Succinct Data Structures for Path Queries[⋆]

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. Consider a tree T on n nodes, each having a weight drawn from $[1..\sigma]$. In this paper, we design succinct data structures to encode T using $nH(W_T) + o(n \lg \sigma)$ bits of space, such that we can support path counting queries in $O(\frac{\lg \sigma}{\lg \lg n} + 1)$ time, path reporting queries in $O((occ+1)(\frac{\lg \sigma}{\lg \lg n}+1))$ time, and path median and path selection queries in $O(\frac{\lg \sigma}{\lg \lg \sigma})$ time, where $H(W_T)$ is the entropy of the multiset of the weights of the nodes in T. Our results not only improve the best known linear space data structures [15], but also match the lower bounds for these path queries [18, 19, 16] when $\sigma = \Omega(n/\text{polylog}(n))$.

1 Introduction

As fundamental structures in computer science, trees are widely used in modeling and representing different types of data in numerous applications. In many scenarios, objects are modeled as nodes of trees, and properties of objects are stored as weights or labels on the nodes. Thus researchers have studied the problem of preprocessing a weighted tree in which each node is assigned a weight, in order to support different path queries, where a path query computes a certain function over the weights of the nodes along a given path in the tree [1, 7, 13, 17, 15]. As an example, Chazelle [7] studied how to sum up weights along a query path efficiently.

In this paper, we design succinct data structures to maintain a weighted tree T such that we can support several path queries. We consider *path counting*, *path reporting*, *path median*, and *path selection* queries, where the first two kinds of queries are also called *path search* queries. The formal definitions of these queries are listed below. For the sake of simplicity, let $P_{u,v}$ be the set of nodes on the path from u to v, where u, v are nodes in T. Also, let $R_{p,q}$ be the set of nodes in T that have a weight in the range $[p, q]$.

– Path Counting Query: Given two nodes u and v, and a range $[p, q]$, return the cardinality of $P_{u,v} \cap R_{p,q}$;
– Path Reporting Query: Given two nodes u and v, and a range $[p, q]$, return the nodes in $P_{u,v} \cap R_{p,q}$;

[⋆] This work was supported by NSERC and the Canada Research Chairs Program.

– Path Selection Query: Given two nodes u and v, and an integer $1 \leq k \leq |P_{u,v}|$, return the k-th smallest weight in the multiset of the weights of the nodes in $P_{u,v}$. If k is fixed to be $\lceil |P_{u,v}|/2 \rceil$, then path selection queries become path median queries.

As mentioned in [15], these queries generalize two-dimensional range counting, two-dimensional range reporting, range median, and range selection queries.

We represent our tree T as an *ordinal (ordered) tree.* This does not significantly impact the space cost, which is dominated by storing the weights of nodes. In addition, we assume that the weights are drawn from $[1..\sigma]$, or rank space. Thus a query range is an integral one, denoted by $[p..q]$ in the rest of this paper.

1.1 Previous Work

The path counting problem was studied by Chazelle [7], whose formulation is different from ours: Weights are assigned to edges instead of nodes, and a query asks for the number of edges on a given path that have a weight in a given range. Chazelle designed a linear space data structure to support queries in $O(\lg n)$ time. His technique, relying on tree partition, requires $O(\lg n)$ query time for an arbitrary set of weights. He et al. [15] slightly improved Chazelle's result. Under the standard word RAM model, their linear space data structure requires $O(\lg \sigma)$ query time when the weights are drawn from a set of σ distinct values.

The path reporting problem was proposed by He et al. [15], who obtained two solutions under the the word RAM model: One requires $O(n)$ words of space and $O(\lg \sigma + occ \lg \sigma)$ query time, the other requires $O(n \lg \lg \sigma)$ words of space but only $O(\lg \sigma + occ \lg \lg \sigma)$ query time, where occ is the size of output, and σ is the size of the set of weights.

The path median problem was first studied by Krizanc et al. [17], who gave two solutions for this problem. The first one, occupying $O(n \lg^2 n)$ words of space, supports queries in $O(\lg n)$ time. The second one requires $O(b \lg^3 n / \lg b)$ query time and $O(n \lg_b n)$ space, for any $2 \leq b \leq n$. Moreover, a linear space data structure that supports queries in $O(n^\epsilon)$ time can be obtained by setting $b = n^\epsilon / \lg^2 n$ for some small constant ϵ. He et al. [15] significantly improved these results. Under the standard word RAM model, their linear space data structure requires $O(\lg \sigma)$ query time only, where σ is the size of the set of weights.

Succinct representation of static trees. The problem of encoding a static tree succinctly has been studied extensively. For unlabeled trees, a series of succinct representations have been designed [12, 14, 8, 9, 21]. For labeled trees, Geary et al. [12] presented a data structure to encode a tree on n nodes, each having a label drawn from an alphabet of size σ, supporting a set of navigational operations in constant time. The overall space cost is $n(\lg \sigma + 2) + O(\sigma n \lg \lg \lg n / \lg \lg n)$ bits, which is much more than the information-theoretic lower bound of $n \lg \sigma + 2n - O(\lg n)$ bits when σ is large. Ferragina et al. [10] and Barbay et al. [2, 3] designed data structures for labeled trees using space close to the information-theoretic minimum, but supporting a more restricted set of operations.

Succinct data structures for range queries. For two-dimensional orthogonal range search queries, the best known result is due to Bose et al. [5].

Path Query Type	Best Known		**new**	
	Query Time	Space	Query Time	Space
Counting	$O(\lg \sigma)$	$O(n)$ words	$O(\frac{\lg \sigma}{\lg \lg n} + 1)$	$nH(W_T) + o(n \lg \sigma)$ bits
Reporting	$O((occ + 1) \lg \sigma)$	$O(n)$ words	$O((occ + 1)(\frac{\lg \sigma}{\lg \lg n} + 1))$	$nH(W_T) + o(n \lg \sigma)$ bits
Median / Selection	$O(\lg \sigma)$	$O(n)$ words	$O(\frac{\lg \sigma}{\lg \lg \sigma})$	$nH(W_T) + o(n \lg \sigma)$ bits

Table 1. Best known linear space data structures and new results for path queries. We assume that *occ* is the size of output. All the results listed in the table are obtained under the standard word RAM model with word size $w = \Omega(\lg n)$. Note that $H(W_T)$ is at most $\lg \sigma$, which is $O(w)$.

They presented a data structure that encodes a set of n points in an $n \times n$ grid using $n \lg n + o(n \lg n)$ bits to support range counting queries in $O(\frac{\lg n}{\lg \lg n})$ time, and range reporting queries in $O((occ + 1) \cdot \frac{\lg n}{\lg \lg n})$ time, where *occ* is the size of output. For range median and range selection queries, Brodal et al. [6] claimed that their data structure achieving $O(\frac{\lg n}{\lg \lg n})$ query time uses $O(n)$ words of space. A careful analysis reveals that their results are even better than claimed: in rank space, their data structure occupies $n \lg n + o(n \lg n)$ bits of space only, which is succinct.

1.2 Our Results

In this paper, we design succinct data structures for path queries over an ordinal tree T on n nodes, each having a weight drawn from $[1..\sigma]$. Let W_T be the multiset consisting of the weights of the nodes in T. Our data structures occupy $nH(W_T) + o(n \lg \sigma)$ bits of space, which is close to the information-theoretic lower bound ignoring lower-order terms, achieving faster query time compared to the best known $\Theta(n \lg n)$-bit data structures [15], which are not succinct. We summarize our main results in Table 1, along with the best known results [15].

It is noteworthy that our data structures for path counting, path median and path selection queries match the lower bounds for related range queries [18, 19, 16] when $\sigma = \Omega(n/\text{polylog}(n))$.

2 Preliminaries

2.1 Succinct Indexes of Bit Vectors and Integer Sequences

The bit vector is a key structure in many applications. Given a bit vector $B[1..n]$, we consider the following operations. For $\alpha \in \{0, 1\}$, $\textsc{Rank}_\alpha(B, i)$ returns the number of α-bits in $B[1..i]$, and $\textsc{Select}_\alpha(B, i)$ returns the position of the i-th α-bit in B. The following lemma addresses the problem of succinct representations of bit vectors.

Lemma 1 ([20]). *Let $B[1..n]$ be a bit vector with m 1-bits. B can be represented in $\lg \binom{n}{m} + O(n \lg \lg n / \lg n)$ bits to support* Rank*,* Select*, and the access to each bit in constant time.*

The RANK/SELECT operations can be generalized to sequences of integers. Given an integer sequence $S[1..n]$ in which the elements are drawn from $[1..s]$, we define $\text{RANK}_\alpha(S,i)$ and $\text{SELECT}_\alpha(S,i)$ by extending the definitions of these operations on bit vectors, letting $\alpha \in [1..s]$. Ferragina et al. [11] presented the following lemma:

Lemma 2 (Theorem 3.1 in [11]). *Let $S[1..n]$ be a sequence of integers in $[1..s]$, where $1 \le s \le \sqrt{n}$. S can be represented in $nH_0(S)+O(s(n\lg\lg n)/\log_s n)$ bits, where $H_0(S)$ is the zeroth empirical entropy of S, to support* RANK*,* SELECT*, and the access to any $O(\log_s n)$ consecutive integers in constant time.*

Note that the last operation is not explicitly mentioned in the original theorem. However, recall that S is divided into blocks of size $\lfloor \frac{1}{2}\log_s n \rfloor$, and the entry of table E that corresponds to some block G can be found in constant time. We can explicitly store in each entry the content of the block that corresponds to this entry. Thus the content of any block can be retrieved in constant time. It is easy to verify that the extra space cost is $o(n)$ bits.

When s is sufficiently small, say $s = O(\lg^\epsilon n)$ for some constant $0 < \epsilon < 1$, the second term in the space cost is upper bounded by $O(s(n\lg\lg n)/\log_s n) = O(n(\lg\lg n)^2/\lg^{1-\epsilon} n) = o(n)$ bits.

The RANK operations can be further generalized to COUNT operations, where $\text{COUNT}_\beta(S,i)$ is defined to be $\sum_{\alpha=1}^{\beta} \text{RANK}_\alpha(S,i)$. Bose et al. [5] presented the following lemma:

Lemma 3 (Lemma 3 in [5]). *Let $S[1..n]$ be a sequence of integers in $[1..t]$, where $t = O(\lg^\epsilon n)$ for some constant $0 < \epsilon < 1$. Providing that any $\lceil \log^\lambda n \rceil$ consecutive integers in S can be retrieved in $O(1)$ time for some constant $\lambda > \epsilon$, S can be indexed using $o(n)$ additional bits to support* COUNT *in constant time.*

2.2 Properties of Ordinal Trees and Forests

In this subsection, we review and extend the discussion of ordinal trees by He et al. [15]. For any two nodes u and v in a given ordinal tree T, they denoted by $P_{u,v}$ the set of nodes on the path from u to v. For any node u and its ancestor w, they denoted by $A_{u,w}$ the set of nodes on the path from u to w, excluding the top node w. That is, $A_{u,w} = P_{u,w} - \{w\}$.

We further consider ordinal forests. An ordinal forest F is a left-to-right ordered list of ordinal trees. The preorder traversal sequence of F is defined to be the left-to-right ordered concatenation of the preorder traversal sequences of the ordinal trees in F. Nodes in the same ordinal tree or the same ordinal forest are identified by their preorder ranks, i.e., their positions in the preorder traversal sequence.

He et al. [15] made use of the deletion operation of tree edit distance [4]. They performed this operation on non-root nodes only. Let u be a non-root node, and let v be its parent. To delete u, its children are inserted in place of u into the list of children of v, preserving the original left-to-right order. We also perform this operation on roots. Let u be a root. An ordinal forest of trees rooted at the children of u occurs after deleting u, where the trees are ordered by their roots.

To support path queries, He et al. [15] defined notation in terms of weights. Let T be an ordinal tree, whose weights of nodes are drawn from $[1..\sigma]$. For any range $[a..b] \subseteq [1..\sigma]$, they defined $R_{a,b}$ to be the set of nodes in T that have a weight in $[a..b]$. They also defined $anc_{a,b}(T, x)$ to be the lowest ancestor of x that has a weight in $[a..b]$. Here we slightly modify their definition. We assume that $anc_{a,b}(T, x) = \text{NULL}$ if x has no ancestor that has a weight in $[a..b]$. To be consistent, we assume that the depth of the NULL node is 0. In addition, we define $F_{a,b}$, which is similar to $T_{a,b}$ in [15], to be the ordinal forest after deleting all the nodes that are not in $R_{a,b}$ from T, where the nodes are deleted from bottom to top. Note that there is a one-to-one mapping between the nodes in $R_{a,b}$ and the nodes in $F_{a,b}$. As proved in [15], the nodes in $R_{a,b}$ and the nodes in $F_{a,b}$ that correspond to them have the same relative positions in the preorder of T and the preorder of $F_{a,b}$.

2.3 Succinct Ordinal Tree Representation Based on Tree Covering

In this subsection, we briefly summarize the tree-covering based representation of ordinal trees [12, 14, 8, 9], which we extend in Section 3 to support more powerful operations such as Count and Summarize.

The tree-covering based representation was proposed by Geary et al. [12] to represent an ordinal tree succinctly to support navigational operations. Let T be an ordinal tree on n nodes. A tree cover of T with a given parameter M is essentially a set of $O(n/M)$ *cover elements*, each being a connected subtree of size $O(M)$. These subtrees, being either disjoint or joined at the common root, cover all the nodes in T.

Geary et al. [12] proposed an algorithm to cover T with *mini-trees* or *tier-1 subtrees* for $M = \max\{\lceil (\lg n)^4 \rceil, 2\}$. Again, they apply the algorithm to cover each mini-tree with *micro-trees* or *tier-2 subtrees* for $M' = \max\{\lceil (\lg n)/24 \rceil, 2\}$. For $k = 1, 2$, the nodes that are roots of the tier-k subtrees are called *tier-k roots*. Note that a tier-k root can be the root node of multiple tier-k subtrees, and a tier-1 root must also be a tier-2 root.

To use tree-covering based representation to support more navigational operations, He et al. [14, Definition 4.22] proposed the notion of tier-k *preorder segments*, where the preorder segments are defined to be maximal substrings of nodes in the preorder sequence that are in the same mini-tree or micro-tree. Farzan and Munro [8] further modified the tree-covering algorithm. Their algorithm computes a tree cover such that nodes in a cover element are distributed into a constant number of preorder segments, as proved by Farzan et al. [9]. These results are summarized in the following lemma:

Lemma 4 (Theorem 1 in [8] and Lemma 2 in [9]). *Let T be an ordinal tree on n nodes. For a parameter M, the tree decomposition algorithm in [8] covers the nodes in T by $\Theta(n/M)$ cover elements of size at most $2M$, all of which are pairwise disjoint other than their root nodes. In addition, nodes in one cover element are distributed into $O(1)$ preorder segments.*

The techniques in [12, 14, 8, 9] encode an unlabeled ordinal tree on n node in $2n + o(n)$ bits to support in constant time a set of operations related to nodes, tier-k subtrees and tier-k roots. For example, given an arbitrary node x, we can compute its depth, and find its i-th ancestor. Given two nodes x and y, we can compute their lowest common ancestor (LCA).

For $k = 1$ or 2, the tier-k subtrees are ordered and specified by their ranks. The following operations can be performed in constant time: For each tier-k subtree, we can find its root node, compute the preorder segments contained in it, and select the i-th node in preorder that belongs to this subtree. For each micro-tree, we can compute the encoding of its structure. For each node that is not a tier-k root, we can find the tier-k subtree to which it belongs, and its relative preorder rank in this tier-k subtree.

Similarly, for $k = 1$ or 2, the tier-k roots are ordered in preorder and specified by their ranks. Let r_i^1/r_i^2 denote the i-th tier-1/tier-2 root. Given a node x, we can compute its rank if x is a tier-k root, or determine that x is not a tier-k root. Conversely, given the rank of a tier-k root, we can compute its preorder rank.

3 Succinct Ordinal Trees over Alphabets of Size $O(\lg^\epsilon n)$

In this section, we present a succinct data structure to encode an ordinal tree T on n weighted nodes, in which the weights are drawn from $[1..t]$, where $t = O(\lg^\epsilon n)$ for some constant $0 < \epsilon < 1$. This succinct representation occupies $n(H_0(PLS_T) + 2) + o(n)$ bits of space to support a set of operations in constant time, where PLS_T is the preorder label sequence of T. As the entropy of W_T only depends on the frequencies of labels, we have $H_0(PLS_T) = H(W_T)$. Thus, the space cost can also be represented as $n(H(W_T) + 2) + o(n)$ bits. However, we use $H_0(PLS_T)$ in this section to facilitate space analysis. The starting points of our succinct representation are Geary et al.'s [12] succinct ordinal tree, and Bose et al.'s [5] data structure for orthogonal range search on a narrow grid.

In the rest of this section, we consider the operations listed below. For the sake of convenience, we call a node x an α-node or an α-ancestor if the weight of x is α. Also, we assume that nodes x, y are contained in T, α and β are in $[1..t]$, a node precedes itself in preorder, and a node is its own 0-th ancestor.

- $\text{Pre-Rank}_\alpha(T, x)$: Return the number of α-nodes that precede x in preorder;
- $\text{Pre-Select}_\alpha(T, i)$: Return the i-th α-node in preorder;
- $\text{Pre-Count}_\beta(T, x)$: Return the number of nodes preceding x in preorder that have a weight $\leq \beta$;
- $\text{Depth}_\alpha(T, x)$: Return the number of α-ancestors of x (excluding the root);
- $\text{Lowest-Anc}_\alpha(T, x)$: Return the lowest α-ancestor of x if such an α-ancestor exists, otherwise return the root of T;
- $\text{Count}_\beta(T, x)$: Return the number of nodes on the path from x to the root of T (excluding the root) that have a weight $\leq \beta$;
- $\text{Summarize}(T, x, y)$: Given that node y is an ancestor of x, this operation returns t bits, where the α-th bit is 1 if and only if there exists an α-node on the path from x to y (excluding y), for $1 \leq \alpha \leq t$.

We first compute the mini-micro tree cover of T using Lemma 4 for $M = \lceil \lg^2 n \rceil$ and $M' = \lceil \lg^\lambda n \rceil$ for some $\max\{\epsilon, \frac{1}{2}\} < \lambda < 1$. It is easy to verify that, with our choice of M and M', the operations in Subsection 2.3, which are related to tree nodes, tier-k subtrees and tier-k roots, can still be supported in constant time, using $2n + o(n)$ bits of space. Let n_1 and n_2 be the numbers of tier-1 roots and tier-2 roots. By Lemma 4, they are bounded by $O(n/M)$ and $O(n/M')$, respectively. To store the weights, we encode $PLS_T[1..n]$ using Lemma 2, occupying $nH_0(PLS_T) + o(n)$ bits of space. These are our main data structures that encode the structure of tree and the weights of nodes. We design auxiliary data structures of $o(n)$ bits to support operations.

Lemma 5. Summarize *can be supported in constant time, using* $o(n)$ *additional bits of space.*

Proof. To support Summarize, we need the following operations. For simplicity, we assume that x is a tier-1 root, and y is a tier-2 root.

- Mini-Depth(T, x): Return the number of tier-1 roots on the path from x to the root of T;
- Mini-Anc(T, x, i): Return the $(i + 1)$-th tier-1 root on the path from x to the root of T, providing that $0 \le i <$ Mini-Depth(T, x);
- Micro-Depth(T, y): Return the number of tier-2 roots on the path from y to the root of T;
- Micro-Anc(T, y, i): Return the $(i + 1)$-th tier-2 root on the path from y to the root of T, providing that $0 \le i <$ Micro-Depth(T, y).

We only show how to support the first two operations. The other operations can be supported in the same way. We compute tree T' by deleting all the nodes other than the tier-1 roots from T. Clearly T' has n_1 nodes. To convert the nodes in T' and the tier-1 roots in T, we construct and store a bit vector $B_1[1..n]$, in which $B_1[i] = 1$ iff the i-th node in preorder of T is a tier-1 root. By Lemma 1, B_1 can be encoded in $o(n)$ bits to support Rank and Select in constant time. Note that the i-th node in T' corresponds to the tier-1 root of preorder rank j in T, providing that $B_1[j]$ is the i-th 1-bit in B_1. Thus the conversion can be done in constant time using Rank and Select. Applying the techniques in Subsection 2.3, we encode T' in $2n_1 + o(n_1) = o(n)$ bits to support Mini-Depth and Mini-Anc directly.

We then construct the following auxiliary data structures:

- A two-dimensional array $D[1..n_1, 0..\lceil \lg n \rceil]$, in which $D[i, j]$ stores a bit vector of length t whose α-th bit is 1 iff there exists an α-node on the path from r_i^1 to Mini-Anc$(T, r_i^1, 2^j)$.
- A two-dimensional array $E[1..n_2, 0..3\lceil \lg \lg n \rceil]$, in which $E[i, j]$ stores a bit vector of length t whose α-th bit is 1 iff there exists an α-node on the path from r_i^2 to Micro-Anc$(T, r_i^2, 2^j)$.
- A table F that stores for every possible weighted tree p on $\le 2M'$ nodes whose weights are drawn from $[1..t]$, every integer i, j in $[1..2M']$, a bit vector of length t whose α-th bit is 1 iff there exists an α-node on the path from the i-th node in preorder of p to the j-th node in preorder of p.

– All these paths do not include the top nodes.

Now we analyze the space cost. D occupies $n_1 \times (\lceil \lg n \rceil + 1) \times t = O(n/\lg^{1-\epsilon} n) = o(n)$ bits. E occupies $n_2 \times (3\lceil \lg\lg n \rceil + 1) \times t = O(n \lg\lg n/\lg^{\lambda-\epsilon} n) = o(n)$ bits. F has $O(n^{1-\delta})$ entries for some $\delta > 0$, each occupying t bits. Therefore, D, E, F occupy $o(n)$ bits in total.

Let r_a^1 and r_b^2 be the roots of the mini-tree and the micro-tree containing x, respectively. Also, let r_c^1 and r_d^2 be the roots of the mini-tree and the micro-tree containing y, respectively. Suppose that $a \neq c$ (the case that $a = c$ can be handled similarly). We define the following two nodes

$$r_e^1 = \textsc{Mini-Anc}(T, r_a^1, \textsc{Mini-Depth}(T, r_a^1) - \textsc{Mini-Depth}(T, r_c^1) - 1),$$
$$r_f^2 = \textsc{Micro-Anc}(T, r_e^1, \textsc{Micro-Depth}(T, r_e^1) - \textsc{Micro-Depth}(T, r_d^2) - 1).$$

To answer operation $\textsc{Summarize}(T, x, y)$, we split the path from x to y into $x - r_b^2 - r_a^1 - r_e^1 - r_f^2 - y$. We compute for each part a bit vector of length t whose α-th bit is 1 iff there exists an α-node on this part. The answer to the Summarize operation is the result of bitwise OR operation on these bit vectors.

The first part and the last part are paths in the same micro-tree. Thus their bit vectors can be computed by table lookups on F. We only show how to compute the bit vector of the third part; the bit vectors of the second and the fourth part can be computed in a similar way. Setting $w = \textsc{Mini-Depth}(T, r_a^1) - \textsc{Mini-Depth}(T, r_e^1)$, $j = \lfloor \lg w \rfloor$, tier-2 root $r_g^1 = \textsc{Mini-Anc}(T, r_a^1, w - 2^j)$, the path from r_a^1 to r_e^1 can be represented as the union of the path from r_a^1 to $\textsc{Mini-Anc}(T, r_a^1, 2^j)$ and path from r_g^1 to $\textsc{Mini-Anc}(T, r_g^1, 2^j)$. Therefore the vector of this part is equal to $D[a, j]$ OR $D[g, j]$. □

Due to space limitations, the details of supporting all the other operations listed in this subsection are omitted. To analyze the space cost of our data structure, we observe that the main cost is due to the sequence PLS_T and Subsection 2.3, which occupy $n(H_0(PLS_T)+2)+o(n)$ bits in total. The other auxiliary data structures occupy $o(n)$ bits of space only. We thus have the following lemma.

Lemma 6. *Let T be an ordinal tree on n node, each having a weight drawn from $[1..t]$, where $t = O(\lg^\epsilon n)$ for some constant $0 < \epsilon < 1$. T can be represented in $n(H_0(PLS_T) + 2) + o(n)$ bits of space to support* Pre-Rank, Pre-Select, Depth, Lowest-Anc, Count, Summarize, *and the navigational operations described in Subsection 2.3 in constant time.*

4 Supporting Path Queries

We now describe the data structures for general path search queries. The basic idea is to build a conceptual range tree on $[1..\sigma]$ with the branching factor $t = \lceil \lg^\epsilon n \rceil$ for some constant $0 < \epsilon < 1$. In this range tree, a range $[a..b]$ corresponds to the nodes in T that have a weight in $[a..b]$, which are also said to be contained in the range $[a..b]$. We say that a range $[a..b]$ is empty if $a > b$.

In addition, the length of a range $[a..b]$ is defined to be $b - a + 1$. Empty ranges have non-positive lengths.

We describe this range tree level by level. At the beginning, we have the top level only, which contains the root range $[1..\sigma]$. Starting from the top level, we keep splitting each range in current lowest level into t child ranges, some of which might be empty. Presuming that $a \leq b$, a range $[a..b]$ will be split into t child ranges $[a_1..b_1], [a_2..b_2], \cdots, [a_t..b_t]$ such that, for $a \leq j \leq b$, the nodes that have a weight j are contained in the child range of subscript $\lceil \frac{t(j-a+1)}{b-a+1} \rceil$. Doing a little math, we have that

$$a_i = \min \left\{ a \leq j \leq b \middle| \lceil \frac{t(j-a+1)}{b-a+1} \rceil = i \right\} = \lfloor \frac{(i-1)(b-a+1)}{t} \rfloor + a;$$
$$b_i = \max \left\{ a \leq j \leq b \middle| \lceil \frac{t(j-a+1)}{b-a+1} \rceil = i \right\} = \lfloor \frac{i(b-a+1)}{t} \rfloor + a - 1.$$

This procedure stops when all the ranges in current lowest level have length 1, which form the bottom level of the conceptual range tree.

We list all the levels from top to bottom. The top level is the first level, and the bottom level is the h-th level, where $h = \lceil \log_t \sigma \rceil + 1$ is the height of the conceptual range tree. It is clear that each value in $[1..\sigma]$ occurs in exactly one range at each level, and each leaf range corresponds to a single value in $[1..\sigma]$.

For each level in the range tree other than the bottom one, we create and explicitly store an auxiliary tree. Let T_l denote the auxiliary tree created for the l-th level. To create T_l, we list all the non-empty ranges at the l-th level in increasing order. Let them be $[a_1..b_1], [a_2..b_2], \cdots, [a_m..b_m]$. We have that $a_1 = 1, b_m = \sigma$, and $b_i = a_{i+1} - 1$ for $i = 1, 2, \cdots, m-1$. Initially, T_l contains only a root node, say r. For $i = 1, 2, \cdots, m$, we compute forest F_{a_i,b_i} as described in Subsection 2.2, and then insert the trees in F_{a_i,b_i} into T_l in the original left-to-right order. That is, for $i = 1, 2, \cdots, m$, from left to right, we append the root node of each tree in F_{a_i,b_i} to the list of children of r. Thus, for $l = 1, 2, \cdots, h-1$, there exists a one-to-one mapping between the nodes in T and the non-root nodes in T_l, and the root of T_l corresponds to the NULL node.

We assign weights to the nodes in T_l. The root of T_l is assigned 1. For each node x in T, we denote by x_l the node at level l that corresponds to x. By the construction of the conceptual range tree, the range containing x at level $l+1$ has to be a child range of the range containing x at level l. We assign a weight α to x_l if the range at level $l+1$ is the α-th child range of the range at level l. As the result, we obtain an ordinal tree T_l on $n+1$ nodes, each having a weight drawn from $[1..t]$. We explicitly store T_l using Lemma 6.

Equations 1 and 2 capture the relationship between x_l and x_{l+1}, for each node x in T and $l \in [1..h-1]$. Remind that nodes are identified by their preorder ranks. Presuming that node x is contained in a range $[a..b]$ at level l, and contained in a range $[a_\gamma..b_\gamma]$ at level $l+1$, which is the γ-th child range of $[a..b]$, we have

$$rank = \textsc{Pre-Rank}_\gamma(T_l, |R_{1,a-1}| + 1)$$
$$x_{l+1} = |R_{1,a_\gamma - 1}| + 1 + \textsc{Pre-Rank}_\gamma(T_l, x_l) - rank \qquad (1)$$
$$x_l = \textsc{Pre-Select}_\gamma(T_l, x_{l+1} - |R_{1,a_\gamma - 1}| - 1 + rank) \qquad (2)$$

With these data structures, we can support path counting and reporting.

Theorem 1. *Let T be an ordinal tree on n nodes, each having a weight drawn from $[1..\sigma]$. Under the word RAM model with word size $w = \Omega(\lg n)$, T can be encoded in (a) $n(H(W_T) + 2) + o(n)$ bits when $\sigma = O(\lg^\epsilon n)$ for some constant $0 < \epsilon < 1$, or (b) $nH(W_T) + O(\frac{n \lg \sigma}{\lg \lg n})$ bits otherwise, supporting path counting queries in $O(\frac{\lg \sigma}{\lg \lg n}+1)$ time, and path reporting queries in $O((occ+1)(\frac{\lg \sigma}{\lg \lg n}+1))$ time, where $H(W_T)$ is the entropy of the multiset of the weights of the nodes in T, and occ is the output size of the path reporting query.*

Proof. Let $P_{u,v} \cap R_{p,q}$ be the query. $P_{u,v}$ can be partitioned into $A_{u,w}$, $A_{v,w}$ (see Subsection 2.2 for the definitions of $A_{u,w}$ and $A_{v,w}$) and $\{w\}$, where $w = LCA(u, v)$. It is trivial to compute $\{w\} \cap R_{p,q}$. We only consider how to compute $A_{u,w} \cap R_{p,q}$ and its cardinality. The computation of $A_{v,w} \cap R_{p,q}$ is similar.

Our recursive algorithm is shown in Algorithm 1. Providing that the range $[a..b]$ is at the l-th level of the conceptual range tree, $c = |R_{1,a-1}|+1$, $d = |R_{1,b}|+1$, and x, z are the nodes in T_l that corresponds to $anc_{a,b}(T, u)$ and $anc_{a,b}(T, w)$, the procedure $\textsc{Search}([a..b], l, c, d, x, z, [p..q])$ returns the cardinality of $A_{u,w} \cap R_{a,b} \cap R_{p,q}$, and reports the nodes in the intersection if the given query is a path reporting query. To compute $A_{u,w} \cap R_{p,q}$ and its cardinality, we only need call $\textsc{Search}([1..\sigma], 1, 1, n + 1, u + 1, w + 1, [p..q])$.

Now let us analyze the procedure $\textsc{Search}$. In line 4, we maximize $[e'..f'] \subseteq [e..f]$ such that $[e'..f']$ can be partitioned into child ranges of subscripts α to β. To compute α (the computation of β is similar), let $\alpha' = \lceil \frac{t(e-a+1)}{b-a+1} \rceil$, the subscript of the child range containing the nodes that have a weight e. α is set to be α' if $e \le a_{\alpha'}$, otherwise α is set to be $\alpha'+1$. If $\alpha \le \beta$, we accumulate $|A_{u,w} \cap R_{a_\alpha, b_\beta}|$ in line 6, where $\textsc{Count}_{\alpha,\beta}(T, x)$ is defined to be $\textsc{Count}_\beta(T, x) - \textsc{Count}_{\alpha-1}(T, x)$. We still need compute $A_{u,w} \cap (R_{e,f} - R_{a_\alpha,b_\beta}) = A_{u,w} \cap R_{e,f} \cap (R_{a_{\alpha-1},b_{\alpha-1}} \cup R_{a_{\beta+1},b_{\beta+1}})$. In the loop starting at line 14, for $\gamma \in \{\alpha - 1, \beta + 1\}$, we check whether $[a_\gamma..b_\gamma] \cap [e..f] \neq \phi$. If the intersection is not empty, then we compute $A_{u,w} \cap R_{e,f} \cap R_{a_\gamma,b_\gamma}$ and accumulate its cardinality by a recursive call in line 19. In lines 16 to 18, we adjust the parameters to satisfy the restriction of the procedure $\textsc{Search}$. We increase c and d as shown in lines 16 and 17, because for $i = 1$ to t, the nodes in T that have a weight in $[a_i..b_i]$ correspond to the nodes in T_l that have a weight i and a preorder rank in $[c + 1..d]$. In line 18, for $\delta = \{x, z\}$, we set $\delta_\gamma = \textsc{Lowest-Anc}_\gamma(T_l, \delta)$. If δ_γ is the root of T_l, say $\delta_\gamma = 1$, then δ_γ is set to be 1. Otherwise, we compute δ_γ by Equation 1.

For path reporting queries, in lines 7 to 12, we report all nodes on the path $A_{u,w}$ that have a weight in $[a_\alpha..b_\beta]$. In line 8, we compute *sum*, a bit vector of length t, whose i-th bit is one iff $A_{u,w} \cap R_{a_i,b_i} \neq \phi$. Since $t = o(\lg n)$, using word operations, we can enumerate each 1-bit whose index is between α and β in constant time per 1-bit. For each 1-bit, assuming its index is i, we find all the nodes in $A_{u,w} \cap R_{a_i,b_i}$ using $\textsc{Lowest-Anc}$ operations repeatedly. At this moment, we only know the preorder ranks of the nodes to report in T_l. We have to convert it to the preorder rank in T_1 by using Equation 2 $l-1$ times per node.

We now analyze the time cost of our algorithm. By the construction of the conceptual range tree, any query range $[p..q]$ will be partitioned into $O(h)$ ranges in the range tree. It takes constant time to access a range in the range tree, since, by Subsection 2.3 and Lemma 6, all the operations on T_l take $O(1)$ time. Hence, it takes $O(h) = O(\frac{\lg \sigma}{\lg \lg n} + 1)$ time to compute $|A_{u,w} \cap R_{p,q}|$. For path reporting queries, it takes $O(h)$ time to report each node in $A_{u,w} \cap R_{p,q}$. Each 1-bit we enumerate in line 9 corresponds to one or more node to report, so the cost of enumeration is dominated by other parts of the algorithm. Therefore, it takes $O(h + |A_{u,w} \cap R_{p,q}|h) = O((|A_{u,w} \cap R_{p,q}| + 1)(\frac{\lg \sigma}{\lg \lg n} + 1))$ time to compute $A_{u,w} \cap R_{p,q}$.

Due to space limitations, the analysis of the space cost is omitted. □

Algorithm 1 The algorithm for path search queries on $A_{u,w}$.

```
 1: procedure SEARCH($[a..b], l, c, d, x, z, [p..q]$)
 2:     Let $[a_1..b_1], [a_2..b_2], \cdots, [a_t..b_t]$ be the child ranges of $[a..b]$;
 3:     $count \leftarrow 0$, $[e..f] = [a..b] \cap [p..q]$;
 4:     $\alpha = \min\{1 \le i \le t | e \le a_i \le b_i\}$, $\beta = \max\{1 \le i \le t | a_i \le b_i \le f\}$;
 5:     if $\alpha \le \beta$ then
 6:         $count \leftarrow count + \text{COUNT}_{\alpha,\beta}(T_l, x) - \text{COUNT}_{\alpha,\beta}(T_l, z)$;
 7:         if the given query is a path reporting query then
 8:             $sum \leftarrow \text{SUMMARIZE}(T_l, x, z)$;
 9:             for each $i \in [\alpha..\beta]$ that the $i$-th bit of $sum$ is one do
10:                 report all $i$-nodes on the path from $x$ to $z$ (excluding $z$);
11:             end for
12:         end if
13:     end if
14:     for $\gamma \in \{\alpha - 1, \beta + 1\}$ do
15:         if $1 \le \gamma \le t$ and $[a_\gamma..b_\gamma] \cap [e..f] \ne \phi$ then
16:             $c_\gamma \leftarrow c + \text{PRE-COUNT}_{\gamma-1}(T_l, d) - \text{PRE-COUNT}_{\gamma-1}(T_l, c)$;
17:             $d_\gamma \leftarrow c + \text{PRE-COUNT}_{\gamma}(T_l, d) - \text{PRE-COUNT}_{\gamma}(T_l, c)$;
18:             $\delta_\gamma \leftarrow$ the preorder rank of $\text{LOWEST-ANC}_\gamma(T_l, \delta)$ in $T_{l+1}$, for $\delta \in \{x, z\}$;
19:             $count \leftarrow count +$ SEARCH($[a_\gamma..b_\gamma], l+1, c_\gamma, d_\gamma, x_\gamma, z_\gamma, [p..q]$);
20:         end if
21:     end for
22:     return $count$;
23: end procedure
```

To support path median and path selection queries, we apply the techniques we have developed in this section to generalize the linear space data structure of Brodal et al.'s [6] for range selection queries. We have the following theorem, and leave the proof in the full version of this paper.

Theorem 2. *Let T be an ordinal tree on n nodes, each having a weight drawn from $[1..\sigma]$. Under the word RAM model with word size $w = \Omega(\lg n)$, T can be represented in (a) $n(H(W_T) + 2) + o(n)$ bits when $\sigma = O(1)$, or (b) $nH(W_T) +$*

$O(\frac{n \lg \sigma}{\lg \lg \sigma})$ bits otherwise, supporting path median and path selection queries in $O(\frac{\lg \sigma}{\lg \lg \sigma})$ time.

References

1. Alon, N., Schieber, B.: Optimal preprocessing for answering on-line product queries. Tech. rep., Tel Aviv University (1987)
2. Barbay, J., Golynski, A., Munro, J.I., Rao, S.S.: Adaptive searching in succinctly encoded binary relations and tree-structured documents. In: CPM. pp. 24–35 (2006)
3. Barbay, J., He, M., Munro, J.I., Rao, S.S.: Succinct indexes for strings, binary relations and multilabeled trees. ACM Transactions on Algorithms 7(4), 52 (2011)
4. Bille, P.: A survey on tree edit distance and related problems. Theor. Comput. Sci. 337(1-3), 217–239 (2005)
5. Bose, P., He, M., Maheshwari, A., Morin, P.: Succinct orthogonal range search structures on a grid with applications to text indexing. In: WADS. pp. 98–109 (2009)
6. Brodal, G.S., Gfeller, B., Jørgensen, A.G., Sanders, P.: Towards optimal range medians. Theor. Comput. Sci. 412(24), 2588–2601 (2011)
7. Chazelle, B.: Computing on a free tree via complexity-preserving mappings. Algorithmica 2, 337–361 (1987)
8. Farzan, A., Munro, J.I.: A uniform approach towards succinct representation of trees. In: SWAT. pp. 173–184 (2008)
9. Farzan, A., Raman, R., Rao, S.S.: Universal succinct representations of trees? In: ICALP (1). pp. 451–462 (2009)
10. Ferragina, P., Luccio, F., Manzini, G., Muthukrishnan, S.: Compressing and indexing labeled trees, with applications. J. ACM 57(1) (2009)
11. Ferragina, P., Manzini, G., Mäkinen, V., Navarro, G.: Compressed representations of sequences and full-text indexes. ACM Transactions on Algorithms 3(2) (2007)
12. Geary, R.F., Raman, R., Raman, V.: Succinct ordinal trees with level-ancestor queries. ACM Transactions on Algorithms 2(4), 510–534 (2006)
13. Hagerup, T.: Parallel preprocessing for path queries without concurrent reading. Inf. Comput. 158(1), 18–28 (2000)
14. He, M., Munro, J.I., Rao, S.S.: Succinct ordinal trees based on tree covering. In: ICALP. pp. 509–520 (2007)
15. He, M., Munro, J.I., Zhou, G.: Path queries on weighted trees. In: ISAAC. pp. 558–568 (2011)
16. 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)
17. 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)
18. Pătraşcu, M.: Lower bounds for 2-dimensional range counting. In: STOC. pp. 40–46 (2007)
19. Pătraşcu, M.: Unifying the landscape of cell-probe lower bounds. SIAM J. Comput. 40(3), 827–847 (2011)
20. Raman, R., Raman, V., Satti, S.R.: Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. ACM Transactions on Algorithms 3(4) (2007)
21. Sadakane, K., Navarro, G.: Fully-functional succinct trees. In: SODA. pp. 134–149 (2010)