Path Queries in Weighted Trees*

Meng He[1], J. Ian Munro[2], and Gelin Zhou[2]

[1] Faculty of Computer Science, Dalhousie University, Canada.
mhe@cs.dal.ca
[2] David R. Cheriton School of Computer Science, University of Waterloo, Canada.
{imunro, g5zhou}@uwaterloo.ca

Abstract. We consider the problem of supporting several different path queries over a tree on n nodes, each having a weight drawn from a set of σ distinct values, where $\sigma \leq n$. One query we support is the path median query, which asks for the median weight on a path between two given nodes. For this and the more general path selection query, we present a linear space data structure that answers queries in $O(\lg \sigma)$ time under the word RAM model. This greatly improves previous results on the same problem, as previous data structures achieving $O(\lg n)$ query time use $O(n \lg^2 n)$ space, and previous linear space data structures require $O(n^\epsilon)$ time to answer a query for any positive constant ϵ [16]. Our linear space data structure also supports path counting queries in $O(\lg \sigma)$ time. This matches the result of Chazelle [8] when σ is close to n, but has better performance when σ is significantly smaller than n. Finally, the same data structure can also support path reporting queries in $O(\lg \sigma + occ \lg \sigma)$ time, where occ is the size of output. In addition, we present a data structure that answers path reporting queries in $O(\lg \sigma + occ \lg \lg \sigma)$ time, using $O(n \lg \lg \sigma)$ space. These are the first data structures that answer path reporting queries.

1 Introduction

Trees are fundamental structures in computer science, being widely used in modeling and representing different types of data in numerous computer applications. In many cases, properties of objects being modeled are stored as weights or labels on the nodes of trees. Thus researchers have studied the preprocessing of weighted trees in which each node is assigned a weight, in order to support different *path queries*, for which a certain function over the weights of the nodes along a given query path in the tree is computed [1, 8, 13, 16].

In this paper, we design data structures to maintain a weighted tree T on n nodes such that we can support the following path queries:

- Path Median Query: Given two nodes u and v, return the median weight on the path from u to v. If there are m nodes on this path, then the median weight is the $\lceil m/2 \rceil$th smallest one among the weights of these nodes.

* This work was supported by NSERC and the Canada Research Chairs Program.

– Path Selection Query: This kind of query is a natural extension of path median queries. We are given two nodes u, v and a positive integer k, and we need return the kth smallest weight on the path from u to v. We assume that k is not larger than the number of nodes on this path.
– Path Counting Query: Given two nodes u, v and a range $[s, t]$, return the number of nodes on the path from u to v whose weights are in this range.
– Path Reporting Query: Given two nodes u, v and a range $[s, t]$, return the set of nodes on the path from u to v that have a weight in this range.

When the given tree T is a path, the above queries become range median, range selection, two-dimensional range counting and range reporting queries. Thus the path queries we consider generalize these fundamental queries to weighted trees.

1.1 Previous work

The path median problem was first presented by Krizanc et al. [16], who gave two solutions for this problem. The first one supports queries in $O(\lg n)$ time[3], and occupies $O(n \lg^2 n)$ words of space. The second one requires $O(b \lg^3 n / \lg b)$ query time and $O(n \lg_b n)$ space, for any $2 \leq b \leq n$. If we set b to be $n^\epsilon / \lg^2 n$ for some small constant ϵ, then the space cost becomes linear, but the query time is $O(n^\epsilon)$. These are the best known results for the path median problem.

The path counting problem was studied by Chazelle in [8]. In his formulation, weights are assigned to edges instead of nodes, and a query asks for the number of edges on a given path whose weights are in a given range. He designed a linear space data structure to support queries in $O(\lg n)$ time. His technique is based on tree partition, so it requires $O(\lg n)$ time for arbitrary set of weights.

Range median and selection queries. In the range median and selection problems, we are given an unsorted array of n elements, each having a weight. A query asks for the median or the kth smallest weight in a range.

For the static version, to obtain constant query time, the known solutions require near-quadratic space [16, 19, 20]. For linear space data structures, Gfeller and Sanders [11] presented a solution to answer a query in $O(\lg n)$ time. Gagie et al. [9] considered this problem in terms of σ, the number of distinct weights. They designed a data structure based on wavelet trees that supports range selection in $O(\lg \sigma)$ time. The best upper bound was achieved by Brodal et al. [4, 5], which has $O(\lg n / \lg \lg n)$ query time. Later, Jørgensen and Larsen showed that Brodal et al.'s result is optimal by proving a lower bound of $\Omega(\lg n / \lg \lg n)$ on query time, providing that data structures for the static range selection problem use $O(n \lg^{O(1)} n)$ bits of space [15].

Orthogonal range counting and reporting queries. The space/time tradeoff of 2-d orthogonal range counting is well studied. Pătraşcu [17, 18] showed that any data structure using $O(n \lg^{O(1)} n)$ space requires $\Omega(\lg n / \lg \lg n)$ query time. In fact, it is possible to achieve this query time with linear space [7, 14].

For the 2-d range reporting problem, the best known results were given by

[3] In this paper we use $\lg n$ to denote $\log_2 n$.

Chan et al. [6]. Let *occ* denote the output size. Chan et al.'s first solution achieves $O(\lg\lg n + occ \lg\lg n)$ query time with $O(n \lg\lg n)$ space. Their second data structure supports queries in $O(\lg^{\epsilon} n + occ \lg^{\epsilon} n)$ time using linear space.

1.2 Our Results

Let σ be the number of distinct weights, clearly $\sigma \leq n$. Without loss of generality, we assume that weights are drawn from $[1..\sigma]$. In the remainder of this paper, we analyze the time and space costs in terms of n and σ.

For path median and path selection queries, under the pointer machine model, our data structure requires $O(\lg \sigma)$ query time and $O(n \lg \sigma)$ words of space (Section 3). Under the word RAM model, the space cost can be reduced to $O(n)$, preserving the same query time (Section 4.1). The word RAM result significantly improves the best known result [16], in which the data structure achieving $O(\lg n)$ query time uses $O(n \lg^2 n)$ space, and the linear space data structure requires $O(n^{\epsilon})$ query time for any positive constant ϵ.

For path counting queries, our data structure for the pointer machine model supports queries in $O(\lg \sigma)$ time, using $O(n \lg \sigma)$ words of space (Section 3). Our data structure for the word RAM model also supports queries in $O(\lg \sigma)$ time, using linear space only (Section 4.1). The best previous result for the path counting problem is due to Chazelle [8], which requires linear space and $O(\lg n)$ query time. The author showed that any path in a tree T can be partitioned into $O(\lg |T|)$ canonical paths, where $|T|$ is the number of nodes in T. Thus, even if the size of the set of weights is very small, Chazelle's data structure still requires $O(\lg n)$ time to answer a query. Our word RAM result matches the result of Chazelle when σ is close to n, and improves it when σ is much smaller than n. In addition, our techniques are conceptually simple.

To the best of our knowledge, path reporting queries have never been studied before. Assuming that *occ* is the size of output, we give three solutions in this paper. The first one, under the pointer machine model, requires $O(n \lg \sigma)$ words of space and $O(\lg \sigma + occ)$ query time (Section 3). The second one, under the word RAM model, requires $O(n)$ words of space and $O(\lg \sigma + occ \lg \sigma)$ query time (Section 4.1). The last one, also under the word RAM model, requires $O(n \lg\lg \sigma)$ words of space but only $O(\lg \sigma + occ \lg\lg \sigma)$ query time (Section 4.2).

To achieve the above results, we generalize powerful techniques such as the wavelet trees [12] and the technique for the ball-inheritance problem [6]. Previously, these techniques were applied to arrays and 2-d point sets only. Our work is the first that successfully generalizes them to answer path queries, and we expect these to be useful for other similar queries over trees in the future.

2 Properties of Ordinal Trees

In this paper, we take T as an ordinal tree. That is, we choose a node to be the root, and define a left-to-right order among siblings in T. In this section we prove several properties of ordinal trees that are useful in designing our data

structures for path queries.

For any nodes u, v in T, let $P_{u,v}$ denote the set of nodes on the path from u to v. If u and v are the same node, then $P_{u,v}$ contains this node only. For any node u and its ancestor c, we define $A_{u,c}$ to be the set of nodes on the path from u to c, excluding the top node c. We assume a node is also its own ancestor. Thus, $A_{u,u}$ is a valid but empty set. It is clear that, for any nodes u, v in T, $P_{u,v}$ is the disjoint union of $A_{u,c}$, $A_{v,c}$ and $\{c\}$, where c is the lowest common ancestor (LCA) of u and v.

Let $[l, r]$ be a range, where $1 \leq l \leq r \leq \sigma$. Let $R_{l,r}$ denote the set of nodes in T that have a weight in $[l, r]$. For any node-weighted ordinal tree S and any node x in S, we define $d_{l,r}(S, x)$, or the $[l, r]$-*depth* of x in S, to be the number of ancestors of x that have a weight in $[l, r]$. The $[1, \sigma]$-depth of x is equivalent to the depth of x in the tree, so we define $d(S, x)$ to be $d_{1,\sigma}(S, x)$ for simplicity. Also, we define $anc_{l,r}(S, x)$ to be the nearest ancestor of x that has a weight in $[l, r]$. If x has no such ancestor, then $anc_{l,r}(S, x)$ is the root of S. An obvious fact is that $d_{l,r}(S, x) = d_{l,r}(S, anc_{l,r}(S, x))$.

Now consider how to compute the intersection of $R_{l,r}$ and $P_{u,v}$. Providing that c is the lowest common ancestor of u and v, we have

$$\begin{aligned} R_{l,r} \cap P_{u,v} &= R_{l,r} \cap (A_{u,c} \cup A_{v,c} \cup \{c\}) \\ &= (R_{l,r} \cap A_{u,c}) \cup (R_{l,r} \cap A_{v,c}) \cup (R_{l,r} \cap \{c\}); \end{aligned} \tag{1}$$

and its cardinality is

$$\begin{aligned} |R_{l,r} \cap P_{u,v}| &= |(R_{l,r} \cap A_{u,c}) \cup (R_{l,r} \cap A_{v,c}) \cup (R_{l,r} \cap \{c\})| \\ &= |R_{l,r} \cap A_{u,c}| + |R_{l,r} \cap A_{v,c}| + |R_{l,r} \cap \{c\}| \\ &= d_{l,r}(T, u) - d_{l,r}(T, c) + d_{l,r}(T, v) - d_{l,r}(T, c) + \mathbf{1}_{R_{l,r}}(c), \end{aligned} \tag{2}$$

where $\mathbf{1}_{R_{l,r}}(c)$ is equal to 1 if $c \in R_{l,r}$, or equal to 0 if not. In order to compute the cardinality efficiently, we need a fast way to compute $d_{l,r}(T, u)$'s. A naive solution is to store a $d_{l,r}$ value for each node in T. This method takes $O(n)$ space for any range $[l, r]$ so that the overall space cost will be $O(n\sigma)$. To save space, we prove several useful properties of ordinal trees.

We introduce the deletion operation of tree edit distance [3], which can be performed at any non-root node of an ordinal tree. Let u be a non-root node and v be its parent. To delete u, we insert its children in place of u into the list of children of v, keeping the original left-to-right order. For range $[l, r]$, we define $T_{l,r}$ to be the ordinal tree after deleting all the non-root nodes that are not in $R_{l,r}$ from T, where the nodes are deleted in level order. Note that $T_{l,r}$ contains $|R_{l,r}|$ or $|R_{l,r}| + 1$ nodes only, depending on whether the root of T is in $R_{l,r}$. The following lemmas play central roles in our data structures. Their proofs are omitted due to space constraints.

Lemma 1. *For any node u and its arbitrary ancestor c, and any range $[l, r]$, $d_{l,r}(T, u) - d_{l,r}(T, c) = d(T_{l,r}, anc_{l,r}(T, u)) - d(T_{l,r}, anc_{l,r}(T, c))$.*

Lemma 2. *For any node u in T, and any two nested ranges $[l, r] \subseteq [l', r']$, $anc_{l,r}(T, u) = anc_{l,r}(T_{l',r'}, anc_{l',r'}(T, u))$.*

Lemma 3. *For any two ranges $[l_1, r_1]$ and $[l_2, r_2]$, the nodes in both T_{l_1,r_1} and T_{l_2,r_2} have the same relative positions in the pre-order traversal sequences of T_{l_1,r_1} and T_{l_2,r_2}.*

3 A Pointer Machine Data Structure

In this section, we present our data structure for the pointer machine model. Our basic idea is to build a conceptual range tree on $[1, \sigma]$: Starting with $[1, \sigma]$, we keep splitting each range into two child ranges differ by at most 1 in length. Formally, providing that $l < r$, the range $[l, r]$ will be split into child ranges $[l_1, r_1]$ and $[l_2, r_2]$, where $l_1 = l$, $r_1 = \lfloor (l + r)/2 \rfloor$, $l_2 = r_1 + 1$ and $r_2 = r$. This procedure stops when $[1, \sigma]$ has been split into σ leaf ranges of length 1, which are located at the bottom level of the range tree. It is easy to see that each leaf node corresponds to a single value in $[1, \sigma]$.

For each range $[l, r]$ in the range tree, we construct and store $T_{l,r}$ explicitly. On each node in $T_{l,r}$ we store its depth, and a pointer linking to the corresponding node in T. For each non-leaf range $[l, r]$, let $[l_1, r_1]$ and $[l_2, r_2]$ be the child ranges of $[l, r]$. We pre-compute and store $anc_{l_1,r_1}(T_{l,r}, u)$ and $anc_{l_2,r_2}(T_{l,r}, u)$ for each node u in $T_{l,r}$.

We now show how to support path queries using the above data structure.

Lemma 4. *The data structure in this section supports path counting queries in $O(\lg \sigma)$ time, and path reporting queries in $O(\lg \sigma + occ)$ time, where occ denotes the output size of the query.*

Proof. Let u and v be the endpoints of the query path, and let $[s, t]$ be the query range. In the path counting and reporting problems, our objective is to compute $R_{s,t} \cap P_{u,v}$ and its cardinality. We first consider how to compute the cardinality for path counting queries. By the property of range trees, each query range $[s, t] \subseteq [1, \sigma]$ can be represented by the union of m disjoint ranges in the range tree, say $[l_1, r_1], \cdots, [l_m, r_m]$, where $m = O(\lg \sigma)$. Because $R_{s,t} \cap P_{u,v} = \bigcup_{1 \le i \le m} (R_{l_i,r_i} \cap P_{u,v})$, we need only compute the cardinality of $|R_{l_i,r_i} \cap P_{u,v}|$ efficiently, for $1 \le i \le m$. As shown in Algorithm 1, our algorithm accesses the range tree from top to bottom, ending at the ranges completely included in $[s, t]$. When visiting range $[l, r]$, $anc_{l,r}(T, \delta)$ is computed for $\delta \in \{u, v, c\}$. Thus, line 12 computes $|R_{l,r} \cap P_{u,v}|$ in $O(1)$ time when $[l, r] \subseteq [s, t]$. In line 16, the algorithm iteratively computes the nearest ancestors of u, v, c for child ranges. Finally, in line 20, the algorithm recurses on child ranges that intersect with $[s, t]$.

For path reporting queries, as shown in lines 8 to 11, our algorithm traverses from x to y, reporting all the nodes on this path except z. Note that for each node being reported, we report its corresponding node in T by following the pointer saved in it. Finally, we report node c if its weight is in this range.

Algorithm 1 The algorithm for path counting and reporting queries.

```
1: procedure QUERY(u, v, [s, t])                                  ▷ [s, t] ⊆ [1, σ].
2:     c ← LCA(u, v);
3:     return SEARCH([1, σ], u, v, c, c, [s, t]);
4: end procedure
5: procedure SEARCH([l, r], x, y, z, c, [s, t])
6:                                  ▷ x = anc_{l,r}(T, u), y = anc_{l,r}(T, v) and z = anc_{l,r}(T, c).
7:     if [l, r] ⊆ [s, t] then
8:         if the given query is a path reporting query then
9:             report all nodes on the path from x to y except z;
10:            report c if its weight is in [l, r];
11:        end if
12:        return d(T_{l,r}, x) + d(T_{l,r}, y) − 2d(T_{l,r}, z) + 1_{R_{l,r}}(c);
13:                                  ▷ |R_{l,r} ∩ P_{u,v}|, by Lemma 1 and Equation 2.
14:    end if
15:    Let [l_1, r_1] and [l_2, r_2] be the child ranges of [l, r];
16:    δ_i ← anc_{l_i,r_i}(T_{l,r}, δ) for δ = {x, y, z} and i = 1, 2;        ▷ By Lemma 2.
17:    count ← 0;
18:    for i ← 1, 2 do
19:        if [l_i, r_i] ∩ [s, t] ≠ φ then
20:            count ← count+ SEARCH([l_i, r_i], x_i, y_i, z_i, c, [s, t]);
21:        end if
22:    end for
23:    return count;
24: end procedure
```

Now we analyze the time cost of Algorithm 1. First of all, the LCA queries can be supported in constant time and linear space (for a simple implementation, see [2]). We thus need only consider our range tree. For range counting queries, Algorithm 1 accesses $O(\lg \sigma)$ ranges, and it spends constant time on each range. For range reporting queries, Algorithm 1 uses $O(1)$ additional time to report each occurrence. Hence, the query time of range counting is $O(\lg \sigma)$, and the query time of range reporting is $O(\lg \sigma + occ)$, where occ is the output size. □

Lemma 5. *The data structure in this section supports path median and selection queries in $O(\lg \sigma)$ time.*

Proof. Let u and v be the nodes given in the query. Our algorithm for the path median and selection problems is shown in Algorithm 2. The algorithm traverses the range tree from top to bottom, ending at some range of length 1. The procedure SELECT computes the pth smallest weight in $R_{l,r} \cap P_{u,v}$, providing that $1 \le p \le |R_{l,r} \cap P_{u,v}|$. If $l = r$, this weight must be l. Otherwise, let $[l_1, r_1]$ and $[l_2, r_2]$ be the child ranges of $[l, r]$, where $r_1 < l_2$. The algorithm computes $count = |R_{l_1,r_1} \cap P_{u,v}|$ and compares it with p in line 15. If p is not larger than $count$, then the algorithm recurses on $[l_1, r_1]$ in line 18; otherwise the algorithm deducts $count$ from p and recurses on $[l_2, r_2]$ in line 20. The time analysis is similar to Lemma 4. □

Algorithm 2 The algorithm for path median and selection queries.

```
1: procedure QUERY(u, v or u, v, k)                          ▷ 1 ≤ k ≤ |P_{u,v}|.
2:     if the given query is a path median query then
3:         k ← ⌈|P_{u,v}|/2⌉;
4:     end if
5:     c ← LCA(u, v);
6:     return SELECT([1, σ], u, v, c, c, k);
7: end procedure
8: procedure SELECT([l, r], x, y, z, c, p)
9:                          ▷ x = anc_{l,r}(T, u), y = anc_{l,r}(T, v) and z = anc_{l,r}(T, c).
10:    if l = r then
11:        return l;
12:    end if
13:    Let [l_1, r_1] and [l_2, r_2] be the child ranges of [l, r], where r_1 < l_2;
14:    δ_i ← anc_{l_i,r_i}(T_{l,r}, δ) for δ = {x, y, z} and i = 1, 2;        ▷ By Lemma 2.
15:    count ← d(T_{l_1,r_1}, x_1) + d(T_{l_1,r_1}, y_1) − 2d(T_{l_1,r_1}, z_1) + 1_{R_{l_1,r_1}}(c);
16:                          ▷ |R_{l_1,r_1} ∩ P_{u,v}|, by Lemma 1 and Equation 2.
17:    if p ≤ count then
18:        return SELECT([l_1, r_1], x_1, y_1, z_1, c, p);
19:    else
20:        return SELECT([l_2, r_2], x_2, y_2, z_2, c, p − count);
21:    end if
22: end procedure
```

With Lemmas 4 and 5, we can present our result on supporting path queries under the pointer machine model.

Theorem 1. *Under the pointer machine model, a tree on n weighted nodes can be represented in $O(n \lg \sigma)$ words of space to support path median, selection and counting queries in $O(\lg \sigma)$ time, and path reporting queries in $O(\lg \sigma + occ)$ time, where the weights are drawn from $[1, \sigma]$, and occ is the output size of the path reporting query.*

Proof. The claim of query time follows from Lemma 4 and Lemma 5. It suffices to analyze the space cost of our data structure. For a range $[l, r]$ in the range tree, our data structure uses $O(|R_{l,r}|)$ words of space to store $T_{l,r}$, depths of nodes, and the pointers to T and child ranges. Thus the adjunct data structures constructed for each level of the range tree occupy $O(n)$ words in total, and the overall space cost of our data structure is $O(n \lg \sigma)$. □

4 Optimizations in the RAM model

In this section we show how to reduce the space cost of the data structure presented in Section 3. We adopt the word RAM model of computation with word size $w = \Omega(\lg n)$. For the path median, selection and counting problems, we achieve $O(\lg \sigma)$ query time with linear space. For the path reporting problem,

we either require $O(\lg \sigma + occ \lg \sigma)$ query time with linear space, or $O(\lg \sigma + occ \lg \lg \sigma)$ query time with $O(n \lg \lg \sigma)$ space.

4.1 Linear Space, but Slower Reporting

Our starting point for space optimization is the succinct representation of ordinal trees. As shown in Lemma 6, there exists a data structure that encodes a tree on n labeled nodes in $O(n)$ bits of space when the number of distinct labels is constant, and supports a set of basic operations in constant time.

Lemma 6 ([10]). *Let S be an ordinal tree on n nodes, each having a label from an alphabet Σ. S can be represented in $n(\lg |\Sigma| + 2) + O(|\Sigma| n \lg \lg \lg n / \lg \lg n)$ bits to support the following queries in constant time. For simplicity, we assume that S contains node x, and x precedes itself in pre-order.*

- Pre-Rank*(S, x): Return the number of nodes that precede x in pre-order;*
- Pre-Rank*(S, x, c): Return the number of nodes that are labeled with c and precede x in pre-order;*
- Pre-Select*(S, i): Return the ith node in pre-order;*
- Pre-Select*(S, i, c): Return the ith node in pre-order that is labeled with c;*
- Depth*(S, x): Return the depth of x;*
- Ancestor*(S, x, c): Return the lowest ancestor of node x labeled with c.*

We now present our linear space data structure.

Theorem 2. *Under the word RAM model with word size $w = \Omega(\lg n)$, a tree on n weighted nodes can be represented in $O(n)$ words of space to support path median, selection and counting queries in $O(\lg \sigma)$ time, and path reporting queries in $O(\lg \sigma + occ \lg \sigma)$ time, where the weights are drawn from $[1, \sigma]$, and occ is the output size of the path reporting query.*

Proof. For our new data structure, we still build a conceptual range tree as in Section 3. Unlike the previous data structure, we store only the succinct representation of $T_{l,r}$ for each range $[l, r]$ in the range tree. We assign a label to each node in $T_{l,r}$ if range $[l, r]$ is not a leaf range. Let $[l_1, r_1]$ and $[l_2, r_2]$ be child ranges of $[l, r]$, where $r_1 < l_2$. We assign label 0 to the root node of $T_{l,r}$. For each non-root node u in $T_{l,r}$, we assign it label i if $u \in T_{l_i,r_i}$ for $i = 1, 2$. By Lemma 6, the succinct representation for range $[l, r]$ occupies $O(|R_{l,r}|)$ bits. Thus, the space cost of our new data structure is $O(n \lg \sigma / w) = O(n)$ words.

Now consider how to answer path queries. Note that the nodes in the succinct representation are indexed by their ranks in pre-order. Let $[l, r]$ be a non-leaf range, and let $[l_1, r_1]$ and $[l_2, r_2]$ be the child ranges of $[l, r]$, where $r_1 < l_2$. Suppose node u is in $T_{l,r}$, and node u' is the node in T_{l_i,r_i} that corresponds to u for $i = 1$ or 2. We show that, once the pre-order rank of u in $T_{l,r}$ is known, the pre-order of u' in T_{l_i,r_i} can be computed in constant time, and vice versa. By the construction of our linear space data structure, T_{l_i,r_i} contains all the nodes

in $T_{l,r}$ that have label 0 or i. By Lemma 3, these nodes have the same relative positions in the pre-order sequences of T_{l_i,r_i} and $T_{l,r}$. We thus have that

$$\textsc{Pre-Rank}(T_{l,r}, u, 0) + \textsc{Pre-Rank}(T_{l,r}, u, i) = \textsc{Pre-Rank}(T_{l_i,r_i}, u'). \quad (3)$$

Applying this formula, it takes constant time to convert the pre-order rank of the same node between two adjacent levels in the range tree.

Since the DEPTH operation is provided, we need only consider how to compute the nearest ancestors of u, v, c for child ranges (Line 16 in Algorithm 1 and line 14 in Algorithm 2). For $\delta = \{x, y, z\}$ and $i = 1, 2$, we can compute $anc_{l_i,r_i}(T_{l,r}, \delta)$ by $\textsc{Ancestor}(T_{l,r}, \delta, i)$. If δ has no ancestor with label i in $T_{l,r}$, then $anc_{l_i,r_i}(T_{l,r}, \delta)$ is the root node of $T_{l,r}$.

It is more complicated to deal with path reporting queries. Unlike the data structure described in Section 3, given a node u in some $T_{l,r}$ that need be reported, we cannot directly find its corresponding node in T by following an appropriate pointer, as we cannot afford storing these pointers. Instead, we compute the pre-order rank of u in the tree constructed for the parent range of $[l, r]$ using Equation 3, and repeat this process until we reach the root range in the range tree. This procedure takes $O(\lg \sigma)$ time for each node to report.

To analyze the query time, we observe that, for path median, selection and counting queries, our linear space data structure still uses constant time on each visited range. Hence, these queries can be answered in $O(\lg \sigma)$ time. For path reporting queries, this data structure requires $O(\lg \sigma)$ additional time for each node to report. Thus, path reporting queries can be answered in $O(\lg \sigma + occ \lg \sigma)$ time, where *occ* is the output size. □

4.2 Slightly More Space, Much Faster Reporting

As shown above, the bottleneck of path reporting queries in our linear space data structure is that it requires $O(\lg \sigma)$ time to find the corresponding node in T for each node in the answer. We apply the technique in [6] to speed up this process. Due to space limitations, we only provide a sketch of the proof.

Theorem 3. *Under the word RAM model with word size $w = \Omega(\lg n)$, a tree on n weighted nodes can be represented in $O(n \lg \lg \sigma)$ words of space to support path reporting queries in $O(\lg \sigma + occ \lg \lg \sigma)$ time, where the weights are drawn from $[1, \sigma]$, and occ is the output size of the path reporting query.*

Proof (Sketch). We maintain a second index for the nodes in T. For each node u in T, we index u by its weight b and its pre-order rank in $T_{b,b}$. Suppose the algorithm decides to report u when accessing range $[l, r]$. We need only find the leaf range corresponding to the weight and the pre-order rank of u in the tree constructed for this leaf range.

Consider the ranges at levels i and $i + \Delta$, where $1 \leq i < i + \Delta \leq \lg \sigma + 1$. Using the space efficient array in [6], which occupies $O(n\Delta)$ bits of space, we can compute in constant time the pre-order rank of each node u at level $i + \Delta$, providing that its pre-order rank at level i is known. For positive integer i, we

define $f(i)$ to be the position in the binary representation of i such that the $(f(i)+1)$th least significant bit is the rightmost 1. For level $1 \leq i \leq \lg \sigma$, using the above method, we maintain an array with $\Delta = 2^{f(\lg \sigma + 1 - i)}$. Given a node u to report, we access these arrays to compute the pre-order rank of u at the leaf range. Since each jumping between two levels increases $f(\lg \sigma + 1 - i)$ by 1, this procedure takes $O(\lg \lg \sigma)$ time. On the other hand, the space cost of these arrays is at most $\frac{1}{w} \sum_{1 \leq i \leq \lg \sigma} O(n \cdot 2^{f(\lg \sigma + 1 - i)}) = O(n \lg \lg \sigma)$ words. □

References

1. Alon, N., Schieber, B.: Optimal preprocessing for answering on-line product querie. Tech. rep., Tel Aviv University (1987)
2. Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: LATIN. pp. 88–94 (2000)
3. Bille, P.: A survey on tree edit distance and related problems. Theor. Comput. Sci. 337(1-3), 217–239 (2005)
4. Brodal, G.S., Gfeller, B., Jørgensen, A.G., Sanders, P.: Towards optimal range medians. Theor. Comput. Sci. 412(24), 2588–2601 (2011)
5. Brodal, G.S., Jørgensen, A.G.: Data structures for range median queries. In: Proc. 20th Annual International Symposium on Algorithms and Computation, Lecture Notes in Computer Science, vol. 5878, pp. 822–831. Springer Verlag, Berlin (2009)
6. Chan, T.M., Larsen, K.G., Pătraşcu, M.: Orthogonal range searching on the RAM, revisited. In: Symposium on Computational Geometry. pp. 1–10 (2011)
7. Chan, T.M., Pătraşcu, M.: Counting inversions, offline orthogonal range counting, and related problems. In: SODA. pp. 161–173 (2010)
8. Chazelle, B.: Computing on a free tree via complexity-preserving mappings. Algorithmica 2, 337–361 (1987)
9. Gagie, T., Puglisi, S.J., Turpin, A.: Range quantile queries: Another virtue of wavelet trees. In: SPIRE. pp. 1–6 (2009)
10. Geary, R.F., Raman, R., Raman, V.: Succinct ordinal trees with level-ancestor queries. ACM Transactions on Algorithms 2(4), 510–534 (2006)
11. Gfeller, B., Sanders, P.: Towards optimal range medians. In: ICALP (1). pp. 475–486 (2009)
12. Grossi, R., Gupta, A., Vitter, J.S.: High-order entropy-compressed text indexes. In: SODA. pp. 841–850 (2003)
13. Hagerup, T.: Parallel preprocessing for path queries without concurrent reading. Inf. Comput. 158(1), 18–28 (2000)
14. JáJá, J., Mortensen, C.W., Shi, Q.: Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In: ISAAC. pp. 558–568 (2004)
15. Jørgensen, A.G., Larsen, K.G.: Range selection and median: Tight cell probe lower bounds and adaptive data structures. In: SODA. pp. 805–813 (2011)
16. Krizanc, D., Morin, P., Smid, M.H.M.: Range mode and range median queries on lists and trees. Nord. J. Comput. 12(1), 1–17 (2005)
17. Pătraşcu, M.: Lower bounds for 2-dimensional range counting. In: STOC. pp. 40–46 (2007)
18. Pătraşcu, M.: (Data) STRUCTURES. In: FOCS. pp. 434–443 (2008)
19. Petersen, H.: Improved bounds for range mode and range median queries. In: SOFSEM. pp. 418–423 (2008)
20. Petersen, H., Grabowski, S.: Range mode and range median queries in constant time and sub-quadratic space. Inf. Process. Lett. 109(4), 225–228 (2009)