1.40704})$ time in total, while we reduce batched path mode/least frequent element query to the min-plus-query-witness problem so that we can answer a batch of n queries in $\tilde{O}(n^{\frac{24+2\omega}{17+\omega}}) = O(n^{1.483814})$ time[1]. Previously, both problems could only be solved in $\tilde{O}(n^{1.5})$ time.

Based on similar techniques, we design a dynamic colored path counting structure supporting both queries and updates in $\tilde{O}(n^{\frac{\omega+1}{\omega+3}}) = O(n^{0.627759})$ time, while our dynamic path mode/least frequent element query structures support each operation in $\tilde{O}(n^{\frac{16+\omega(1,2,1)}{26+\omega(1,2,1)}}) = O(n^{0.658139})$ time, where $\omega(1,2,1)$ denotes the minimum value such that the product of an $n \times n^2$ matrix and an $n^2 \times n$ matrix can be computed in $O(n^{\omega(1,2,1)+\epsilon})$ time for any constant $\epsilon > 0$. We also solve batched range mode/least frequent element query problems over arrays in $\tilde{O}(n^{\frac{18+2\omega}{13+\omega}}) = O(n^{1.479603})$ time. Both problems can be viewed as special cases of these batched path queries, and previously, the fastest algorithm for batched range mode queries and batched range least frequent element queries use $O(n^{1.4805})$ and $\tilde{O}(n^{1.5})$ time, respectively.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Information systems → Data structures

Keywords and phrases min-plus product, range mode queries, range least frequent queries, path queries, colored path counting, path mode queries, path least frequent queries

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.59

Funding This work was supported by NSERC of Canada.

Acknowledgements The authors would like to thank the anonymous reviewers for their valueable comments and suggestions.

1 Introduction

Trees are used to represent information in many areas of computer science. In tree-structured data, additional properties such as categorical information are often encoded as colors of tree nodes. To facilitate the retrieval of color information, researchers have defined the following queries over an ordinal tree T on n nodes with each node assigned a color from

[1] The $\tilde{O}$ notation hides the `polylog`(n) factors, e.g., $O(n \lg^2 n) = \tilde{O}(n)$, and ω denotes the best possible exponent of square matrix multiplication.

$\{0, 1, \ldots, C-1\}$, where $C \leq n$: Given a path in T, a *path colored counting query* returns the number of distinct colors assigned to the nodes in this path, while a *path mode query* or a *path least frequent element query* returns the most frequent or the least frequent color among the multiset of colors assigned to nodes in this path, respectively[2]. The color that is assigned to the most number of nodes in a path is called the *mode* of the path. These queries can be used to compute fundamental statistic information over tree-structured data.

Researchers have studied these query problems and designed data structure solutions [32, 11, 21]. Different time-space tradeoffs have been achieved, and among the best linear-space solutions under word RAM, the structure of He and Kazi [21] can answer a colored path counting query in $O(\sqrt{n} \lg\lg C)$ time, while the solutions of Durocher et al. [11] can answer a path mode query or a path least frequent element query in $O(\sqrt{n/w} \lg\lg n)$ time, where w denotes the number of bits stored in a word. The support for these queries is thus much slower than the support for many other path queries in trees such as path minimum [9, 2, 30, 10, 5, 8], path medium [32, 35, 24, 25], path counting [9, 32, 35, 24, 25] and path majority [11, 13], for which linear-space solutions with sublogarithmic or even constant query times exist.

However, researchers have given evidence to show that these solutions to colored path counting, path mode and path least frequent element are efficient, by proving conditional lower bounds. It has been shown that the multiplication of two $\sqrt{n} \times \sqrt{n}$ Boolean matrices can be performed by answering n colored path counting queries, n path mode queries or n path least frequent element queries. This reduction was explicitly given for colored path counting [21], while for the other two path queries, it follows from the same conditional lower bound on range mode queries [6] and range least frequent element queries [7] in arrays, for which we preprocessing an array A, such that, given a range $[i, j]$, we can find the most frequent or least frequent element in $A[i, j]$ efficiently. Note that when the given tree has a single path only, path mode and path least frequent element queries become range mode and range least frequent element queries, respectively. This reduction means, with current knowledge, the total running time of answering n of these path or range queries, including preprocessing, cannot be faster than $n^{\omega/2}$, save for polylogarithmic speedups, where $\omega < 2.37286$ denotes the exponent of matrix multiplication [1]. Furthermore, since the best known combinatorial approach of multiplying two $n \times n$ Boolean matrices require $\Theta(n^3/\texttt{polylog}(n))$ time [38], the total time of answering n of these queries cannot be faster than $n^{1.5}$, save for polylogarithmic speedups, using pure combinatorial methods with current knowledge. Since the structures of He and Kazi [21] and Durocher et al. [11] can be built in $\tilde{O}(n^{1.5})$ time, they can be used to answer n colored path counting or path mode/least frequent element queries in $\tilde{O}(n^{1.5})$ time, matching this conditional lower bound on pure combinatorial methods within polylogarithmic factors.

The problem of answering n queries given offline is the batched version of these query problems. To achieve $O(n^{1.5-\epsilon})$-time solutions for some positive constant ϵ, Williams and Xu [37] reduced batched range mode to the min-plus product of a pair of matrices of special structures, which makes it possible to answer a batch of n range mode queries over an array of length n in $O(n^{1.4854})$ time. Gu et al. [17] further improved the running time to $O(n^{1.4805})$. Similar ideas have also yielded dynamic range mode structures with $O(n^{0.655994})$ query and update times [17]. This is surprising as Jin and Xu [27] showed that dynamic range mode

[2] These problems can also be defined over free trees. However, we follow the definitions given in previous work [11, 21] and assume that T is an ordinal tree, as this allows us to directly apply previous solutions to the problems defined over ordinal trees.

structure cannot simultaneously support update and query in $O(n^{2/3-\epsilon})$ time for any positive constant ϵ using purely combinatorial method with current knowledge. Even before that, Kaplan et al. [31] used sparse matrix multiplication to answer *n 2D orthogonal colored range counting queries* over n colored points on the plane in $\tilde{O}(n^{\frac{2\omega}{\omega+1}})$ total time. This query counts the number of distinct colors assigned to points in an axis-aligned query rectangle and is related to path colored counting in the sense that they both generalize 1D colored range counting [14, 34] and have the same conditional lower bound.

Despite all these exciting works which beat the conditional lower bounds for combinatorial methods, no previous work was done to solve batched colored path counting or batched path mode in $O(n^{1.5-\epsilon})$-time. No similar results were achieved for batched range least frequent queries in arrays or batched path least frequent element, either. Therefore, we use matrix multiplication and min-plus product to solve these problems and their dynamic versions.

1.1 Previous Work

The study on colored range counting started in the 1D case, for which Gupta et al. [19] showed a reduction to 2D orthogonal range counting over uncolored points, hence achieving a linear space solution with $O(\lg n/\lg\lg n)$ query time. This problem has been studied extensively in 2D [18, 31, 16, 33, 15], for which Kaplan et al. [29] proved the conditional lower bound based on Boolean matrix multiplication which we discussed previously. They further designed a data structure occupying $O((\frac{n}{t})^2 \lg^6 n + n\lg^4 n)$ words that supports 2D colored range counting in $O(t\lg^7 n)$ time for any $0 < t \leq n$, which was later improved by Gao and He [15] who shaved off several $\log n$-factors in time/space bounds. Kaplan et al. also showed how to solve batched 2D orthogonal colored range counting in $\tilde{O}(n^{\frac{2\omega}{\omega+1}}) = O(n^{1.40704})$ time by reducing it to sparse matrix multiplication. Recently, Jin and Xu [27] presented a dynamic 2D orthogonal colored range counting structure with $\tilde{O}(n^{2/3})$ query and update times. Colored range counting has also been studied in high dimensions [18, 31, 16]. Finally, He and Kazi [21] considered colored path counting in trees and proved a conditional lower bound which is also based on Boolean matrix multiplication. They designed an $O(n + \frac{n^2}{t^2})$-word structure that answers queries in $O(t\lg\lg C)$ time for any $t \in [1, n]$, and it can be constructed in $O(\frac{n^2}{t}\lg\lg C)$ time. Hence it implies an $O(n^{3/2}\lg\lg C)$-time solution to batched colored path counting.

Since Krizanc et al. [32] proposed range and path mode query problems, a long series of papers have been published on these and related problems [32, 6, 7, 12, 37, 36, 17, 27]. The best linear-space solutions include the structure of Chan et al. [6] that answers range mode queries in arrays in $O(\sqrt{n/w})$ time, the structure of Durocher et al. [11] that answers range least frequent element queries in arrays in $O(\sqrt{n/w})$ time, and the structures of Durocher et al. [11] that answer path mode / least frequent elements in trees in $O(\sqrt{n/w}\lg\lg n)$ time. Chan et al. [6] also studied dynamic range mode in arrays, and their solution was later improved by El-Zein et al. [12], whose linear-space structure supports both queries and updates in $O(n^{2/3})$ time. El-Zein et al. also designed a linear-space structure supporting range least frequent element in $O(n^{2/3}\lg n\lg\lg n)$ time and updates in $O(n^{2/3})$ time; different from the original definition of range least frequent, the query is allowed to return a color that does not appear in the query range but appears elsewhere in the array. A Monte Carlo structure is designed in the dynamic case for the original range least frequent element query. It is worth mentioning that all the results summarized in this paragraph use purely combinatorial approaches. They match the conditional lower bounds [6, 7, 27] within polylogarithmic factors.

Recently, more efficient solutions to the batched range mode problem in arrays [36, 37, 17] have been found. Williams and Xu [37] reduced this problem to the min-plus product of a pair of matrices. The second matrix has the property that the entries at each row are non-decreasing, which allows designing a truly subcubic time algorithm for min-plus product of two $n \times n$ matrices. With it, they can solve batched range mode in $O(n^{1.4854})$ time. Later, the query time was improved by Gu et al. [17] to $O(n^{1.4805})$. Sandlund and Xu [36] broke the $O(n^{2/3})$ per-operation time barrier for dynamic range mode in arrays; they reduced the problem to the min-plus-query-witness problem, and achieved a dynamic data structure that supports both queries and updates in $O(n^{0.655994})$ time. Later, Gu et al. [17] further improved the time for each operation to $O(n^{0.6524})$.

1.2 Our Contributions

Our Results. We have achieved the following results:

- an $\tilde{O}(n^{\frac{2\omega}{\omega+1}}) = O(n^{1.40704})$-time algorithm for batched colored path counting in trees, improving the previous best approach which solves this problem in $\tilde{O}(n^{1.5})$ time [21];
- an $\tilde{O}(n^{\frac{24+2\omega}{17+\omega}}) = O(n^{1.483814})$-time algorithm for batched path mode/least frequent element queries in trees, improving the previous best result with $\tilde{O}(n^{1.5})$ running time [11];
- an $\tilde{O}(n^{\frac{18+2\omega}{13+\omega}}) = O(n^{1.479603})$ time algorithm for batched range mode/least frequent element queries in arrays, while the previous best results solve batched range mode in $O(n^{1.4805})$ time [17] and batched range least frequent element in $\tilde{O}(n^{1.5})$ time [11];
- a dynamic colored path counting structure for trees supporting queries and updates in $\tilde{O}(n^{\frac{\omega+1}{\omega+3}}) = O(n^{0.627759})$ time, while the best dynamic structure for 2D orthogonal colored range counting, as a related problem, supports each operation in $\tilde{O}(n^{2/3})$ time [27];
- a dynamic data structure for path mode/least frequent element queries in trees supporting queries and updates in $\tilde{O}(n^{\frac{16+\omega(1,2,1)}{26+\omega(1,2,1)}}) = O(n^{0.658139})$ worst-case time. Here, $\omega(1,2,1)$ denotes the minimum value such that the product of an $n \times n^2$ matrix and an $n^2 \times n$ matrix can be computed in $O(n^{\omega(1,2,1)+\epsilon})$ time for any constant $\epsilon > 0$. These bounds are close to the $O(n^{0.6524})$ query and update times for dynamic range mode [17] which can be viewed as a special case of dynamic path mode, while the previous best result for dynamic range least frequent element supports queries and updates in $\tilde{O}(n^{2/3})$ time [12].

Overview of Our Approach. To achieve these results, we develop new algorithmic ideas to address the challenges we encounter due to the tree topology. The first challenge is how to apply a divide-and-conquer approach to batched path queries. The solution of Williams and Xu [37] to batched range mode recursively divides the input array into halves, and at each level of recursion, they build data structures to answer queries whose ranges straddle the midpoint. This ensures that the set of possible queries considered share subranges instead of being disjoint, facilitating preprocessing. The solution of Kaplan et al. [31] to batched 2D orthogonal colored range counting is based on a similar idea in 2D. To adapt to tree topology, we apply the centroid decomposition of trees recursively instead. Then, in each component obtained as a result of the decomposition, we preprocess for queries whose paths cross the centroid. This decomposition scheme helps us solve all three batched path queries.

When preprocessing for query paths that contain the centroid in a component, we mark a subset of nodes and attempt to use either sparse matrix multiplication or min-plus product as in previous work. However, more twists to previous approaches are needed. In the solution of Kaplan et al. [31] to batched 2D orthogonal colored counting, the matrices that they need to multiply during preprocessing are already sparse. This is however not the case in our solution to batched path colored counting. To resolve this, we use the properties of our node marking scheme to carefully reduce the problem of multiplying these matrices to the multiplication of two different but related matrices that are sparse. There is a similar challenge for batched path mode. Previous solutions to batched range mode in [36, 17] reduces the preprocessing for each set of query ranges to the min-plus-query-witness problem over two matrices of which the second matrix is monotone, i.e., entries in the same row are non-decreasing.This allows applying strategies such as dividing each entry by a carefully chosen integer and rounding down the result to decrease the number of different entries in the matrix. In our case, due to the tree topology, the second matrix is not monotone, so this way of applying integer division fails to decrease the number of different entries. To resolve this issue, we design a two-tier scheme to mark nodes for preprocessing, and use the properties of this marking scheme to reduce the weights of entries of the second matrix using different formulas depending on at which tier the corresponding tree nodes are marked. This allows us to decrease the number of different entries to speed up preprocessing.

In range mode queries over arrays, the input elements are split into two categories, i.e., frequent elements and infrequent elements, based on the frequency of each distinct element, and the elements in the different categories are processed in different ways. As mentioned above, the min-plus-query-witness problem was considered in [36, 17] to solve dynamic range mode, and it is used to handle frequent elements. Naturally, a data structure that solves dynamic range mode on arrays can answer static range mode queries as well. By combining an existing solution to the min-plus-query-witness problem shown in [17, Lemma 33] and a static data structure that handles infrequent elements shown in [37, Proof of Theorem 6.1], we design a static data structure for range mode queries on arrays. This simple combination leads to a new static data structure with faster preprocessing time and query time.

In this paper, we describe our solutions to batched colored path counting and batched path mode queries to show the details of these ideas. Due to space constraints, our solutions to other problems such as batched range mode queries in arrays, batched range least frequent element queries in arrays, dynamic path colored counting, dynamic path mode queries, and dynamic path least frequent queries are deferred to the full version of this paper.

2 Preliminaries

This section introduces the notation and the previous results used in this paper.

Notation. Given an ordinal tree T, let $|T|$ denote the number of nodes in T, and let $\perp$ denote its root. For any two nodes $x, y \in T$, we use $P_{x,y}$ to represent the path whose endpoints are x and y. Thus, $P_{x,\perp}$ is a root-to-node path. If y is an ancestor of x, then $P'_{x,y}$ is defined to be the path whose endpoints are x and the child of y that is an ancestor of x, i.e., $P'_{x,y} = P_{x,y} \setminus \{y\}$. Furthermore, $c(x)$ denotes the color assigned to x, and $C(P_{x,y})$ (or $C(P'_{x,y})$) denotes the set of colors that appear in $P_{x,y}$ (or $P'_{x,y}$). Finally, T_x refers to the subtree of T rooted at node x, and `parent`(x) is the parent of x.

Navigation in colored ordinal trees. Regarding each tree node color as integer label, the input tree, studied in this paper, is both an ordinal tree and a labeled tree. To support the basic navigational operations on it, we apply the succinct representation of ordinal trees by [22] and the result of He et al. [23] on labeled tree representations. The following lemma summarizes the operations used in our solution and the complexity. Following their notation, we call a node (resp. ancestor) assigned color α an α-node (resp. α-ancestor).

▶ **Lemma 1** ([22, 23]). *Let T be an ordinal tree on n nodes with each node assigned a color from $\{0, 1, \dots, C-1\}$, where $C \le n$. A data structure occupying $n \lg C + 2n + o(n \lg C)$ bits can be built over T in $O(n)$ time to support:*

- $\texttt{depth}_\alpha(x)$ *in $O(\lg\lg C)$ time, which returns the number of α-ancestors of node x;*[3]
- $\texttt{depth}(x)$ *in $O(1)$ time, which returns the number of ancestors of x;*
- $\texttt{LCA}(x, y)$ *in $O(1)$ time, which returns the lowest common ancestor of nodes x and y.*

Given a query path $P_{x,y}$ and a color α in a tree T represented by Lemma 1, He and Kazi [21] showed how to use $\texttt{depth}_\alpha$ and $\texttt{LCA}$ to compute the number of appearances of α in $P_{x,y}$ in $O(\lg\lg C)$ time. This implies the support of *colored path emptiness*, which asks whether a color α appears in $P_{x,y}$. We can further use it to compute $\{|C(P_{x,\perp})| : x \in T)\}$, i.e., the numbers of distinct colors on all root-to-node paths. To do this, perform a preorder traversal of T, and each time we visit a node x, we compute $|C(P_{x,\perp})|$ as follows: If x is the root, then $|C(P_{x,\perp})|$ is 1. Otherwise, locate $x' = \texttt{parent}(x)$, and answer a colored path emptiness query to find out whether $c(x)$ appears in $P_{x',\perp}$. If it does, set $|C(P_{x,\perp})| = |C(P_{x',\perp})|$; otherwise, $|C(P_{x,\perp})| = |C(P_{x',\perp})| + 1$. This process uses $O(n \lg\lg C)$ time.

Node sampling. In our solutions, we use the following lemma based on the pigeonhole principle to select a subset of tree nodes and precompute information for them.

▶ **Lemma 2** ([26]). *Let T be a tree on n nodes whose height is h. Given an integer $t \in [1, n]$, an integer $\ell \in [0, t-1]$ can be found in $O(n)$ time such that the total number of nodes whose depths are $\ell + i \times t$ for some $i \in [0, \lfloor \frac{h-\ell}{t} \rfloor]$ is at most n/t.*

Sparse rectangular matrix multiplication. We use the following result of Kaplan et al. [31]:

▶ **Lemma 3** ([31, Theorem 2.5]). *Let A be an $m \times n$ matrix having at most t non-zero entries, where $t \ge m^{\frac{\omega+1}{2}}$. Then, given the list of non-zero entries of A as the input without storing A verbatim, the product of A and the transpose of A can be computed in $O(tm^{\frac{\omega-1}{2}})$ time.*

3 Batched Colored Path Counting

We first design a data structure to answer a restricted version of colored path counting which requires the query path to contain the root (Section 3.1). Then we generalize it to handle arbitrary paths, which yields a new result for batched colored path counting (Section 3.2).

3.1 Color Counting over Paths Containing the Root

Data structures. To design a data structure for a restricted version of colored path counting which requires the query path to contain the root, we first represent the tree T using Lemma 1. As discussed in Section 2, this structure supports colored path emptiness. Then, for an integer parameter $0 < X \le n$ to be chosen later, we select at most n/X nodes of T using Lemma 2 and mark them. This means we mark nodes at every X levels of T, starting from some level $\ell \in [0, X-1]$ determined by Lemma 2. In addition, we mark the root node as

[3] Note that the structure of He et al. [23] can support $\texttt{depth}_\alpha(x, i)$ in $O(\lg \frac{\lg C}{\lg w})$ time, faster than what is stated in Lemma 1. However, this requires a string representation with support for rank and select [4] which uses perfect hashing, and it is not known how to construct this structure in $O(n)$ deterministic time. Therefore, we swap it with the string representation of Belazzougui et al. [3] which can be constructed in linear deterministic time and achieve the bounds in Lemma 1.

well. This means the number, m, of nodes that we mark satisfies $m \leq n/X + 1$. We refer to the i-th marked node visited in a preorder traversal as the *i-th marked node* for short, where i starts from 0, and this node is denoted by x_i. Thus, x_0 is the root. For each marked node x, we precompute $r(x)$ which is the rank of x among marked nodes defined this way, as well as the value $|C(P_{x,\perp})|$. Furthermore, each node in the tree stores a flag indicating whether it is marked, as well as a pointer to the lowest marked proper ancestor.

Next, we construct an $m \times m$ matrix M. For every pair of integers i and j in $[0, m-1]$, $M[i,j]$ stores $|C(P_{x_i,\perp}) \cap C(P_{x_j,\perp})|$, i.e., the number of distinct colors that appear in both the path between the i-marked node and the root and the path between j-th marked node and the root. It is worth mentioning that our query algorithm to be described later only uses entries of M that correspond to two marked nodes whose lowest common ancestor is the root. The other entries are never used, but we precompute them regardless.

Overall, our data structures use $O(n + (\frac{n}{X})^2)$ words.

Query algorithm. To describe the query algorithm, let $P_{x,y}$ denote a query path containing the root. Since $\perp \in P_{x,y}$, we always have $\texttt{LCA}(x,y) = \perp$. W.l.o.g., we assume that neither x nor y is the root node. Let x' and y' denote the lowest marked ancestors of x and y, respectively, and we divide the query path $P_{x,y}$ into three disjoint subpaths: $P'_{x,x'}$, $P_{x',y'}$ and $P'_{y,y'}$. Following the inclusion-exclusion principle, we have that $|C(P_{x',y'})| = |C(P_{x',\perp})| + |C(P_{\perp,y'})| - M[r(x'), r(y')]$. Since the three terms at the right-hand side of this formula have all been precomputed, $|C(P_{x',y'})|$ can be computed in constant time. Next, we count the number of distinct colors that appear in $P'_{x,x'}$ but not in $P_{x',y'}$. This can be done by iterating through each node z in $P'_{x,x'}$ in the direction towards x and check whether $c(z)$ appears in $P_{\texttt{parent}(z),y'}$ by performing a path emptiness query. The number of distinct colors that are in $P'_{y,y'}$ but not in $P_{x,y'}$ can be counted in a similar way. Adding these two counts to $|C(P_{x',y'})|$ yields the answer. Since x (resp. y) and x' (resp. y') are at most $X-1$ levels apart, the query time is $O(X \lg\lg C)$. This query algorithm is adapted from an algorithm of He and Kazi [21] for arbitrary query paths, though we use a different matrix.

Preprocessing. Lemmas 1 and 2, together with the discussions on how to compute $\{|C(P_{x,\perp})| : x \in T\}$ in Section 2, can be applied to construct all our data structure components except the matrix M in $O(n \lg\lg C)$ time. To compute M, one way is to define an $m \times C$ matrix A, in which entry $A[i,\alpha] = 1$ if color $\alpha \in C(P_{x_i,\perp})$, and it is 0 otherwise. Then we compute M as AA^T, where A^T denotes the transpose of A. However, since C can be as large as n, the multiplication of A and A^T can be costly.

Instead, to compute M, we use the computation of two related but different $m \times m$ matrices as stepping stones, and one of these two, which we call $\hat{M}$, can be computed via sparse rectangular matrix multiplication [31]. Before defining $\hat{M}$, we introduce more notation. Let a be a marked node and b its lowest marked proper ancestor. We define $\hat{C}(a) = C(P'_{a,b}) \setminus C(P_{b,\perp})$, i.e., the set of colors that appear on the path from a to b (including a but excluding b) but not on the path between and including b and the root. Since there are at most X nodes in $P'_{a,b}$, we have $|\hat{C}(a)| \leq X$.

With these definitions, we can define $\hat{M}$ as an $m \times m$ matrix, in which, for each pair of integers $i, j \in [0, m-1]$, $\hat{M}[i,j]$ stores $|\hat{C}(x_i) \cap \hat{C}(x_j)|$. To compute $\hat{M}[i,j]$, we define an $m \times C$ matrix $\hat{A}$, and for each integer $i \in [0, m-1]$ and each color $\alpha \in [0, C-1]$, $\hat{A}[i,\alpha]$ is set to be 1 if $\alpha \in \hat{C}(x_i)$ and 0 otherwise. Then $\hat{M} = \hat{A}\hat{A}^T$. Since each row of $\hat{A}$ indicates whether each color appears in the set $\hat{C}(a)$ for some marked node a, each row has at most X non-zero entries. Since $\hat{A}$ has m rows and $m \leq n/X + 1$, overall $\hat{A}$ has at most $n + X \leq 2n$

non-zero entries only. A list of non-zero entries of $\hat{A}$ can be computed in $O(n)$ time; the details are omitted due to page limit. With these non-zero entries as input, we can apply Lemma 3 to compute $\hat{M}$ in $O(n^{(\omega+1)/2}/X^{(\omega-1)/2})$ time for any $X \in [n^{(\omega-1)/(\omega+1)}, n]$.

The other $m \times m$ matrix that is used to help us compute M is called M'. For each pair of integers $i, j \in [0, m-1]$, entry $M'[i,j]$ stores $|\hat{C}(x_i) \cap C(P_{x_j,\perp})|$. To compute M', we perform a preorder traversal of T. Each time we visit a marked node x_j, we compute the j-th column of M' as follows: If $x_j = \perp$, then $j = 0$ and, for each $i \in [0, m-1]$, we set entry $M'[i, 0]$ to be 0, since $c(\perp) \notin \hat{C}(x_i)$. If x_j is not the root, we locate the lowest marked proper ancestor, y, of x_j. Since y is visited before x_j, $M'[i, r(y)]$ has already been computed, and we set $M'[i,j] = M'[i, r(y)] + \hat{M}[i,j]$. To see the correctness, observe that $M'[i,j] = |\hat{C}(x_i) \cap C(P_{x_j,\perp})| = |\hat{C}(x_i) \cap (C(P_{y,\perp}) \cup \hat{C}(x_j))|$; since $C(P_{y,\perp}) \cap \hat{C}(x_j) = \emptyset$, this is equal to $|\hat{C}(x_i) \cap C(P_{y,\perp})| + |\hat{C}(x_i) \cap \hat{C}(x_j)| = M'[i, r(y)] + \hat{M}[i,j]$. This way we can compute M' in $O(n + (\frac{n}{X})^2)$ time provided that $\hat{M}$ is available.

After computing $\hat{M}$ and M', we can compute M by performing another preorder traversal of T. Each time we visit a marked node x_i, we compute the i-th row of M as follows: If $x_i = \perp$, then $M[i,j] = 1$ for any $j \in [0, m-1]$. Otherwise, we locate the lowest marked proper ancestor, y, of x_i. Since y is visited before x_i, $M[r(y), j]$ has already been computed, and we set $M[i,j] = M[r(y), j] + M'[i,j]$. To see the correctness, observe that $M[i,j] = |C(P_{x_i,\perp}) \cap C(P_{x_j,\perp})| = |(C(P_{y,\perp}) \cup \hat{C}(x_i)) \cap C(P_{x_j,\perp})|$; since $C(P_{y,\perp}) \cap \hat{C}(x_i) = \emptyset$, this is equal to $|C(P_{y,\perp}) \cap C(P_{x_j,\perp})| + |\hat{C}(x_i) \cap C(P_{x_j,\perp})| = M[r(y), j] + M'[i,j]$. In this way, we can compute M in $O(n + (\frac{n}{X})^2)$ time after computing $\hat{M}$ and M'. The total preprocessing time is hence $O(n \lg\lg C + (\frac{n}{X})^2 + \frac{n^{(\omega+1)/2}}{X^{(\omega-1)/2}})$ for any integer $X \in [n^{\frac{\omega-1}{\omega+1}}, n]$.

3.2 Color Counting on an Arbitrary Path

We now generalize the structure in the previous section to support queries over arbitrary paths. Our strategy is to decompose T recursively using *centroid decomposition* [28]; a *centroid* of an n-node tree is a node whose removal splits the tree into connected components each containing at most $n/2$ nodes, and this node can be found in $O(n)$ time.

At level 0 of the recursion, the given tree T is a connected component by itself and we call it the level-0 component. We find a centroid, u, of T, and define a new rooted tree T^u by designating u as the root of T, reorienting edges when necessary. Then we build the query structure in Section 3.1 over T^u. Afterwards, we remove u from T, and build our data structure recursively over each connected component that has more than X nodes. In general, at the i-th level of the recursion, we have a set of connected components called *level-i components* obtained by removing from T the centroids computed in previous levels of the recursion. For each component, we compute its centroid and designate the centroid as the root of this component to build the query structure of Section 3.1. One minor detail is that, before building the query structure over a component of size n', we need to ensure that colors are encoded as nonnegative integers less than n'. Thus, when $n' \leq C$, we sort the colors that appear in this component using integer sorting in $O(n' \lg\lg n') = O(n' \lg\lg C)$ time [20] and re-encode these colors using their ranks. Then we remove the centroid of each level-i component to split it into a set of level-$(i+1)$ components and recurse. When a component has at most X nodes, we no longer apply this recursive procedure to it, and we call it a *base component.* Thus, we have $O(\lg(n/X))$ recursion levels.

In addition, for each node $x \in T$, we store a list of $O(\lg(n/X))$ pointers, and the i-th pointer maps x to its copy in a level-i component; this pointer is a null if x is removed as a centroid node found in a previous level. Furthermore, we build a weighted tree T' by assigning weights to the nodes of T as follows: If a node x is chosen as the centroid node of a level-i component, its weight is i. If x is never chosen as a centroid, then its weight is ∞. Then we construct the linear-space data structure of Chan et al. [8, Theorem 1.1] to support *path minimum queries* over T' in constant time; a path minimum query returns the node with minimum weight in a given query path.

Since we have $O(\lg(n/X))$ recursion levels, both the space costs and construction time of this new structure is a factor of $O(\lg(n/X))$ more than those of the structure in Section 3.1; the detailed analyses are deferred to the full version of this paper. To answer a query with $P_{x,y}$ as the query path, query T' to find the smallest weight, s, assigned to nodes in $P_{x,y}$. If $s = \infty$, then x and y are in the same base component, and thus $P_{x,y}$ has at most X vertices. We can traverse $P_{x,y}$ in T to count the number of distinct colors; each time we visit a new node, we perform a path emptiness query to determine whether we have already encountered this color. This way we can answer the query in $O(X \lg\lg C)$ time. Otherwise, observe that no nodes in $P_{x,y}$ are chosen as centroids for recursion level $s-1$ or smaller. Therefore, x and y must reside in a same level-s component S, and $P_{x,y}$ contains the centroid of S. Since this centroid is designated as the root of S before building the query structure of Section 3.1 over S, we can use this query structure to answer the query in $O(X \lg\lg C)$ time. Thus we have:

▶ **Lemma 4.** *Let T be a colored ordinal tree on n nodes with each node assigned a color from $\{0, 1, \ldots, C-1\}$, where $C \leq n$, and let X be an arbitrary integer in $[n^{\frac{\omega-1}{\omega+1}}, n]$. A data structure of $O((\lg \frac{n}{X})(n + \frac{n^2}{X^2}))$ words can be constructed in $O((\lg \frac{n}{X})(n \lg\lg C + (\frac{n}{X})^2 + \frac{n^{(\omega+1)/2}}{X^{(\omega-1)/2}}))$ time to support colored path counting query in $O(X \lg\lg C)$ time.*

We finally solve the batched colored path counting problem by first building the query structure of Lemma 4 and then using it to answer n queries. Setting $X = n^{\frac{\omega-1}{\omega+1}}$ yields:

▶ **Theorem 5.** *A batch of n colored path counting queries over a colored tree T on n nodes can be answered in $\tilde{O}(n^{\frac{2\omega}{\omega+1}}) = O(n^{1.40704})$ time in total.*

4 Batched Path Mode Queries

To solve batched path mode queries over tree T, let t_1 and t_2 be two constant parameters such that $0 \leq t_2 \leq t_1 \leq 1$. We categorize node colors into two different types: a color α is an *infrequent color* if it is assigned to at most n^{1-t_1} nodes in T; otherwise, we call it a *frequent color*. Thereby, a mode of a query path could be either a frequent or an infrequent color. We use the following lemma to find the most frequent element in the multiset of infrequent colors assigned to the nodes in a query path $P_{x,y}$; due to space constraints, we defer the proof to the full version of this paper.

▶ **Lemma 6.** *An $O(n^{2-t_1} \lg^4 n)$-word structure can be constructed in $O(n^{2-t_1} \lg^5 n)$ time, such that, given a query path $P_{x,y}$, the most frequent element and its frequency in the multiset of infrequent colors assigned to the nodes in $P_{x,y}$ can be computed in $O(n^{1-t_1} \lg^5 n)$ time.*

To find the most frequent element in the multiset of frequent colors that appear in $P_{x,y}$, we mark $O(n^{1-t_2})$ nodes using a two-level marking scheme (Section 4.1). Section 4.2 then handles the case in which the endpoints of a query path containing the root are both marked. Finally, We assemble all components in Section 4.3 and solve batched range mode.

4.1 Marking $O(n^{1-t_2})$ Nodes

Let X be an integer parameter in $[n^{t_2}, n]$ to be determined later. We choose at most n/X nodes of T using Lemma 2, mark these nodes and the root, and call them *tier-1 marked nodes*. Since the tier-1 marked nodes form a subset of levels of T, removing them splits T

into a forest of subtrees. Then we use the same lemma to mark nodes at every $\lceil n^{t_2} \rceil$ levels of each subtree starting from some level of the subtree, and these nodes are called *tier-2 marked nodes*. Since the total number of nodes over all subtrees is less than n, there are $O(n^{1-t_2})$ tier-2 marked nodes. We regard both tier-1 and tier-2 marked nodes as *marked nodes*, and there are $O(n/X + n^{1-t_2}) = O(n^{1-t_2})$ marked nodes in total. It follows that a path that connects any non-root node to its lowest marked proper ancestor contains no more than $\lceil n^{t_2} \rceil + 1$ nodes. As before, we refer to the i-th marked node, x_i, visited in a preorder traversal as the *i-th marked node* for short, starting from 0, and $r(x')$ is the rank of the marked node x'.

4.2 Paths with Marked Endpoints Containing the Root: Frequent Colors

Now, for a path $P_{x',y'}$ such that x' and y' are both marked nodes and $\perp \in P_{x',y'}$, we show how to find the most frequent element and its frequency in the multiset of frequent colors assigned to nodes in $P_{x',y'} \backslash \{\perp\} = P'_{x',\perp} \cup P'_{y',\perp}$. Note that for each color other than $c(\perp)$, its frequencies in $P'_{x',\perp} \cup P'_{y',\perp}$ and in $P_{x',y'}$ are exactly the same, while later we consider $c(\perp)$ separately. A frequent color appears more than n^{1-t_1} times in T, so there are only $O(n^{t_1})$ distinct frequent colors. We number the frequent colors incrementally starting from 0 in an arbitrary order, and we refer to the frequent color numbered by k as color f_k. Let μ denote the total number of marked nodes, and let κ denote the total number of distinct frequent colors. Then $\mu = O(n^{1-t_2})$ and $\kappa = O(n^{t_1})$. Next, we construct a $\mu \times \kappa$ matrix M. Corresponding to marked node x_i and frequent color f_k, $M_{i,k}$ stores the negation of the frequency of f_k in path $P'_{x_i,\perp}$. It follows that the min-plus product[4] of M and its transpose, denoted by $M \star M^T$, is a $\mu \times \mu$ matrix, in which entry $(M \star M^T)_{i,j}$ stores the negation of the maximum frequency of a frequent color in $P'_{x_i,\perp} \cup P'_{x_j,\perp}$, provided $\perp \in P_{x_i,x_j}$. As before, some entries of this matrix correspond to a pair of nodes whose lowest common ancestor is not the root and are thus never used, but we compute these entries regardless. To compute $M \star M^T$ efficiently, it suffices to solve the following problem using M and M^T as input matrices.

▶ **Problem 1** (Min-Plus-Query-Witness problem [36]). *Build a data structure upon a pair of input matrices A and B, such that, given two integers i and j and a set S of integers in a query, the index k^* with $A_{i,k^*} + B_{k^*,j} = \min_{k \notin S}\{A_{i,k} + B_{k,j}\}$ can be found efficiently.*

Gu et al. [17] solved this problem with three preprocessing steps, which we generalize for our solution. Henceforth, we use k^* to denote the index such that $M_{i,k^*} + M^T_{k^*,j} = \min_{k \notin S}\{M_{i,k} + M^T_{k,j}\}$. All the proofs omitted from this section will be made available in the full version of this paper.

Preprocessing Step 1. Gu et al. provided an efficient solution to Problem 1 when the second input matrix is monotone, i.e., entries in the same row are non-decreasing. In our case, the second matrix, M^T, does not have such a property due to the tree topology. Therefore, our first step is to generalize their definition of *total range* over a monotone matrix to our notion of *total difference*, defined over an arbitrary $a \times b$ matrix A as $\sum_{j=1}^{b-1}(\sum_{k=0}^{a-1}[\![A_{k,j} \neq A_{k,j-1}]\!])$, in which the Iverson bracket $[\![A_{k,j} \neq A_{k,j-1}]\!]$ evaluates to 1 if $A_{k,j} \neq A_{k,j-1}$ is true and 0 otherwise. We then bound the total difference of M^T:

4 Given an $m \times n$ matrix A and an $n \times p$ matrix B, the min-plus product of A and B, denoted by $A \star B$, is the $m \times p$ matrix in which entry $(A \star B)_{i,j} = \min_k\{A_{i,k} + B_{k,j}\}$.

▶ **Lemma 7.** *The total difference of the matrix M^T is $O(n)$.*

The total difference of M^T is however not small enough to lead to an efficient solution to Problem 1 directly. The strategy of Gu et al. is to divide each entry of the second input matrix by a carefully chosen integer and round down the result, to decrease the total difference of the matrix. However, the same method will fail to decrease the total difference of M^T, as M^T is not monotone, so we design a new approach to construct a matrix $\tilde{B}$ from M^T that has a smaller total difference. To introduce this approach, we define $\hat{\texttt{parent}}_1(v)$ to be node v's lowest proper ancestor that is tier-1 marked and $\hat{\texttt{parent}}(v)$ to be v's lowest proper ancestor that is either tier-1 or tier-2 marked. For a tier-1 marked node $\hat{v}_1$, let $R_k(\hat{v}_1)$ denote the frequency of the frequent color f_k in $P'_{\hat{v}_1,\perp}$, and for a tier-2 marked node $\hat{v}_2$, let $R'_k(\hat{v}_2)$ denote the frequency of f_k in $P'_{\hat{v}_2,\hat{\texttt{parent}}(\hat{v}_2)}$ and $\Psi(\hat{v}_2)$ denote the set of tier-2 marked nodes in $P'_{\hat{v}_2,\hat{\texttt{parent}}_1(\hat{v}_2)}$. We define parameter W to be $\lfloor n^\theta \rfloor$, where θ is a constant with $0 \le \theta \le 1$. With the notations above, we introduce matrix $\tilde{B}$ which has the same size as M^T. For each pair (k, j), if the j-th column of M^T corresponds to a tier-1 marked node $\hat{v}_1$, then $\tilde{B}_{k,j}$ stores $-\lfloor \frac{R_k(\hat{v}_1)}{W} \rfloor$. Otherwise, this column must correspond to a tier-2 marked node $\hat{v}_2$; let $\hat{v}_1$ denote $\hat{\texttt{parent}}_1(\hat{v}_2)$ and we set $\tilde{B}_{k,j} = -\lfloor \frac{R_k(\hat{v}_1)}{W} \rfloor - \sum_{u \in \psi(\hat{v}_2)} \lfloor \frac{R'_k(u)}{W} \rfloor$. Then we can bound the total difference of $\tilde{B}$:

▶ **Lemma 8.** *The total difference of $\tilde{B}$ is $O(\frac{n}{X} \cdot n^{t_1} + \frac{n}{W})$. Setting X to be $\lfloor W n^{t_1} \rfloor$, the total difference of $\tilde{B}$ is $O(n/W)$.*

We define a matrix $\tilde{A}$, in which entry $\tilde{A}_{i,k} = \lfloor \frac{M_{i,k}}{W} \rfloor$, and a matrix $\tilde{C}'$ of the same size as matrix $M \star M^T$, in which $\tilde{C}'_{i,j}$ stores the $(|S|+1)$-st smallest element in $\{\tilde{A}_{i,k} + \tilde{B}_{k,j} : k \in [\kappa]\}$. Let $[n]$ denote $\{0, 1, \cdots, n-1\}$. For each $i \in [\mu]$ and each $j \in [\mu]$, we define set $L_{i,j}$ to be $\{(\tilde{A}_{i,k} + \tilde{B}_{k,j}, k) : k \in [\kappa]\}$. The small total difference of $\tilde{B}$ allows us to borrow ideas from [17] to design a fast algorithm to compute $\tilde{C}'$ and to build a structure that maintains $L_{i,j}$:

▶ **Lemma 9.** *Matrix $\tilde{C}'$ can be computed in $\tilde{O}(n^{2-2t_2} + n^{(2-t_2)}/W)$ time using matrices $\tilde{A}$ and $\tilde{B}$. Furthermore, in the same amount of time, a data structure of $\tilde{O}(n^{2-2t_2} + n^{(2-t_2)}/W)$ words can be constructed upon $\tilde{A}$ and $\tilde{B}$, such that, given a pair (i, j) and an integer $t \ge 1$, the t smallest pairs in $L_{i,j}$, keyed by the first item of each pair, can be listed in $\tilde{O}(t)$ time.*

Preprocessing Step 2. In this step, we take matrices M, M^T and $\tilde{C}'$ as input and partially solve Problem 1 under certain conditions. Let $\rho \ge 0$ be a parameter to be chosen later; let c be any constant that is no less than 1. For each $r \in [(c+2)\mu^\rho \ln n]$, we construct matrices A^r and B^r as follows: Sample j^r uniformly at random from $[\mu]$. Set $A^r_{i,k}$ to be $M_{i,k} + M^T_{k,j^r} - W \cdot \tilde{C}'_{i,j^r}$ if $|M_{i,k} + M^T_{k,j^r} - W \cdot \tilde{C}'_{i,j^r}| \le 2(W-1) \cdot (3 + W \cdot n^{(t_1-t_2)})$ and $A^{r'}_{i,k} = \infty$ for all $r' < r$; otherwise, set $A^r_{i,k} = \infty$. In addition, set entry $B^r_{k,j}$ to be $M^T_{k,j} - M^T_{k,j^r}$ if $M^T_{k,j^r} \ne \infty$; otherwise, set $B^r_{k,j} = 0$. Following [17], for a pair (i,k), if $A^r_{i,k} \ne \infty$ for some r, we call (i,k) *covered*; otherwise it is *uncovered*. We call a triple (i,k,j) *weakly relevant* if $|M_{i,k} + M^T_{k,j} - W \cdot \tilde{C}'_{i,j}| \le 2(W-1) \cdot (3 + W \cdot n^{(t_1-t_2)})$, and we call a triple (i,k,j) *almost relevant* if $0 \le \tilde{A}_{i,k} + \tilde{B}_{k,j} - \tilde{C}'_{i,j} \le \frac{(W-1)\cdot(3+W\cdot n^{(t_1-t_2)})}{W}$. Let $\omega(1, t_1/(1-t_2), 1)$ denote the smallest number such that the product of an $n \times \lceil n^{t_1/(1-t_2)} \rceil$ matrix and an $\lceil n^{t_1/(1-t_2)} \rceil \times n$ matrix can be computed in $O(n^{\omega(1,t_1/(1-t_2),1)+\epsilon})$ time for any constant $\epsilon > 0$. We have:

▶ **Lemma 10.** *Given matrices A^r and B^r, a data structure of $\tilde{O}(2(W-1)(3+Wn^{(t_1-t_2)}) \cdot n^{(1-t_2)\cdot(\rho+2+\frac{t_1}{1-t_2}-\sigma)} + n^{(1-t_2)\cdot(\rho+1+\frac{2t_1}{1-t_2}-\sigma)})$ words can be built in $\tilde{O}(2(W-1)(3+Wn^{(t_1-t_2)}) \cdot n^{(1-t_2)\cdot(\rho+\omega(1,t_1/(1-t_2),1)+\frac{t_1}{1-t_2}-\sigma)})$ time to partially solve the min-plus-query-witness query* *problem upon matrices M and M^T. More precisely, given a query (S, i, j), if (i, k^*) has been covered, then k^* can be found in $\tilde{O}(|S| + n^{(1-t_2)\cdot(\rho+\sigma)})$ time, where ρ and σ are two constant parameters with $\rho \geq 0$, $0 \leq \sigma \leq \frac{t_1}{1-t_2}$. The randomized part in this preprocessing step can be derandomized. Furthermore, after preprocessing, the number of triples that are almost relevant and uncovered is $O(n^{(1-t_2)\cdot(2+\frac{t_1}{1-t_2}-\rho)})$.*

Preprocessing Step 3. Finally, for each pair (i, j), we define $V_{i,j} = \{(M_{i,k} + M^T_{k,j}, k) : (i, k, j)$ is an uncovered and almost relevant triple$\}$ and apply the same method in [17]:

▶ **Lemma 11.** *In $\tilde{O}(n^{2-2t_2} + n^{(2-t_2)}/W + n^{(1-t_2)\cdot(2+\frac{t_1}{1-t_2}-\rho)})$ time, one can find all almost relevant and uncovered triples and build a data structure of $\tilde{O}(n^{2-2t_2} + n^{(1-t_2)\cdot(2+\frac{t_1}{1-t_2}-\rho)})$ words upon them, such that, given a pair (i, j) and an integer $t \geq 1$, the t smallest elements in $V_{i,j}$, keyed by the first item in each pair, can be listed in $\tilde{O}(t)$ time.*

The Querying Procedure. The query algorithm is similar to the one shown in [17]. Let (S, i, j) be the query parameters. If (i, k^*) is covered, then we use Lemma 10 to find k^* in $\tilde{O}(|S|+n^{(1-t_2)(\rho+\sigma)})$ time. Otherwise, we claim that $\tilde{A}_{i,k^*}+\tilde{B}_{k^*,j}-\tilde{C'}_{i,j} \leq \frac{(W-1)(Wn^{t_1-t_2}+3)}{W}$ (the proof will be made available in the full version of this paper); therefore, either (i, k^*, j) is almost relevant, or $\tilde{A}_{i,k^*} + \tilde{B}_{k^*,j} - \tilde{C'}_{i,j} < 0$. If (i, k^*, j) is almost relevant, then $(M_{i,k^*} + M^T_{k^*,j}, k^*)$ is among the $(|S| + 1)$ smallest elements in $V_{i,j}$; thereby, we can find k^* in $\tilde{O}(|S|)$ time using Lemma 11. If $\tilde{A}_{i,k^*} + \tilde{B}_{k^*,j} - \tilde{C'}_{i,j} < 0$, then $(\tilde{A}_{i,k^*} + \tilde{B}_{k^*,j}, k^*)$ is among the $(|S| + 1)$ smallest elements in $L_{i,j}$; we can find k^* for this case in $\tilde{O}(|S|)$ time using Lemma 9. As a result, we have solved the min-plus-query-witness problem over M and M^T and achieved Lemma 12. Recall that parameter W was set to be $\lfloor n^\theta \rfloor$.

▶ **Lemma 12.** *A data structure of $\tilde{O}(n^{2-2t_2} + n^{(1-t_2)(1-\theta)+1} + n^{(1-t_2)(2+\frac{t_1}{1-t_2}-\rho)} + n^{(1-t_2)(\rho+1+\frac{2t_1}{1-t_2}-\sigma)} + n^{2\theta(1-t_2)+(t_1-t_2)+(1-t_2)(\rho+2+\frac{t_1}{1-t_2}-\sigma)})$ words can be built upon M and M^T in $\tilde{O}(n^{1+(1-t_2)(1-\theta)} + n^{2\theta(1-t_2)+(t_1-t_2)+(1-t_2)(\rho+\omega(1,\frac{t_1}{1-t_2},1)+\frac{t_1}{1-t_2}-\sigma)} + n^{(1-t_2)(2+\frac{t_1}{1-t_2}-\rho)})$ time, such that a query defined in Problem 1 can be answered in $\tilde{O}(|S| + n^{(1-t_2)(\sigma+\rho)})$ time, where $\rho \geq 0$, $0 \leq \sigma \leq \frac{t_1}{1-t_2}$, and $0 \leq \theta \leq 1$.*

As a result, we can apply Lemma 12 to find the most frequent element and its frequency in the multiset of frequent colors assigned to nodes in $P'_{x',\perp} \cup P'_{y',\perp}$ provided that x' and y' are marked and $\perp \in P_{x',y'}$. In the static case, the excluded set S is always set to be empty.

4.3 Mode Queries on an Arbitrary Path

We first represent tree T using Lemma 1. As discussed in Section 2, this structure supports finding the number of appearances of a color on a path in $O(\lg\lg C)$ time. Then we mark $O(n^{1-t_2})$ nodes of T as discussed in Section 4.1. We compute the number of appearances of each color on T to determine whether it is frequent or infrequent. Then we construct the data structures of Lemmas 6 and 12 for queries over infrequent and frequent colors, respectively.

Let $P_{x,y}$ be a query path containing the root. W.l.o.g., we assume that neither x nor y is the root node. If $P_{x,y}$ contains less than two marked nodes, then, by our node-marking strategy, $|P_{x,y}| = O(n^{t_2})$. In this case, a mode on $P_{x,y}$ can be found in $O(n^{t_2})$ time. Otherwise, let x' and y' denote the lowest marked ancestors of x and y, respectively, and we divide $P_{x,y}$ into four disjoint parts: $P'_{x,x'}$, $P'_{x',\perp} \cup P'_{y',\perp}$, $\perp$ and $P'_{y,y'}$. Since there are $O(n^{t_2})$ nodes in $P'_{x,x'} \cup P'_{y,y'} \cup \perp$, it requires $O(n^{t_2} \lg\lg C)$ time to scan the nodes in $P'_{x,x'} \cup P'_{y,y'} \cup \perp$, and for each color encountered, count its occurrences in $P_{x,y}$. Let c_1 be the color with maximum number of occurrences found this way. Then we apply Lemma 6 to find the infrequent color, c_2, with maximum frequency in $P_{x',y'}$ in $\tilde{O}(n^{1-t_1})$ time, and we query over the data structure of Lemma 12, setting the excluded set $S = \emptyset$, to find the frequent color, c_3, with maximum frequency in $P'_{x',\perp} \cup P'_{y',\perp}$ in $\tilde{O}(n^{(1-t_2)(\sigma+\rho)})$ time. We also obtain the frequency of c_2 in $P_{x',y'}$ and the frequency of c_3 in $P'_{x',\perp} \cup P'_{y',\perp}$ when finding c_2 and c_3. Note that, if the mode is not c_1, then the mode does not appear in $P'_{x,x'} \cup P'_{y,y'} \cup \perp$, so it must be either c_2 or c_3. Hence it suffices to compare the frequency of c_1 in $P_{x,y}$, the frequency of c_2 in $P_{x',y'}$, and the frequency of c_3 in $P'_{x',\perp} \cup P'_{y',\perp}$ to find the answer to the query.

Finally, we apply the technique in Section 3.2 to compute the mode in an arbitrary path:

▶ **Lemma 13.** *Let T be a colored ordinal tree on n nodes with each node assigned a color from $\{0, 1, \ldots, C-1\}$, where $C \leq n$. A data structure of $\tilde{O}(n^{2-t_1} + n^{2-2t_2} + n^{(1-t_2)(1-\theta)+1} + n^{(1-t_2)(\rho+1+\frac{2t_1}{1-t_2}-\sigma)} + n^{2\theta(1-t_2)+(t_1-t_2)+(1-t_2)(\rho+2+\frac{t_1}{1-t_2}-\sigma)} + n^{(1-t_2)(2+\frac{t_1}{1-t_2}-\rho)})$ words can be constructed in $\tilde{O}(n^{2-t_1} + n^{2\theta(1-t_2)+(t_1-t_2)+(1-t_2)(\rho+\omega(1,\frac{t_1}{1-t_2},1)+\frac{t_1}{1-t_2}-\sigma)} + n^{(1-t_2)(2+\frac{t_1}{1-t_2}-\rho)} + n^{1+(1-t_2)(1-\theta)})$ time, such that a path mode query can be answered in $\tilde{O}(n^{t_2} + n^{1-t_1} + n^{(1-t_2)(\sigma+\rho)})$ time, where $\rho \geq 0$, $0 \leq \sigma \leq \frac{t_1}{1-t_2}$, and $0 \leq \theta \leq 1$.*

Applying Lemma 13 to batched path mode and setting $t_1 = \frac{10}{17+\omega(1,1,1)}$, where $\omega(1,1,1) \in [2, 2.37286)$, $t_2 = 1 - t_1$, $\theta = \frac{2t_1-1}{t_1}$, $\rho = \frac{4t_1-2}{t_1}$, and $\sigma = \frac{3-5t_1}{t_1}$ yields:

▶ **Theorem 14.** *A batch of n path mode queries over a colored tree T on n nodes can be answered in $\tilde{O}(n^{\frac{24+2\omega}{17+\omega}}) = O(n^{1.483814})$ time.*

References

1 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

2 Stephen Alstrup and Jacob Holm. Improved algorithms for finding level ancestors in dynamic trees. In *International Colloquium on Automata, Languages, and Programming*, pages 73–84. Springer, 2000.

3 Djamal Belazzougui, Fabio Cunial, Juha Kärkkäinen, and Veli Mäkinen. Linear-time string indexing and analysis in small space. *ACM Transactions on Algorithms (TALG)*, 16(2):1–54, 2020.

4 Djamal Belazzougui and Gonzalo Navarro. Optimal lower and upper bounds for representing sequences. *ACM Transactions on Algorithms (TALG)*, 11(4):1–21, 2015.

5 Gerth Stølting Brodal, Pooya Davoodi, and S Srinivasa Rao. Path minima queries in dynamic weighted trees. In *Workshop on Algorithms and Data Structures*, pages 290–301. Springer, 2011.

6 Timothy M Chan, Stephane Durocher, Kasper Green Larsen, Jason Morrison, and Bryan T Wilkinson. Linear-space data structures for range mode query in arrays. *Theory of Computing Systems*, 55(4):719–741, 2014.

7 Timothy M Chan, Stephane Durocher, Matthew Skala, and Bryan T Wilkinson. Linear-space data structures for range minority query in arrays. *Algorithmica*, 4(72):901–913, 2015.

8 Timothy M Chan, Meng He, J Ian Munro, and Gelin Zhou. Succinct indices for path minimum, with applications. *Algorithmica*, 78(2):453–491, 2017.

9 Bernard Chazelle. Computing on a free tree via complexity-preserving mappings. *Algorithmica*, 2(1):337–361, 1987.

10 Erik D Demaine, Gad M Landau, and Oren Weimann. On cartesian trees and range minimum queries. In *International Colloquium on Automata, Languages, and Programming*, pages 341–353. Springer, 2009.

11 Stephane Durocher, Rahul Shah, Matthew Skala, and Sharma V Thankachan. Linear-space data structures for range frequency queries on arrays and trees. *Algorithmica*, 74(1):344–366, 2016.

12 Hicham El-Zein, Meng He, J Ian Munro, and Bryce Sandlund. Improved time and space bounds for dynamic range mode. In *26th Annual European Symposium on Algorithms (ESA 2018)*. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2018.

13 Travis Gagie, Meng He, Gonzalo Navarro, and Carlos Ochoa. Tree path majority data structures. *Theoretical Computer Science*, 833:107–119, 2020.

14 Travis Gagie, Juha Kärkkäinen, Gonzalo Navarro, and Simon J Puglisi. Colored range queries and document retrieval. *Theoretical Computer Science*, 483:36–50, 2013.

15 Younan Gao and Meng He. Space efficient two-dimensional orthogonal colored range counting. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 46:1–46:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. Pdoi:10.4230/LIPIcs.ESA.2021.46.

16 Roberto Grossi and Søren Vind. Colored range searching in linear space. In *Scandinavian Workshop on Algorithm Theory*, pages 229–240. Springer, 2014.

17 Yuzhou Gu, Adam Polak, Virginia Vassilevska Williams, and Yinzhan Xu. Faster monotone min-plus product, range mode, and single source replacement paths. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 75:1–75:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. Pdoi:10.4230/LIPIcs.ICALP.2021.75.

18 Prosenjit Gupta, Ravi Janardan, and Michiel Smid. Further results on generalized intersection searching problems: counting, reporting, and dynamization. *Journal of Algorithms*, 19(2):282–317, 1995.

19 Prosenjit Gupta, Ravi Janardan, and Michiel Smid. Algorithms for generalized halfspace range searching and other intersection searching problems. *Computational Geometry*, 6(1):1–19, 1996.

20 Yijie Han. Deterministic sorting in o (n log log n) time and linear space. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 602–608, 2002.

21 Meng He and Serikzhan Kazi. Data structures for categorical path counting queries. In *32nd Annual Symposium on Combinatorial Pattern Matching (CPM 2021)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

22 Meng He, J Ian Munro, and S Srinivasa Rao. Succinct ordinal trees based on tree covering. In *International Colloquium on Automata, Languages, and Programming*, pages 509–520. Springer, 2007.

23 Meng He, J Ian Munro, and Gelin Zhou. A framework for succinct labeled ordinal trees over large alphabets. *Algorithmica*, 70(4):696–717, 2014.

24 Meng He, J Ian Munro, and Gelin Zhou. Data structures for path queries. *ACM Transactions on Algorithms (TALG)*, 12(4):1–32, 2016.

25 Meng He, J Ian Munro, and Gelin Zhou. Dynamic path queries in linear space. *Algorithmica*, 80(12):3728–3765, 2018.

26 David Hutchinson, Anil Maheshwari, and Norbert Zeh. An external memory data structure for shortest path queries. *Discrete Applied Mathematics*, 126(1):55–82, 2003.

27 Ce Jin and Yinzhan Xu. Tight dynamic problem lower bounds from generalized bmm and omv. *arXiv preprint arXiv:2202.11250*, 2022.

28 Camille Jordan. Sur les assemblages de lignes. *Journal für die reine und angewandte Mathematik*, 70:185–190, 1869. URL: `http://eudml.org/doc/148084`.

29 Haim Kaplan, Natan Rubin, Micha Sharir, and Elad Verbin. Efficient colored orthogonal range counting. *SIAM Journal on Computing*, 38(3):982–1011, 2008.

30 Haim Kaplan and Nira Shafrir. Path minima in incremental unrooted trees. In *European Symposium on Algorithms*, pages 565–576. Springer, 2008.

31 Haim Kaplan, Micha Sharir, and Elad Verbin. Colored intersection searching via sparse rectangular matrix multiplication. In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 52–60, 2006.

32 Danny Krizanc, Pat Morin, and Michiel Smid. Range mode and range median queries on lists and trees. *Nordic Journal of Computing*, 12(1):1–17, 2005.

33 J Ian Munro, Yakov Nekrich, and Sharma V Thankachan. Range counting with distinct constraints. In *CCCG*, 2015.

34 Yakov Nekrich. Efficient range searching for categorical and plain data. *ACM Transactions on Database Systems (TODS)*, 39(1):1–21, 2014.

35 Manish Patil, Rahul Shah, and Sharma V Thankachan. Succinct representations of weighted trees supporting path queries. *Journal of Discrete Algorithms*, 17:103–108, 2012.

36 Bryce Sandlund and Yinzhan Xu. Faster dynamic range mode. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

37 Virginia Vassilevska Williams and Yinzhan Xu. Truly subcubic min-plus product for less structured matrices, with applications. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 12–29. SIAM, 2020.

38 Huacheng Yu. An improved combinatorial algorithm for boolean matrix multiplication. *Information and Computation*, 261:240–247, 2018.