Top-k Color Queries On Tree Paths[⋆]

Stephane Durocher[1], Rahul Shah[2],
Matthew Skala[1], and Sharma V. Thankachan[2]

[1] University of Manitoba, Winnipeg, Canada, {durocher,mskala}@cs.umanitoba.ca
[2] Louisiana State University, Baton Rouge, USA, {rahul,thanks}@csc.lsu.edu

Abstract. We present a data structure for the following problem: Given a tree $\mathcal{T}$, with each of its nodes assigned a color in a totally ordered set, preprocess $\mathcal{T}$ to efficiently answer queries for the top k distinct colors on the path between two nodes, reporting the colors sorted in descending order. Our data structure requires linear space of $O(n)$ words and answers queries in $O(k)$ time.

1 Introduction and Related Work

Given an array $A[1..n]$ of color values in $\{1, 2, \ldots, \sigma\}$ and a function $p(c)$ that defines priorities for the colors, the *array range top-k color query problem* is to report, given indices a and b and a count k, the k distinct color values of highest priority to occur in the array range $A[a..b]$, in descending order of priority. A data structure for this problem must preprocess the array to answer queries efficiently. In this work we generalize the array range top-k color query problem to paths in trees.

Karpinski and Nekrich [15] give a data structure for the array problem with $O(k)$ query time using $O(n \log \sigma)$ bits, which is asymptotically optimal. Other related problems include reporting all distinct colors, counting the number of distinct colors in the query range, finding the most frequent, least frequent, majority or minority color in the query range, and so on. Efficient data structures offering different space-time trade-offs for these kinds of problems are known [1–6, 8–11, 16]. Such problems arise frequently in information retrieval and computational geometry.

Any two nodes in a tree define a unique path between them, just as two elements of an array define a unique range. When the entire tree is a path, tree paths reduce to array ranges. Most array range query problems can thus be generalized to a tree setting, and some of these problems are well studied. Krizanc et al. [16] discuss finding the median and mode on tree paths. He et al. [12] give efficient linear-space solutions for path selection (including median), counting, and reporting, on individual node values (weights) as opposed to classes of distinct node values (colors) [12]. Recent results include succinct data structures for these weighted path query problems [13, 19].

[⋆] Work supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and National Science Foundation (NSF) Grants CCF–1017623 (R. Shah and J. S. Vitter) and CCF–1218904 (R. Shah).

Suppose each node in a tree $\mathcal{T}$ of n nodes is assigned a color $c \in \{1, \ldots, \sigma\}$ and each color has a priority $p(c)$. Let $\mathcal{T}[a..b]$ represent the unique path connecting the nodes with preorder indices a and b. Then the *tree path top-k color query problem* is to report, given indices a and b and a count k, the k distinct color values of highest priority to occur in $\mathcal{T}[a..b]$, in descending order according to priority. We will prove the following result in Word-RAM model.

Theorem 1. *There exists a linear-space data structure of $O(n)$ words for answering tree path top-k color queries in $O(k)$ time.*

2 Our solution

We begin by defining some useful concepts. Let $\mathcal{T}$ be a tree with n nodes. We define the *size* of an internal node v to be the number of leaves in the subtree rooted at v. Then the *heavy path* of the tree $\mathcal{T}$ is the path starting from the root, which each node v on the path is the largest-size child of its parent. Let the root of a heavy path be its highest node, that is, closest to the root of $\mathcal{T}$. The *heavy path decomposition* of the tree $\mathcal{T}$ is the operation where we decompose each off-path subtree of the heavy path recursively; therefore, edges in $\mathcal{T}$ will be partitioned into disjoint heavy paths. Each leaf node belongs to a unique path in the heavy decomposition, and each heavy path contains exactly one leaf. Therefore the number of heavy paths is exactly equal to the number of leaves. This decomposition will allow us to break query paths into a small number of pieces by the following lemma.

Lemma 1 (Sleator and Tarjan [20]). *Any path from the root to a leaf in $\mathcal{T}$ intersects at most $\log n$ paths of the heavy path decomposition.*

We will also use a data structure of Navarro and Nekrich for three-sided two-dimensional top-k queries [18]. Given a set of n points on an $n \times n$ grid, each having a weight, this data structure can report the k points of greatest weight in a query region of the form $[a, b] \times [0, h]$, in $O(h + k)$ time. Furthermore, it reports the points in order of decreasing weight, and it reports them online, that is, in constant time per point after the $O(h)$ time to start the query; k need not be specified in advance.

Any tree path $\mathcal{T}[a..b]$ can be divided into two overlapping paths $\mathcal{T}[a..z]$ and $\mathcal{T}[z..b]$, where z is the lowest common ancestor (LCA) of a and b. If we can answer tree path top-k color queries on these two paths in $O(k)$ time, then we can merge the answers to answer such queries on arbitrary paths in the tree in the same time. Therefore, it suffices to solve the following problem.

Problem 1. Preprocess $\mathcal{T}$ to efficiently answer tree path top-k color queries on paths of the form $\mathcal{T}[a..z]$ where z is an ancestor of a.

A tree path top-k query involves three constraints. Each element returned must be (1) on the query path; (2) among the top k; and (3) of a distinct color. In other words, each color must be reported, and counted towards the top k, only once, even if many elements of that color appear on the query path. We eliminate the duplicates using an adaptation of Muthukrishnan's chaining approach to reporting distinct colors in array ranges [17].

Let $depth(i)$ be the number of nodes on the path from the root to a node i. Let $chain(i)$ be the depth of the lowest ancestor of i that has the same color as i, with $chain(i) = 0$ if there is no such ancestor. If there exists at least one node with color c in $\mathcal{T}[a..z]$, z being an ancestor of a, then there exists a unique node i in $\mathcal{T}[a..z]$ with color c and $chain(i) < depth(z)$. Therefore, Problem 1 can be rephrased as follows:

Problem 2. Preprocess $\mathcal{T}$ to efficiently find, given a node a, one of its ancestors z, and a count k, the top k colors in decreasing order of priority among the nodes in the set $\{i \in \mathcal{T}[a..z] | chain(i) < depth(z)\}$.

For any node given by its preorder rank i, let $\phi(i)$ be the root of the path containing i in the heavy path decomposition of $\mathcal{T}$. Let ℓ_j be the preorder rank of the jth leftmost leaf in $\mathcal{T}$. We can transform $\mathcal{T}$ to another tree $\mathcal{T}'$, which is actually a path, by concatenating the paths $\mathcal{T}[\ell_i, \phi(\ell_i)]$ for each i up to the number of leaves. Then we can define an array $A[1..n]$ containing the priorities of the colors of nodes in $\mathcal{T}'$, in order along the path starting from the root. property is ensured. Any subpath of a path in the heavy path decomposition must correspond to a contiguous range of A.

We build the optimal array range top-k color query data structure of Karpinski and Nekrich on A, using $O(n \log \sigma)$ bits [15]. From each node i in $\mathcal{T}$, we store the index of the corresponding entry in A. The total space consumption is $O(n)$ words assuming $\sigma \leq n$. Because heavy paths are contiguous in A, the special case where both a and z are on the same heavy path can be handled optimally by first finding the corresponding range in A, and then performing a top-k color query on the array range top-k color query data structure.

For the case of a and z not on the same path of the decomposition, we map each node i in $\mathcal{T}$ to a weighted two-dimensional point (x_i, y_i) with weight w_i, letting w_i be the priority of the color of node i, x_i be the index in A corresponding to that node, and y_i be the number of paths of the heavy path decomposition intersected by the path from the root to $chain(i)$. We build the data structure [3] of Navarro and Nekrich for three-sided two-dimensional top-k queries on these weighted points [18, Theorem 2.1]. Because $y_i \leq \log n$, the query time to return k points from this data structure is $O(\log n + k)$.

To answer a general tree path top-k color query, we first find the lowest common ancestors of its endpoints and split the path into two queries of the form $\mathcal{T}[a..z]$, with z an ancestor of a. Then for each of the two, we partition the query path $\mathcal{T}[a..z]$ into at most $\log n$ disjoint subpaths $\mathcal{T}[a_1..\phi(a_1)]$, $\mathcal{T}[a_2..\phi(a_2)]$, ..., $\mathcal{T}[a_r..z]$, where $a_1 = a$, a_i is the parent of $\phi(a_{i-1})$ for $i = 2, 3, \ldots, r$, and $r \leq \log n$

[3] Note that an alternate approach is to build the data structure described in [15] over the list of colors corresponding to each heavy path, and later perform top-k queries on all heavy paths intersect with $\mathcal{T}[a..z]$. However, the drawback of this approach is that the same color may be reported from several heavy paths.

is such that z is on the subpath $\mathcal{T}[a_r..\phi(a_r)]$. This step takes only $O(r) = O(\log n)$ time, by consulting a stored copy of the heavy path decomposition.

The query $\mathcal{T}[a_r..z]$ corresponds to a contiguous range in A, so we can find its top k colors in $O(k)$ time and merge them in $O(k)$ time with the top k colors for $\mathcal{T}[a_1..\phi(a_{r-1})]$. It only remains to query $\mathcal{T}[a_1..\phi(a_{r-1})]$ efficiently.

From the definition in Problem 2, we have that a node i can only be part of the result for $\mathcal{T}[a_1..\phi(a_{r-1})]$ if $\mathit{chain}(i) < \mathit{depth}(\phi(a_{r-1}))$. In fact, it suffices to check that the number of centroidal paths intersected by the path from the root of $\mathcal{T}$ to $\mathit{chain}(i)$ is less than the number intersected from the root to $\phi(a_{r-1})$. Since $\phi(a_{r-1})$ and its parent cannot be on the same path of the decomposition, any ancestor of it must be in a path nearer the root.

Let h be the number of paths of the heavy path decomposition that are intersected by the path from $\phi(a_{r-1})$ to the root, and let $A[s_i..e_i]$ be the range of A associated with $\mathcal{T}[a_i..\phi(a_i)]$. Then for each $i \in \{1, 2, \ldots, r-1\}$, the weighted points $(x_j, y_j) \in [s_i, e_i] \times [0, h]$ correspond to nodes with distinct colors on the path $\mathcal{T}[a_1..\phi(a_{r-1})]$. The union of those lists would include the top k colors in $\mathcal{T}[a_1..\phi(a_{r-1})]$, but we still must merge the lists.

We issue $r-1$ simultaneous queries to the top-k geometric data structure, corresponding to the query regions $[s_i, e_i] \times [0, h]$ for $i = 1, 2, \ldots, r-1$. The answers can be merged using a max-heap H with its size limited to at most $r - 1 = O(\log n)$ points. We insert the first point returned from each R_i into H. Then we repeat the following steps until we have reported k colors:

1. Extract the highest weighted point in H and report it.
2. If the reported point was from the query box R_i, then fetch the next highest weighted point from R_i and insert it into H.

Since the size of H is always $\log^{O(1)} n$, we can use an atomic heap, which can perform all heap operations in constant time in the Word RAM model [7]. Therefore, the number of heap operations, and the required time, can be bounded by $O(k + \log n)$. Each three-sided two-dimensional top-k query takes $O(\log n)$ time in addition to the number of points it returns. Therefore the total time for query is $O(k + \log^2 n)$.

Lemma 2. *There exists a linear-space data structure for answering tree path top-k color queries in $O(k + \log^2 n)$ time.*

For $k = \Omega(\log^2 n)$, that time is optimal. To handle the case of small k, we use other techniques. First, we will choose a subset of the nodes in $\mathcal{T}$, called the *marked* nodes, as follows. Let g be an integer to be chosen later called the *grouping factor*, and mark every node i in $\mathcal{T}$ such that $i \equiv 0 \pmod{g}$. Also mark the lowest common ancestor of any pair of marked nodes. This is a simplified version of the scheme introduced by Hon et al. [14] for identifying marked nodes in a suffix tree. It has the properties given in the following lemma (the proof is deferred to the full version).

Lemma 3. *If we mark all nodes i such that $i \equiv 0 \pmod{g}$, and all lowest common ancestors of pairs of such nodes, then:*

1. *the number of marked nodes is $O(n/g)$;*
2. *the lowest marked ancestor of any node is $O(g)$ nodes above it;*
3. *any* unmarked *node has at most one unique highest marked descendant; and*
4. *for any unmarked node i, the subtree of $\mathcal{T}$ rooted at i and excluding the subtree rooted at its highest marked descendant j (if any) contains $O(g)$ nodes.*

We mark nodes in $\mathcal{T}$ using $g = \log^3 n$. For every marked node i and every j such that j is an ancestor of i and is the root of a path in the heavy path decomposition, we store explicitly a precomputed sorted list of the top $O(\log^2 n)$ colors on the path $\mathcal{T}[i..j]$. The space is bounded by $O((n/\log^3 n)\log^2 n \log n) = O(n)$ words. Using these precomputed lists we can find the top k colors in $\mathcal{T}[a..z]$, where a is a marked node and z is one of its ancestors, by splitting the query, as before, into $\mathcal{T}[a_1..\phi(a_{r-1})]$ and $\mathcal{T}[a_r..z]$. The former is precomputed and the latter corresponds to a contiguous range of A. We can find the top k colors in both of them and merge the lists in $O(k)$ time.

Lemma 4. *There exists a linear-space data structure for answering tree path top-k color queries of the form $\mathcal{T}[a..z]$ in $O(k)$ time, where a is a marked node and z is one of its ancestors.*

Next we must handle the case in which a is not a marked node. For any node i such that i is not marked but its parent is marked, define the *mini-tree* $\mathcal{T}^i$ to be the subtree rooted at i but excluding any descendants of the highest marked descendant of i, if any. By Lemma 3, $\mathcal{T}^i$ is of size $O(g)$. We choose a grouping factor $g' = \log^3 g = \Theta(\log^3 \log n)$ and use it to mark nodes within each mini-tree, and build the data structure of Lemma 4 for each mini-tree. The total space is bounded by $O(n)$ because each node in $\mathcal{T}$ belongs to exactly one mini-tree. By querying from a to the root of its mini-tree, and from the parent of that root to z, and merging the results, we can answer top-k color queries of the form $\mathcal{T}[a..z]$ in $O(k)$ time when a is marked in its mini-tree and z is its ancestor, even if a is not marked in T.

Finally, we generalize the optimal-time solution to arbitrary a. For any i not marked in $\mathcal{T}$ nor in the mini-tree that contains i, let j be the lowest marked node above i in the same mini-tree. The node j is at most $g' = O(\log^3 \log n)$ nodes above i. Therefore the top k colors on the path $\mathcal{T}[i..j]$, for all choices of i, can be stored in $O(n(\log^3 \log n)^2 \log\log\log n)$ bits: there are n choices of i, $O(\log^3 \log n)$ lists, each of length at most $O(\log^3 \log n)$, and only $O(\log\log\log n)$ bits are needed (as indices into the mini-block) to store the entries in the lists. That is a total of $o(n)$ words.

Then to answer an arbitrary query $\mathcal{T}[a..b]$, we first split into two queries $\mathcal{T}[a..z]$ and $\mathcal{T}[b..z]$ with z the lowest common ancestor of a and b. We can find the top k colors from a to its lowest marked ancestor within its mini-tree, and from there to the root of the mini-tree, using the precomputed lists. From the lowest marked ancestor of a in $\mathcal{T}$ to the highest marked ancestor of a below z, we use Lemma 4; and from there to z we do an array range query in A. We do the same with the query $\mathcal{T}[b..z]$. All these queries can be performed, and their results merged, in $O(k)$ time. This completes the proof of Theorem 1.

References

1. D. Belazzougui, T. Gagie, and G. Navarro. Better space bounds for parameterized range majority and minority. In *Proc. WADS*, 2013.
2. D. Belazzougui, G. Navarro, and D. Valenzuela. Improved compressed indexes for full-text document retrieval. *J. Discrete Algorithms*, 18:3–13, 2013.
3. T. M. Chan, S. Durocher, K. G. Larsen, J. Morrison, and B. T. Wilkinson. Linear-space data structures for range mode query in arrays. In *Proc. STACS*, volume 14, pages 291–301, 2012.
4. T. M. Chan, S. Durocher, M. Skala, and B. T. Wilkinson. Linear-space data structures for range minority query in arrays. In *Proc. SWAT*, volume 7357 of *LNCS*, pages 295–306. Springer, 2012.
5. S. Durocher, M. He, J. I. Munro, P. K. Nicholson, and M. Skala. Range majority in constant time and linear space. *Inf. & Comp.*, 222:169–179, 2013.
6. S. Durocher, R. Shah, M. Skala, and S. V. Thankachan. Linear-space data structures for range frequency queries on arrays and trees. In *Proc. MFCS*, 2013.
7. M. L. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. Syst. Sci.*, 48(3):533–551, 1994.
8. T. Gagie, M. He, J. I. Munro, and P. Nicholson. Finding frequent elements in compressed 2D arrays and strings. In *Proc. SPIRE*, volume 7024 of *LNCS*, pages 295–300. Springer, 2011.
9. T. Gagie, J. Kärkkäinen, G. Navarro, and S. J. Puglisi. Colored range queries and document retrieval. *Theor. Comput. Sci.*, 483:36–50, 2013.
10. T. Gagie, S. J. Puglisi, and A. Turpin. Range quantile queries: Another virtue of wavelet trees. In *Proc. SPIRE*, volume 5721 of *LNCS*, pages 1–6. Springer, 2009.
11. B. Gfeller and P. Sanders. Towards optimal range medians. In *Proc. ICALP*, volume 5555 of *LNCS*, pages 475–486. Springer, 2009.
12. M. He, J. I. Munro, and G. Zhou. Path queries in weighted trees. In *Proc. ISAAC*, pages 140–149, 2011.
13. M. He, J. I. Munro, and G. Zhou. Succinct data structures for path queries. In *Proc. ESA*, pages 575–586, 2012.
14. W.-K. Hon, R. Shah, and J. S. Vitter. Space-efficient framework for top-k string retrieval problems. In *Proc. FOCS*, pages 713–722, 2009.
15. M. Karpinski and Y. Nekrich. Top-k color queries for document retrieval. In *Proc. SODA*, pages 401–411, 2011.
16. D. Krizanc, P. Morin, and M. Smid. Range mode and range median queries on lists and trees. In *Proc. ISAAC*, volume 2906 of *LNCS*, pages 517–526. Springer, 2003.
17. S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proc. SODA*, pages 657–666, 2002.
18. G. Navarro and Y. Nekrich. Top-k document retrieval in optimal time and linear space. In *Proc. SODA*, pages 1066–1077, 2012.
19. M. Patil, R. Shah, and S. V. Thankachan. Succinct representations of weighted trees supporting path queries. *J. Discrete Algorithms*, 17:103–108, 2012.
20. D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.