Path Queries in Weighted Trees

by

Gelin Zhou

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2012

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

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 various path queries, for which a certain function over the weights of the nodes along a given query path in the tree is computed [3, 14, 22, 26].

In this thesis, we consider the problem of supporting several various path queries over a tree on n weighted nodes, where the weights are drawn from a set of σ distinct values. 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 [26].

We also consider the path counting query and the path reporting query, where a path counting query asks for the number of nodes on a query path whose weights are in a query range, and a path reporting query requires to report these nodes. Our linear space data structure supports path counting queries with $O(\lg \sigma)$ query time. This matches the result of Chazelle [14] when σ is close to n, and has better performance when σ is significantly smaller than n. 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 $O(n \lg \lg \sigma)$ words of space. These are the first data structures that answer path reporting queries.

Acknowledgements

First and foremost, I offer my sincere gratitude to my supervisor, Professor J. Ian Munro, who has supported me throughout my research with his patience and knowledge while allowing me to work in my own way.

I wish also to express my sincere acknowledgement to Assistant Professor Meng He, with whom I have worked closely. As a friendly mentor and a careful collaborator, he has made as much effort as he could to help me improve my research.

I would like to thank my friends in the Algorithm and Complexity Lab. I have had much fun and have learnt a lot from the discussion with them.

Finally, I would like to thank my readers, Professor Ming Li and Professor Alejandro López-Ortiz, for their valuable comments to improve this thesis.

Dedication

This is dedicated to my parents.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Path Queries

Trees are fundamental structures in computer science, being widely used in modeling and representing different types of data in numerous computer applications. The Unix file system can be visualized as an ordinal tree (a rooted tree of arbitrary degree with ordered children) in which internal nodes correspond to folders, and leaf nodes correspond to files or folders. Deleting a folder in the file system is equivalent to removing the subtree rooted at the node corresponding to the folder. The tree network is a representative type of network topology, which contains a central node as the root, and several star networks as the subtrees of the root. This type of network topology is ideal when workstations are distributed into groups that occupy relatively small physical regions. XML documents can be essentially modeled as ordinal trees in which each node is assigned a tag drawn from a tag set. One can analyze XML documents using navigational operations on the corresponding tag trees.

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 various *path queries*, for which a certain function over the weights of the nodes along a given query path in the tree is computed [3, 14, 22, 26].

In this thesis, we design data structures to maintain a weighted tree on n nodes such

that we can support various path queries. The *path median* query asks for the median weight on a path between two given nodes, and the *path selection* query returns the k-th smallest weight. The *path counting* query asks for the number of nodes on a query path whose weights are in a query range, while the *path reporting* query requires to report these nodes. When the given tree 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.

1.2 Computational Models

We consider three computational models in this thesis. The first one is the pointer machine model [35], which is very restricted. In this model, a data structure is represented as a directed graph on nodes, each storing a constant number of data values and containing a constant number of pointers to other nodes. This model allows algorithms to access a node by following a pointer to this node, to compare two values or two pointers, and to create new nodes, values and pointers. In previous formulations [9, 23], algorithms can perform arithmetic operations on data values but not on pointers. The running time of an algorithm in this model is measured by the number of accesses to nodes and the number of operations performed, while the space cost is measured by the number of nodes in the directed graph.

The word random access machine (word RAM) model [1, 17], is a realistic model for modern computers. In this model, data is stored in an array of memory locations, or *machine words*. Each word consists of $w = \Omega(\lg n)$ bits [1], where n is the size of the input instance. A word can be addressed by its subscript, and can be read or rewritten in $O(1)$ time. This model also allows algorithms to perform arithmetic and bitwise operations on a constant number of words in $O(1)$ time. The running time of an algorithm in this model is measured by the number of memory accesses and the number of operations performed on words, while the space cost is measured by the number of words used.

It is not surprising that the word RAM model is stronger than the pointer machine model. Pointers under the pointer machine model can be regarded as subscripts of memory locations under the word RAM model. Thus, the word RAM model is able to simulate

[1] In this thesis we use $\lg n$ to denote $\log_2 n$.

algorithms on pointer machines, and to further perform arithmetic and bitwise operations on pointers, which are not allowed under the pointer machine model.

The two-dimensional version of the orthogonal range reporting problem, which is introduced later in Section 1.3.1, serves as an example to show the advantage of the word RAM model. Under the pointer machine model, Chazelle [16] proved that any data structure for the two-dimensional range reporting problem using $O(\lg^{O(1)} n + occ)$ query time requires $\Omega(n \lg n / \lg \lg n)$ space, where occ is the size of output. However, this lower bound was "surpassed" by Alstrup et al. [4] under the word RAM model, who obtained a data structure that occupies $O(n \lg^{\epsilon} n)$ words of space for any constant $\epsilon > 0$, and supports queries in $O(\lg \lg n + occ)$ time.

Finally, the cell probe model [36] is the strongest model of computation we consider. This model is similar to the word RAM model, except that there is no charge for computation on words. The running time under this model is measured by the number of memory accesses only. Since this model is strictly stronger than the other two models, lower bounds obtained under this model also apply to the pointer machine model and the word RAM model. Unfortunately, it is very hard to prove meaningful lower bounds under this model, even for simple data structure problems.

1.3 Previous Work

In this section, we review the previous work related to the path queries introduced in Section 1.1. We start with the special cases of these queries, which are two-dimensional orthogonal range counting and reporting queries, and range median and selection queries. We survey these range queries in Sections 1.3.1 and 1.3.2. Then, we review the results of path counting queries, and the ones of path median and selection queries in Sections 1.3.3 and 1.3.4, respectively. We do not have a section for path reporting queries. As far as we know, they have not been studied before.

1.3.1 Two-Dimensional Orthogonal Range Searching

As a fundamental problem in computational geometry, the two-dimensional orthogonal range searching problem arises from databases and geographic information system (GIS)

applications. In this problem, we maintain N, a set of points on an $n \times n$ grid, such that we can support the following range queries:

- Range Counting Query: Given R, a rectangle whose edges are parallel to the axes, return the cardinality of $N \cap R$.
- Range Reporting Query: Given R, a rectangle whose edges are parallel to the axes, return the points in $N \cap R$.

Two-dimensional orthogonal range counting and reporting queries are special cases of path counting and reporting queries, where the input trees are paths.

The time-space tradeoff of the two-dimensional orthogonal range counting problem has been extensively studied. The standard range tree [6, 28, 27] requires $O(n \lg n)$ words of space, and supports queries in $O(\lg n)$ time. Later, Chazelle [15] designed a linear space data structure that achieves the same query time. The query time has been further reduced to $O(\lg n / \lg \lg n)$, preserving the same space cost [24, 8, 12]. These linear space data structures are optimal, since Pătraşcu [31, 32] has proven that any data structure using $O(n \lg^{O(1)} n)$ bits of space requires $\Omega(\lg n / \lg \lg n)$ query time.

Chan and Pătraşcu [12] also studied the construction time. Their data structure supports queries in $O(\lg n / \lg \lg n)$ time with linear space, and takes only $O(n\sqrt{\lg n})$ time in preprocessing, while the earlier data structures with logarithmic query time require $O(n \lg n)$ construction time. The results on the two-dimensional orthogonal range counting problem are summarized in Table 1.1.

Reference	Model	Space	Query Time	Construction Time
[6, 28, 27]	Pointer Machine	$O(n \lg n)$	$O(\lg n)$	$O(n \lg n)$
[15]	RAM	$O(n)$	$O(\lg n)$	$O(n \lg n)$
[24, 8]	RAM	$O(n)$	$O(\lg n / \lg \lg n)$	$O(n \lg n)$
[12]	RAM	$O(n)$	$O(\lg n / \lg \lg n)$	$O(n\sqrt{\lg n})$
[31, 32]	Cell Probe	$O(n \lg^{O(1)} n)$	$\Omega(\lg n / \lg \lg n)$	–

Table 1.1: Results on the two-dimensional orthogonal range counting problem.

The two-dimensional orthogonal range reporting problem has also been studied heavily. Here we denote by *occ* the size of output. Under the pointer machine model, the standard

range tree [6, 28, 27] implies a solution using $O(n \lg n)$ space and $O(\lg n + occ)$ query time. Chazelle [13] provided a data structure that answers queries in $O(\lg n + occ)$ time using $O(n \lg n / \lg \lg n)$ space. This is optimal due to a lower bound later proved by Chazelle [16], which indicates that any data structure for this problem using $O(\lg^{O(1)} n + occ)$ query time requires $\Omega(n \lg n / \lg \lg n)$ space.

Under the word RAM model, the best time-space tradeoff for this basic problem is still open. Chazelle [15] presented a linear space data structure that supports queries in $O(\lg n + occ \lg^{\epsilon} n)$ time for any constant $\epsilon > 0$, which was later reduced to $O(\lg n / \lg \lg n + occ \lg^{\epsilon} n)$ by Nekrich [29]. Overmars [30] gave a method using $O(n \lg n)$ words of space and $O(\lg \lg n + occ)$ query time. The query time is optimal under the cell probe model for the data structures using $O(n \lg^{O(1)} n)$ bits of space. This is proved by a reduction from the colored predecessor search problem [33]. Alstrup et al. [4] presented two data structures. The first one occupies $O(n \lg^{\epsilon} n)$ space for any constant $\epsilon > 0$, and achieves the optimal $O(\lg \lg n + occ)$ query time. The second one requires $O(n \lg \lg n)$ space, and answers queries in $O(\lg^2 \lg n + occ \lg \lg n)$ time. The latest results were obtained by Chan et al. [11], whose first solution achieves $O(\lg \lg n + occ \lg \lg n)$ query time with $O(n \lg \lg n)$ space, and the second solution supports queries in $O(\lg^{\epsilon} n + occ \lg^{\epsilon} n)$ time using linear space. The results on the two-dimensional orthogonal range reporting problem are summarized in Table 1.2.

Reference	Model	Space	Query Time
[6, 28, 27]	Pointer Machine	$O(n \lg n)$	$O(\lg n + occ)$
[13]	Pointer Machine	$O(n \lg n / \lg \lg n)$	$O(\lg n + occ)$
[16]	Pointer Machine	$\Omega(n \lg n / \lg \lg n)$	$O(\lg^{O(1)} n + occ)$
[15]	RAM	$O(n)$	$O(\lg n + occ \lg^{\epsilon} n)$
[29]	RAM	$O(n)$	$O(\lg n / \lg \lg n + occ \lg^{\epsilon} n)$
[30]	RAM	$O(n \lg n)$	$O(\lg \lg n + occ)$
[4]	RAM	$O(n \lg^{\epsilon} n)$	$O(\lg \lg n + occ)$
[4]	RAM	$O(n \lg \lg n)$	$O(\lg^2 \lg n + occ \lg \lg n)$
[11]	RAM	$O(n \lg \lg n)$	$O(\lg \lg n + occ \lg \lg n)$
[11]	RAM	$O(n)$	$O(\lg^{\epsilon} n + occ \lg^{\epsilon} n)$
[33]	Cell Probe	$O(n \lg^{O(1)} n)$	$\Omega(\lg \lg n + occ)$

Table 1.2: Results on the two-dimensional orthogonal range reporting problem.

1.3.2 Range Median and Selection

The range median and selection problem was first proposed by Krizanc et al. [26]. In this problem, an unsorted array of n elements is given, and a query asks for the median or the k-th smallest element in a range. These queries are special cases of path median and selection queries.

Krizanc et al. [26] presented the earliest time-space tradeoffs for this problem. The first one supports queries in $O(\lg n)$ time, and occupies $O(n \lg^2 n / \lg \lg n)$ space. The second one answers queries in constant time, using $O(n^2 \lg \lg n / \lg n)$ space. The third one requires $O(b \lg^2 n / \lg b)$ query time and $O(n \lg_b n)$ space, for any $2 \leq b \leq n$. If b is set to be $n^\epsilon / \lg n$ for a small constant $\epsilon > 0$, then the space cost becomes linear, but the query time grows to $O(n^\epsilon)$. The second solution of Krizanc et al.'s [26] has been slightly improved by Petersen and Grabowski [34], whose data structure with constant query time requires $O(n^2 \lg^2 \lg n / \lg^2 n)$ words of space. The first and the third solutions of Krizanc et al.'s [26] have been significantly improved by the linear space data structures developed recently. Gfeller and Sanders [20] presented a solution to answer a query in $O(\lg n)$ time. Gagie et al. [18] considered this problem in terms of σ, the number of distinct weights, and designed a data structure based on wavelet trees that supports range selection queries in $O(\lg \sigma)$ time. The best upper bound was achieved by Brodal et al. [9, 10], which requires $O(\lg n / \lg \lg n)$ 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 [25]. The results on the range median and selection problem are summarized in Table 1.3.

1.3.3 Path Counting

The problem of supporting path counting queries was studied by Chazelle [14]. 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. Chazelle designed a linear space data structure to support queries in $O(\lg n)$ time, which is based on tree partition. He proved that any tree T containing at least two edges can be partitioned into two subtrees that contain at least one-third of the edges of T. Based on this lemma, an emulation dag of the input tree can be constructed, in which each edge corresponds to a canonical path in the tree. Chazelle further showed that any path in the input tree can

Reference	Model	Space	Query Time	Restrictions
[26]	Pointer Machine	$O(n \lg^2 n / \lg \lg n)$	$O(\lg n)$	–
[26]	Pointer Machine	$O(n^2 \lg \lg n / \lg n)$	$O(1)$	–
[26]	Pointer Machine	$O(n \lg_b n)$	$O(b \lg^2 n / \lg b)$	$2 \leq b \leq n$
[26]	Pointer Machine	$O(n)$	$O(n^\epsilon)$	–
[34]	RAM	$O(n^2 \lg^2 \lg n / \lg^2 n)$	$O(1)$	–
[20]	RAM	$O(n)$	$O(\lg n)$	–
[18]	RAM	$O(n)$	$O(\lg \sigma)$	–
[9, 10]	RAM	$O(n)$	$O(\lg n / \lg \lg n)$	–
[25]	Cell Probe	$O(n \lg^{O(1)} n)$	$\Omega(\lg n / \lg \lg n)$	–

Table 1.3: Results on the range median and selection problem.

be partitioned into $O(\lg n)$ canonical paths. He finally obtained the data structure by a generalized range tree, along with a compaction technique in [15]. The bottleneck of the time cost is due to the number of canonical paths in partitions of query paths, so this approach requires $O(\lg n)$ time for an arbitrary set of weights.

1.3.4 Path Median and Path Selection

The path median and selection problem introduced in Section 1.1 was also proposed by Krizanc et al. [26], who presented two solutions for this problem. The first one supports queries in $O(\lg n)$ time, 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 b is set to be $n^\epsilon / \lg^2 n$ for some small constant $\epsilon > 0$, then the space cost becomes linear, but the query time is $O(n^\epsilon)$. These are the best known results for the path median and selection problem. We summarize their results in Table 1.4, along with our improvements.

The approach of Krizanc et al.'s [26] is also based on tree partition. Their data structures maintain a set of subpaths in which the weights of the nodes are sorted, such that any query path can be divided into the disjoint union of two or more subpaths in the set. Thus the answer to the query can be obtained by performing a binary search on these subpaths. The time and space cost is determined by the number of subpaths required to partition a query path.

Reference	Model	Space	Query Time	Restrictions
[26]	Pointer Machine	$O(n \lg^2 n)$	$O(\lg n)$	–
[26]	Pointer Machine	$O(n \lg_b n)$	$O(b \lg^3 n / \lg b)$	$2 \leq b \leq n$
[26]	Pointer Machine	$O(n)$	$O(n^\epsilon)$	–
new	Pointer Machine	$O(n \lg \sigma)$	$O(\lg \sigma)$	–
new	RAM	$O(n)$	$O(\lg \sigma)$	–

Table 1.4: Results on the path median and selection problem.

1.4 Our Contributions

The primary contribution of this thesis is the technique of *tree extraction*, which plays a central role in developing the data structures for path queries. This technique is inspired by the deletion operation of tree edit distance [7], though its usage is completely different from computing the edit distance between two labeled ordinal trees. The basic idea of tree extraction is to extract a subset of nodes from an ordinal tree or an ordinal forest, retaining some relative properties among the nodes in this subset. This technique allows us to perform divide-and-conquer approaches, in a space-efficient way, on the set of the weights rather than the structure of the input tree, which is completely different from the tree partition based approaches used in [14, 26]. Following this technique, we obtain the results as follows:

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 (Chapter 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 [26], 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 under the pointer machine model supports queries in $O(\lg \sigma)$ time, using $O(n \lg \sigma)$ words of space (Chapter 3). Our data structure under 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 [14], which requires linear space and $O(\lg n)$ query time. He showed that any path in a tree on n nodes can be partitioned into $O(\lg n)$ canonical paths. 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 thesis. The first one, under the pointer machine model, requires $O(n \lg \sigma)$ words of space and $O(\lg \sigma + occ)$ query time (Chapter 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 [21] and the technique for the ball-inheritance problem [11]. Previously, these techniques were applied to arrays and two-dimensional 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.

The rest of this thesis is organized as follows: Chapter 2 presents the technique of tree extraction, and analyzes path queries using this technique. Chapter 3 describes our data structures for path queries under the pointer machine model. Chapter 4 shows how to optimize the data structures under the word RAM model. Finally, Chapter 5 concludes this thesis with a summarization and some open problems.

The results presented in this thesis have been published in ISAAC'11 [23].

Chapter 2

Supporting Path Queries Using Tree Extraction

In this chapter, we present the main technique used in this thesis, tree extraction. This technique is defined in terms of the deletion operation of tree edit distance [7]. First of all, in Section 2.1, we give the formal definitions of the path queries we consider in this thesis. In Section 2.2, we review several concepts related to ordinal trees and ordinal forests. We then define and analyze tree extraction in Section 2.3, and finally apply this technique to the path queries in Section 2.4.

2.1 The Path Query Problems

In the path query problems introduced in Section 1.1, we maintain a given tree on n weighted nodes, where the weights are drawn from a set of σ distinct values, 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 in the multiset of the weights of these nodes.
- Path Selection Query: This type of queries is a natural extension of path median queries. We are given two nodes u and v, and an integer $k > 0$, and we need return

the k-th smallest weight in the multiset of the weights of the nodes 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 and v, and a range $[p, q]$, 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 and v, and a range $[p, q]$, return the set of nodes on the path from u to v whose weights are in this range.

As mentioned in Section 1.1, these queries generalize range median, range selection, two-dimensional range counting and range reporting queries.

Note that σ is the number of distinct weights, clearly $\sigma \leq n$. Without loss of generality, we assume that weights are drawn from $[1..\sigma]$, or rank space. Thus any query range is an integral one, denoted by $[p..q]$ in the rest of this thesis. We later analyze the time and space cost in terms of n and σ.

2.2 Ordinal Trees and Forests

In this thesis, we take the given tree as an ordinal one. That is, the tree is rooted, and a left-to-right order is defined among siblings. The preorder traversal sequence of an ordinal tree is defined recursively as follows.

Definition 2.2.1. *The preorder traversal sequence of a subtree rooted at a leaf node consists of the leaf node only. The preorder traversal sequence of a subtree rooted at an internal node v is a sequence of nodes that starts with v, followed by the left-to-right ordered concatenation of the preorder traversal sequences of the subtrees rooted at the children of v.*

For example, the preorder traversal sequence of the ordinal tree T shown in Figure 2.1(a) is $abcdefg$.

We also consider ordinal forests. An ordinal forest F is defined to be a left-to-right ordered list of ordinal trees. As a special case, an ordinal tree is an ordinal forest containing only a single ordinal tree. For any node u in F, the depth of u in F is equivalent to the depth of u in the ordinal tree containing u. Similarly, a node v is an ancestor of u in F if and only if they are in the same ordinal tree, and v is an ancestor of u in that tree. Note

that we assume a node is its own ancestor. In addition, 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.

An ordinal forest can be regarded as an ordinal tree in which the root is a non-removable dummy node, and the list of children of the dummy node is exactly the list of roots of the ordinal forest. To be consistent, the dummy node is not taken into account for the preorder traversal sequence of the tree rooted at the dummy node, or the depths of the nodes in that tree. In the rest of this thesis, we make use of both views of ordinal forests interchangeably.

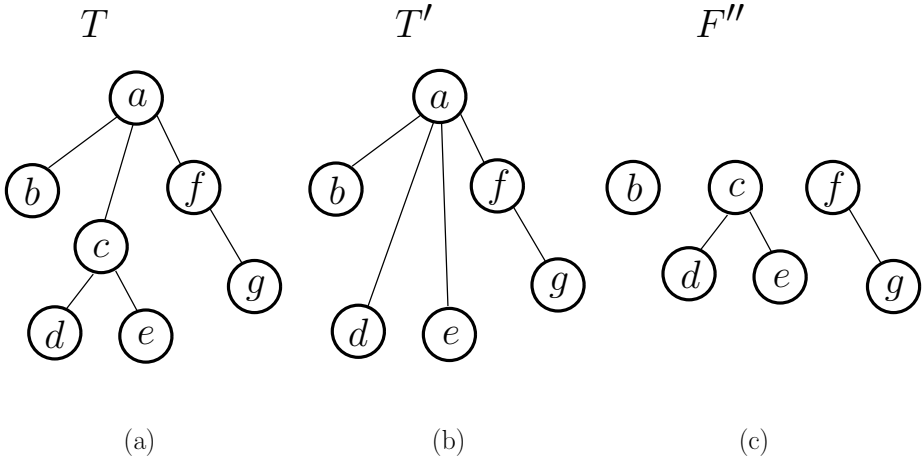

Figure 2.1: (a) An ordinal tree T whose nodes are identified by the letters on them. (b) The ordinal tree T' obtained by deleting a non-root node c from T. (c) The ordinal forest F'' obtained by deleting the root node a from T.

We introduce the deletion operation of tree edit distance [7]. Unlike the original definition, here we can delete any node from an ordinal forest, besides the roots of the ordinal trees in the forest. The formal definition is shown as follows.

Definition 2.2.2. *Suppose we would like to delete a node u from an ordinal forest F, where u is contained in an ordinal tree T.*

- *Case 1: u is a non-root node of T. Let v be the parent node of u. To delete u, we insert its children in place of u into the list of children of v, preserving the original*

left-to-right order.

- *Case 2: u is the root node of T. If T contains u only, we simply dispose T, and remove it from the list of ordinal trees of F. Otherwise, we insert the subtrees rooted at the children of u in place of T into the list of ordinal trees of F, preserving the original left-to-right order.*

For example, the ordinal tree T' shown in Figure 2.1(b) is obtained by deleting a non-root node c from T, while the ordinal forest F'' in Figure 2.1(c) is obtained by deleting the root node a. Note that in both examples, the remaining nodes are identified by the same letters as the corresponding nodes in T.

We have the following properties of the deletion operation.

Proposition 2.2.3. *The deletion operation preserves (a) the ancestor-descendant relationship, and (b) the relative positions in preorder among the remaining nodes.*

As shown in Figure 2.1, node a is an ancestor of node d in both T and T', node c is an ancestor of node e in both T and F'', and node f is ancestor of node g in all these trees and forests. In addition, the preorder traversal sequence of T', $abdefg$, is the one of T with c being removed, while the preorder traversal sequence of F'', $bcdefg$, is the one of T with a being removed.

2.3 Tree Extraction

We first give the formal definition of tree extraction, which is based on the deletion operation described in Definition 2.2.2.

Definition 2.3.1. *Let F be an ordinal forest and let $V(F)$ be the set of nodes in F. For any set $X \subseteq V(F)$, we denote by F_X the ordinal forest obtained by deleting all the nodes that are not in X from F, where the nodes are deleted from bottom to top. F_X is called the X-extraction of F.*

To clarify notation, the nodes in F are denoted by lowercase letters, while the nodes in F_X are denoted by lowercase letters with a subscript X. To illustrate the one-to-one

mapping between the nodes in X and the nodes in F_X, we denote by u_X a node in F_X if and only if its *corresponding* node in F is denoted by u.

We further define enhanced depth and ancestor operators with respect to sets of nodes. For any node $u \in V(F)$, and any set of nodes $X \subseteq V(F)$, we define $d_X(F, u)$, or the X-*depth* of u in F, to be the number of ancestors of u that belong to X. The $V(F)$-depth of u is equivalent to the depth of u in F, which is denoted by $d(F, u)$. Also, we define $anc_X(F, u)$ to be the nearest ancestor of u that belongs to X. If u has no such ancestor, then $anc_X(F, u)$ returns `dummy`, which is the dummy node. For completeness, we define $d_X(F, \texttt{dummy}) = 0$ and $anc_X(F, \texttt{dummy}) = \texttt{dummy}$. An obvious fact is that $d_X(F, u) = d_X(F, anc_X(F, u))$, for any node $u \in V(F)$, and any set of nodes $X \subseteq V(F)$.

Based on the concepts of depths, ancestors and preorder traversal sequences, the following lemmas capture some essential properties of tree extraction.

Lemma 2.3.2. *For any set of nodes $X \subseteq V(F)$, if F contains an ordinal tree T only, and the root of T is contained in X, then F_X contains an ordinal tree only.*

Proof. Let u be the root of T, clearly u is the ancestor of the other nodes in F. u_X is also the ancestor of the other nodes in F_X, since u is in X, and the deletion operation does not change the ancestor-descendant relationship among the remaining nodes. Thus F_X contains a single ordinal tree only, which is rooted at u_X. □

Lemma 2.3.3. *For any node $u \in V(F)$, and any set of nodes $X \subseteq V(F)$, the following equation holds:*

$$d_X(F, u) = d(F_X, v_X),$$

where v_X is the node in F_X that corresponds to $v = anc_X(F, u)$.

Proof. Because $d_X(F, u) = d_X(F, anc_X(F, u))$ for any node $u \in V(F)$, we need only prove the lemma for $u \in X$. In this case, v is equal to u. Observe that deleting a node not in X does not change the X-depth of any other node. By induction on the number of the deletion operations, we can show that, for any node $u \in X$, $d_X(F, u)$ is equal to the depth of u_X in F_X. □

Lemma 2.3.4. *For any node $u \in V(F)$, and any two sets of nodes $X \subseteq Y \subseteq V(F)$, $anc_X(F, u)$ corresponds to node $anc_{X_Y}(F_Y, v_Y)$ in F_Y, where X_Y is the set of nodes in*

F_Y that correspond to the nodes in X, and v_Y is the node in F_Y that corresponds to $v = anc_Y(F, u)$.

Proof. Let $w = anc_X(F, u)$ and $z_Y = anc_{X_Y}(F_Y, v_Y)$. Our lemma clearly holds if $w =$ `dummy`. Otherwise, w_Y must be an ancestor of v_Y in F_Y, and w_X must be an ancestor of z_X in F_X. By the definition of $anc_X(F, u)$, the depth of z_X in F_X cannot be larger than w_X. Hence, z_X is equal to w_X and z is equal to w. □

Lemma 2.3.5. *A sequence of nodes is corresponding to another if they have the same length, and the nodes at the same position correspond to the same node in F. For any set of nodes $X \subseteq V(F)$, the preorder traversal sequence of F_X is corresponding to the sequence of nodes obtained by removing the nodes that are not in X from the preorder traversal sequence of F.*

Proof. Since F_X is obtained by deleting nodes from F, by applying Proposition 2.2.3(b) multiple times, we can show that our lemma holds. □

2.4 Applying Tree Extraction to Path Queries

Now let us apply the lemmas described in Section 2.3 to the path queries defined in Section 2.1. Let T be the given tree on n nodes, each having a weight drawn from $[1..\sigma]$. For any integral range $[a..b] \subseteq [1..\sigma]$, we define $R_{a,b}$ to be the set of nodes in T that have a weight in $[a..b]$. We denote by $F_{a,b}$ the $R_{a,b}$-extraction of T, which is a forest containing exactly $|R_{a,b}|$ nodes. Note that the nodes in $F_{a,b}$ are not weighted, though they may correspond to weighted nodes in T. Thus, $F_{1,\sigma}$ has the same structure as T, but the nodes in $F_{1,\sigma}$ are not weighted. An example on constructing $F_{1,1}$ and $F_{3,4}$ for an weighted ordinal tree with $n = 10$ and $\sigma = 4$ is illustrated in Figure 2.2.

For any ordinal forest F_X that is extracted from T, and any node u_X in F_X, we define $d_{a,b}(F_X, u_X)$, the $[a..b]$-depth of u_X in F_X, to be the number of ancestors of u_X that correspond to a node in $R_{a,b}$. Also, we define $anc_{a,b}(F_X, u_X)$, the $[a..b]$-ancestor of u_X in F_X, to be the nearest ancestor of u_X that corresponds to a node in $R_{a,b}$. If no such an ancestor exists, then $anc_{a,b}(F_X, u_X)$ is defined to be `dummy`. We further generalize these definitions to T. That is, $d_{a,b}(T, u)$ is equivalent to $d_{a,b}(F_{1,\sigma}, u_{R_{1,\sigma}})$, $anc_{a,b}(T, u)$ is

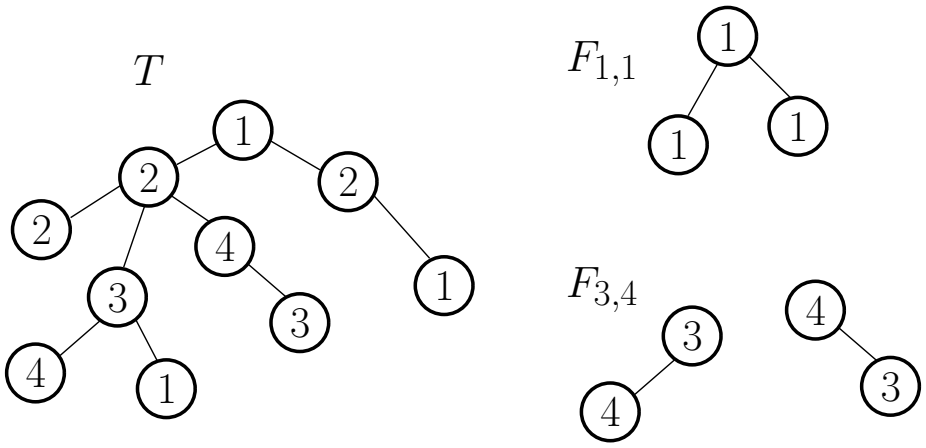

Figure 2.2: An example with $n = 10$ nodes and $\sigma = 4$. The integer on each node of $F_{1,1}$ and $F_{3,4}$ is the weight of its corresponding node in T.

equivalent to the node in T that corresponds to $anc_{a,b}(F_{1,\sigma}, u_{R_{1,\sigma}})$, where $u_{R_{1,\sigma}}$ is the node in $F_{1,\sigma}$ that corresponds to u.

For any nodes u and 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 in T and its ancestor w, we define $A_{u,w}$ to be the set of nodes on the path from u to w, excluding the top node w. Thus, $A_{u,u}$ is a valid but empty set. It is clear that, for any nodes u and v in T, $P_{u,v}$ is the disjoint union of $A_{u,w}$, $A_{v,w}$ and $\{w\}$, where w is the lowest common ancestor (LCA) [2] of u and v.

Consider how to compute the intersection of $R_{a,b}$ and $P_{u,v}$. Providing that w is the lowest common ancestor of u and v, we have

$$\begin{aligned} R_{a,b} \cap P_{u,v} &= R_{a,b} \cap (A_{u,w} \cup A_{v,w} \cup \{w\}) \\ &= (R_{a,b} \cap A_{u,w}) \cup (R_{a,b} \cap A_{v,w}) \cup (R_{a,b} \cap \{w\}); \end{aligned} \tag{2.1}$$

and its cardinality is

$$\begin{aligned} |R_{a,b} \cap P_{u,v}| &= |(R_{a,b} \cap A_{u,w}) \cup (R_{a,b} \cap A_{v,w}) \cup (R_{a,b} \cap \{w\})| \\ &= |R_{a,b} \cap A_{u,w}| + |R_{a,b} \cap A_{v,w}| + |R_{a,b} \cap \{w\}| \\ &= d_{a,b}(T,u) - d_{a,b}(T,w) + d_{a,b}(T,v) - d_{a,b}(T,w) + \mathbf{1}_{R_{a,b}}(w), \end{aligned} \tag{2.2}$$

where $\mathbf{1}_{R_{a,b}}(w)$ is equal to 1 if $w \in R_{a,b}$, or equal to 0 if not. In order to compute the cardinality efficiently, we need a fast way to compute $d_{a,b}(T,u)$'s. A naive solution is to

store the value of $d_{1,b}(T,u)$ for any $1 \leq b \leq \sigma$ and any node u in T. This method requires $O(\sigma)$ words for each node in T so that the overall space cost would be $O(n\sigma)$. To save space, we apply the results in Lemmas 2.3.3 to 2.3.5 with ranges of weights. The following lemmas are obtained by replacing X and Y with $R_{a,b}$ and $R_{a',b'}$ in Lemmas 2.3.3 to 2.3.5.

Corollary 2.4.1. *For any node u in T and its arbitrary ancestor w, and any range $[a..b] \subseteq [1..\sigma]$, we have*

$$d_{a,b}(T,u) - d_{a,b}(T,w) = d(F_{a,b}, x_{R_{a,b}}) - d(F_{a,b}, z_{R_{a,b}}),$$

where $x_{R_{a,b}}$ and $z_{R_{a,b}}$ are nodes in $F_{a,b}$ that correspond to $x = anc_{a,b}(T,u)$ and $z = anc_{a,b}(T,w)$, respectively.

Corollary 2.4.2. *For any node u in T, and any two nested ranges $[a..b] \subseteq [a'..b'] \subseteq [1..\sigma]$, $anc_{a,b}(T,u)$ corresponds to node $anc_{a,b}(F_{a',b'}, v_{R_{a',b'}})$ in $F_{a',b'}$, where $v_{R_{a',b'}}$ is the node in $F_{a',b'}$ that corresponds to $v = anc_{a',b'}(T,u)$.*

Corollary 2.4.3. *For any range $[a..b] \subseteq [1..\sigma]$, the preorder traversal sequence of $F_{a,b}$ is corresponding to the sequence of nodes obtained by removing the nodes that are not in $R_{a,b}$ from the preorder traversal sequence of T.*

Chapter 3

Data Structures under the Pointer Machine Model

In this chapter, we present our data structure under the pointer machine model, for the path queries defined in Section 2.1. The data structure is based on the technique of tree extraction presented in Chapter 2. It requires $O(n \lg \sigma)$ 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 occ is the size of output.

3.1 Basic Structures

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 that differ by at most 1 in length. Formally, providing that $a < b$, the range $[a..b]$ will be split evenly into child ranges $[a_1..b_1]$ and $[a_2..b_2]$, where $a_1 = a$, $b_1 = \lfloor (a+b)/2 \rfloor$, $a_2 = b_1 + 1$ and $b_2 = b$. This procedure stops when $[1..\sigma]$ has been split into σ leaf ranges of length 1, each of which corresponds to a single value in $[1..\sigma]$. These leaf ranges may not occur at the same level of the range tree, as shown in the example illustrated in Figure 3.1. Several important properties of range trees are summarized in Lemma 3.1.1, which can be found in earlier work on the topic [6, 28, 27].

Lemma 3.1.1 ([6, 28, 27]). *A range tree on* $[1..\sigma]$ *has exactly* $\lceil \lg \sigma \rceil + 1$ *levels, and contains exactly* $2\sigma - 1$ *ranges. At each level, each value in* $[1..\sigma]$ *is included in at most one range. In addition, any range* $[p..q] \subseteq [1..\sigma]$ *can be represented as the union of at most* $2\lceil \lg \sigma \rceil + 1$ *disjoint ranges in the range tree.*

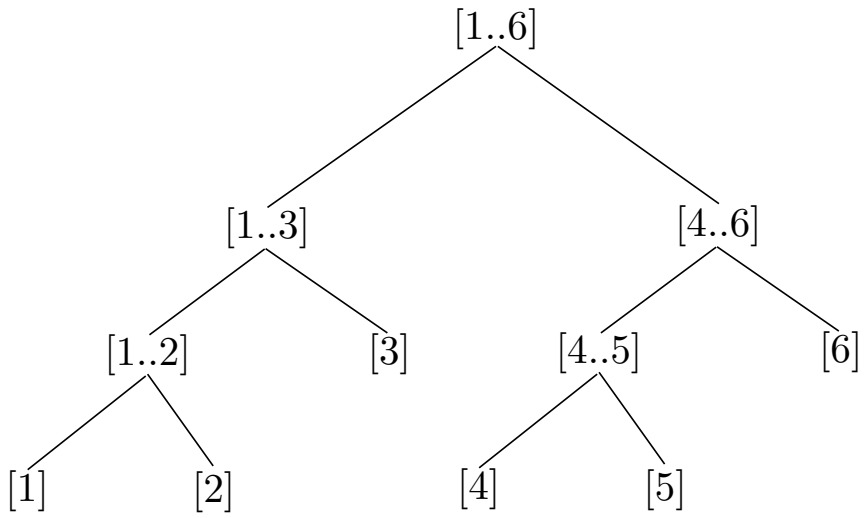

Figure 3.1: A conceptual range tree on [1..6].

For each range $[a..b]$ in the range tree, we construct and store $F_{a,b}$ explicitly. For the sake of convenience, we explicitly add a dummy root to each $F_{a,b}$, and denote by $T_{a,b}$ the new ordinal tree rooted at the dummy node. The dummy root is omitted for the preorder traversal sequence of $T_{a,b}$ and the depths of the nodes in $T_{a,b}$, so $F_{a,b}$ and $T_{a,b}$ can be used interchangeably for the depth and ancestor operators, and the corollaries listed in Section 2.4 still hold for $T_{a,b}$'s. Figure 3.2 gives an example on constructing all the $T_{a,b}$'s for the ordinal tree T in Figure 2.2.

On each node in $T_{a,b}$ we store its depth, and a pointer linking to the corresponding node in T. For each non-leaf range $[a..b]$, let $[a_1..b_1]$ and $[a_2..b_2]$ be the child ranges of $[a..b]$. For each node x in $T_{a,b}$, we pre-compute and store the value of $d(T_{a,b}, x)$, and the pointer to the node in T_{a_i,b_i} that corresponds to $anc_{a_i,b_i}(T_{a,b}, x)$ for $i = 1, 2$, which is denoted by $pointer_{a,b}(x, i)$ for simplicity.

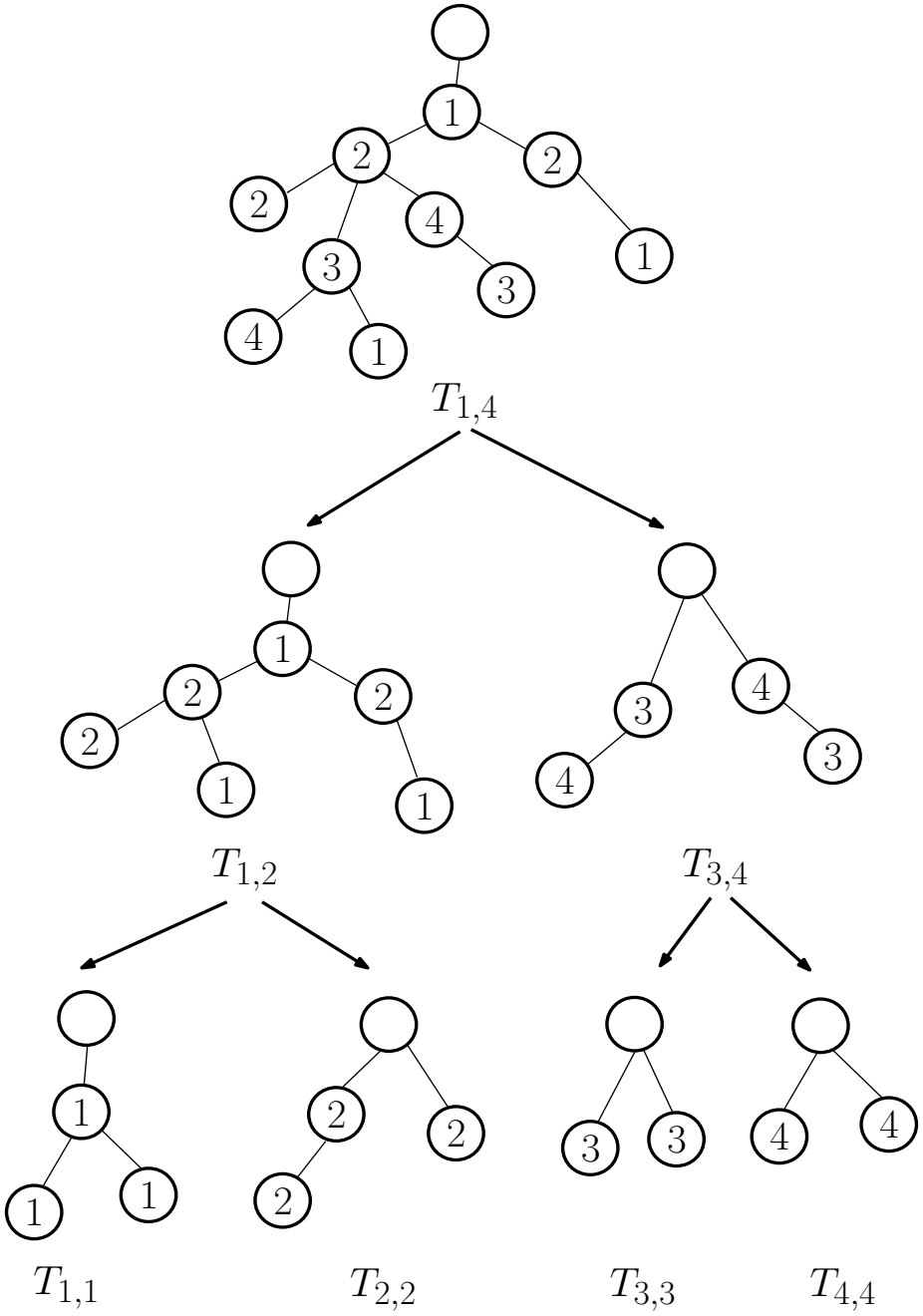

Figure 3.2: An example with $n = 10$ nodes and $\sigma = 4$. The integer on each non-dummy node is the weight of its corresponding node in T.

3.2 Answering Queries

We now describe how to support the path queries defined in Section 2.1 using the above data structure. In the following proofs, we make use of the notation defined in Section 2.4.

Lemma 3.2.1. *The data structure described in Section 3.1 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 $[p..q]$ be the query range. In the path counting and reporting problems, our objective is to compute $R_{p,q} \cap P_{u,v}$ and its cardinality. We first consider how to compute the cardinality for path counting queries. By Lemma 3.1.1, each query range $[p..q] \subseteq [1..\sigma]$ can be represented as the union of m disjoint ranges in the range tree, say $[a_1..b_1], \cdots, [a_m, b_m]$, where $m = O(\lg \sigma)$. Because $R_{p,q} \cap P_{u,v} = \bigcup_{1 \le i \le m} (R_{a_i,b_i} \cap P_{u,v})$, we need only compute the cardinality of $|R_{a_i,b_i} \cap P_{u,v}|$ efficiently, for $1 \le i \le m$.

The algorithm is shown in Algorithm 1. Providing that $[a..b]$ is a range in the conceptual range tree, x, y, and z are nodes in $T_{a,b}$ that correspond to $anc_{a,b}(T, u)$, $anc_{a,b}(T, v)$ and $anc_{a,b}(T, w)$, the procedure SEARCH$([a..b], x, y, z, [p..q])$ returns the cardinality of $P_{u,v} \cap R_{a,b} \cap R_{p,q}$, and reports the nodes in the intersection if the given query is a path reporting query. To compute $P_{u,v} \cap R_{p,q}$, we need only call SEARCH$([1..\sigma], u, v, w, [p..q])$.

Our algorithm accesses the range tree from top to bottom, ending at the ranges completely included in $[p..q]$. Line 8 computes $|R_{a,b} \cap P_{u,v}|$ in $O(1)$ time when $[a..b] \subseteq [p..q]$. In line 12, the algorithm iteratively computes the nearest ancestors of x, y, z for child ranges. Finally, in line 16, the algorithm recurses on child ranges that intersect with $[p..q]$.

For path reporting queries, as shown in lines 4 to 7, 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 the node in T that corresponds to it by following the pointer saved on this node. Finally, we report node w if its weight is in this range.

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 [5]). We thus need only consider our range tree. For path counting queries, Algorithm 1 accesses $O(\lg \sigma)$ ranges, and it spends constant time on each range. For path reporting queries, Algorithm 1 uses $O(1)$ additional time to report each occurrence. Hence, the query time of path

counting queries is $O(\lg \sigma)$, and the query time of path reporting queries is $O(\lg \sigma + occ)$, where occ is the output size. □

Algorithm 1 The algorithm for path counting and reporting queries.

```
1: procedure SEARCH([a..b], x, y, z, [p..q])
2:     ▷ x, y and z correspond to anc_{a,b}(T, u), anc_{a,b}(T, v) and anc_{a,b}(T, w), respectively.
3:     if [a..b] ⊆ [p..q] then
4:         if the given query is a path reporting query then
5:             report all nodes on the path from x to y except z;
6:             report node w if its weight is in [a..b];
7:         end if
8:         return d(T_{a,b}, x) + d(T_{a,b}, y) − 2d(T_{a,b}, z) + 1_{R_{a,b}}(w);
9:                                   ▷ |R_{a,b} ∩ P_{u,v}|, by Corollary 2.4.1 and Equation 2.2.
10:    end if
11:    Let [a_1..b_1] and [a_2..b_2] be the child ranges of [a..b];
12:    δ_i ← pointer_{a,b}(δ, i) for δ = {x, y, z} and i = 1, 2;        ▷ By Corollary 2.4.2.
13:    count ← 0;
14:    for i ← 1, 2 do
15:        if [a_i..b_i] ∩ [p..q] ≠ φ then
16:            count ← count+ SEARCH([a_i..b_i], x_i, y_i, z_i, [p..q]);
17:        end if
18:    end for
19:    return count;
20: end procedure
```

Lemma 3.2.2. *The data structure in this section supports path median and selection queries in $O(\lg \sigma)$ time.*

Proof. It suffices to consider path selection queries only. Let u and v be the nodes given in the query, and let k be the rank of the weight to select. Our algorithm for the path median and selection problem is shown in Algorithm 2. Providing that $[a..b]$ is a range in the conceptual range tree, x, y, and z are nodes in $T_{a,b}$ that correspond to $anc_{a,b}(T, u)$, $anc_{a,b}(T, v)$ and $anc_{a,b}(T, w)$, the procedure SELECT$([a..b], x, y, z, s)$ returns the s-th smallest weight among the weights of the nodes in $R_{a,b} \cap P_{u,v}$. To compute the given query, we

only need call SELECT$([1..\sigma], u, v, w, k)$.

Now let us analyze the procedure SELECT. If $a = b$, the weight to return must be a. Otherwise, let $[a_1..b_1]$ and $[a_2..b_2]$ be the child ranges of $[a..b]$, where $b_1 < a_2$. The algorithm computes $count = |R_{a_1,b_1} \cap P_{u,v}|$ and compares it with s in line 8. If s is not larger than $count$, then the algorithm recurses on $[a_1..b_1]$ in line 11; otherwise the algorithm deducts $count$ from s and recurses on $[a_2..b_2]$ in line 13. Algorithm 2 traverses the range tree from top to bottom, visiting at most $O(\lg \sigma)$ ranges, and finally ending at some range of length 1. It also spends only constant time on each visited range. This algorithm thus takes $O(\lg \sigma)$ time to answer a path median or selection query. □

Algorithm 2 The algorithm for path median and selection queries.

```
1:  procedure SELECT([a..b], x, y, z, s)
2:      ▷ x, y and z correspond to anc_{a,b}(T, u), anc_{a,b}(T, v) and anc_{a,b}(T, w), respectively.
3:      if a = b then
4:          return a;
5:      end if
6:      Let [a_1..b_1] and [a_2..b_2] be the child ranges of [a..b], where b_1 < a_2;
7:      δ_i ← pointer_{a,b}(δ, i) for δ = {x, y, z} and i = 1, 2;          ▷ By Corollary 2.4.2.
8:      count ← d(T_{a_1,b_1}, x_1) + d(T_{a_1,b_1}, y_1) − 2d(T_{a_1,b_1}, z_1) + 1_{R_{a_1,b_1}}(w);
9:                                  ▷ |R_{a_1,b_1} ∩ P_{u,v}|, by Corollary 2.4.1 and Equation 2.2.
10:     if s ≤ count then
11:         return SELECT([a_1..b_1], x_1, y_1, z_1, s);
12:     else
13:         return SELECT([a_2..b_2], x_2, y_2, z_2, s − count);
14:     end if
15: end procedure
```

With Lemmas 3.2.1 and 3.2.2, we can present our result on supporting path queries under the pointer machine model.

Theorem 3.2.3. *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 3.2.1 and Lemma 3.2.2. It suffices to analyze the space cost of our data structure. For a range $[a..b]$ in the range tree, our data structure uses $O(|R_{a,b}|)$ words of space to store $T_{a,b}$, depths of nodes, and the pointers to T and the child ranges of $[a..b]$. 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 $O(n \lg \sigma)$. □

Chapter 4

Data Structures under the Word RAM Model

In this chapter we show how to reduce the space cost of the data structure presented in Chapter 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 $O(n \lg \lg \sigma)$ words of space.

4.1 Linear Space, but Slower Reporting

Our starting point for space optimization is the succinct representation of labeled ordinal trees. As shown in Lemma 4.1.1, 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 a constant, and supports a set of basic operations in constant time. We list in Lemma 4.1.1 only a small subset of operations provided by the succinct representation of labeled ordinal trees, which are sufficient for our data structures.

Lemma 4.1.1 ([19]). *Let S be an ordinal tree on n nodes, each having a label drawn 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 operations in constant time. We assume that node x is contained in S, and $\alpha \in \Sigma$.*

- PRE-RANK(S, x): *Return the number of nodes that precede x in preorder;*
- PRE-RANK$_\alpha(S, x)$: *Return the number of nodes that are labeled with α and precede x in preorder;*
- PRE-SELECT(S, i): *Return the ith node in preorder;*
- PRE-SELECT$_\alpha(S, i)$: *Return the ith node in preorder that is labeled with α;*
- DEPTH(S, x): *Return the depth of x;*
- ANCESTOR$_\alpha(S, x)$: *Return the lowest ancestor of node x that is labeled with α.*

We now present our linear space data structure.

Theorem 4.1.2. *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 3.1. Unlike the previous data structure, we store only the succinct representation of $T_{a,b}$ for each range $[a..b]$ in the range tree. We assign a label to each node in $T_{a,b}$ if range $[a..b]$ is not a leaf range. Let $[a_1..b_1]$ and $[a_2..b_2]$ be child ranges of $[a..b]$, where $b_1 < a_2$. We assign label 0 to the dummy root in $T_{a,b}$. For each non-root node x in $T_{a,b}$, we assign it label i if x corresponds to a node in R_{a_i,b_i} for $i = 1$ or 2. By Lemma 4.1.1, the succinct representation for range $[a..b]$ occupies $O(|R_{a,b}|)$ 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 preorder. Let $[a..b]$ be a non-leaf range, and let $[a_1..b_1]$ and $[a_2..b_2]$ be the child ranges of $[a..b]$, where $b_1 < a_2$. Suppose node x is in $T_{a,b}$, and node x_i is the node in T_{a_i,b_i} that corresponds to x for $i = 1$ or 2. We show that, once the preorder rank of x in $T_{a,b}$ is known, the preorder of x_i in T_{a_i,b_i} can be computed in constant time, and vice versa. By the construction of our linear space data structure, T_{a_i,b_i} contains the nodes that correspond to the nodes in $T_{a,b}$ that have a label i. By Lemma 2.3.5, these

nodes have the same relative positions in the preorder traversal sequences of T_{a_i,b_i} and $T_{a,b}$. We thus have that

$$\begin{aligned} \textsc{Pre-Rank}_i(T_{a,b}, x) &= \textsc{Pre-Rank}(T_{a_i,b_i}, x_i), \\ x_i &= \textsc{Pre-Select}(T_{a_i,b_i}, \textsc{Pre-Rank}_i(T_{a,b}, x)), \quad (4.1) \\ x &= \textsc{Pre-Select}_i(T_{a,b}, \textsc{Pre-Rank}(T_{a_i,b_i}, x_i)). \quad (4.2) \end{aligned}$$

Applying this formula, it takes constant time to convert the preorder rank of two corresponding nodes between two adjacent levels in the range tree.

Since $\textsc{Depth}$ is provided, we need only consider how to compute $pointer_{a,b}(\delta, i)$ for child ranges (Line 12 in Algorithm 1 and line 7 in Algorithm 2). For $\delta = \{x, y, z\}$ and $i = 1, 2$, we can compute the node in $T_{a,b}$ that corresponds to $pointer_{a,b}(\delta, i)$ by $\textsc{Ancestor}_i(T_{a,b}, \delta)$, and convert it to the corresponding node in T_{a_i,b_i} using Equation 4.1. If δ has no ancestor with label i in $T_{a,b}$, then $pointer_{a,b}(\delta, i)$ would be the dummy root of T_{a_i,b_i}.

It is more complicated to deal with path reporting queries. Unlike the data structure described in Section 3, given a node x in some $T_{a,b}$ that need be reported, we cannot directly find its corresponding node in T by following an appropriate pointer, as we cannot afford to store these pointers. Instead, we compute the preorder rank of the node in the tree constructed for the parent range of $[a..b]$ that corresponds to x using Equation 4.2, 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 [11] to speed up this process, which requires the following space-efficient representation of arrays.

Lemma 4.2.1 (Lemma 2.3 in [11]). *An array $A[1..n]$ of elements from an alphabet Σ can be represented in $O(n \lg |\Sigma|)$ bits to support query* Partial-Rank*(i) in constant time, which returns the number of elements in $A[1..i]$ that equal to $A[i]$.*

Now we give the proof for Theorem 4.2.2.

Theorem 4.2.2. *Under the word RAM model with word size $w = \Omega(\lg n)$, a tree on n weighted nodes can be represented in $O(n \lg \lg \sigma)$ 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. Without loss of generality, we assume that σ is equal to a power of 2. In this case, the range tree we constructed is a perfect binary tree that has exactly $\lg \sigma + 1$ levels. We maintain a second index for the nodes in T. For each node x in T, we index x by its weight c and its preorder rank in $T_{c,c}$. Suppose the algorithm decides to report x when accessing range $[a..b]$. We need only find the leaf range corresponding to the weight and the node in the tree constructed for this leaf range that corresponds to x.

Consider the ranges at levels l and $l + \Delta$, where $1 \leq l < l + \Delta \leq \lg \sigma + 1$. A range $[a..b]$ at level l has exactly 2^Δ descendent ranges at level $l + \Delta$. Let them be $[a_1..b_1], [a_2..b_2], \cdots, [a_{2^\Delta}..b_{2^\Delta}]$. We have that, for $i = 1, 2, 3, \cdots, 2^\Delta$,

$$\begin{aligned} a_i &= a + (i-1) \cdot \frac{b-a+1}{2^\Delta}, \\ b_i &= a + i \cdot \frac{b-a+1}{2^\Delta} - 1. \end{aligned}$$

For each range $[a..b]$ at level l, we define $S_{a,b,\Delta}[1..|R_{a,b}|]$ to be a sequence of integers in which $S_{a,b,\Delta}[j] = i$ if and only if the j-th node in the preorder traversal sequence of $T_{a,b}$ corresponds to a node in T_{a_i,b_i}.

Now we show how to "jump" from level l to level $l + \Delta$ using the sequence $S_{a,b,\Delta}$. Let x be a node in $T_{a,b}$. We first access $i = S_{a,b,\Delta}[j]$ for $j =$ Pre-Rank$(T_{a,b}, x)$, such that we know x corresponds node x_i in T_{a_i,b_i}. By Corollary 2.4.3, we can further compute the preorder rank of x_i in T_{a_i,b_i} by the equation

$$\text{Pre-Rank}(T_{a_i,b_i}, x_i) = \text{Rank}_i(S_{a,b,\Delta}, j) = \text{Partial-Rank}(S_{a,b,\Delta}, j).$$

For positive integer l, we define $f(l)$ to be the position in the binary representation of l such that the $(f(l)+1)$-th least significant bit is the rightmost 1-bit. For example, we have

$f(3) = 0$ and $f(8) = 3$. For level $1 \leq l \leq \lg \sigma$, we construct $S_{a,b,\Delta}$ with $\Delta = 2^{f(\lg \sigma + 1 - l)}$ for each range $[a..b]$ at this level. Then we concatenate the sequences for all the ranges at level l in increasing order of their start positions, and maintain the concatenated sequence using the space-efficient representation presented in Lemma 4.2.1. It is easy to see that the concatenated sequence has n integers, each being drawn from $[1..2^{\Delta}]$. Thus, this sequence occupies $O(n\Delta)$ bits of space. The space cost of the concatenated sequences for all levels is at most $\frac{1}{w} \sum_{1 \leq l \leq \lg \sigma} O(n \cdot 2^{f(\lg \sigma + 1 - l)}) = O(n \lg \lg \sigma)$ words.

With the auxiliary data structures described above, for any node x in $T_{a,b}$ that is decided to report, we can reach the node at a leaf range that corresponds to x by jumping. Since each jump increases $f(\lg \sigma + 1 - l)$ by 1, and each jump takes constant time only, this process uses $O(\lg \lg \sigma)$ time for each node to report. $\square$

Chapter 5

Conclusions and Open Problems

We have presented tree extraction, a creative technique for maintaining labeled or weighted trees. This technique supports data structures that recursively split the set of the weights of nodes, while previous divide-and-conquer approaches on trees [14, 26] only partition the structure of the input tree. Following this technique, the path queries with respect to the weights of nodes are supported efficiently.

We have obtained new and improved upper bounds for path queries. Under the pointer machine model, our data structures require $O(n \lg \sigma)$ space, supporting path median, selection and counting queries in $O(\lg \sigma)$ time, and path reporting queries in $O(\lg \sigma + occ)$ time, where occ is the size of output. Under the word RAM model, the space cost of the data structures for path median, selection and counting queries can be reduced to $O(n)$, preserving the same query time. Also, we obtain two more time-space tradeoffs for path reporting queries: The linear space data structure answers path reporting queries in $O(\lg \sigma + occ \lg \sigma)$ time, and the data structure using $O(n \lg \lg \sigma)$ words of space achieves $O(\lg \sigma + occ \lg \lg \sigma)$ query time.

Any data structure for path queries can support the corresponding range queries with the same time and space cost, thus the lower bounds for range queries [31, 32, 33, 25] also apply to the corresponding path queries. In fact, there are still gaps between the data structures obtained in this thesis and the best known upper bounds for the range queries. Let us focus on the case in which σ is close to n. Under the word RAM model, our linear space data structures for path median, selection and counting queries require $O(\lg n)$ time, while the optimal data structures for range median and selection queries [9, 10]

and the ones for two-dimensional orthogonal range counting queries [24, 8, 12] require $O(\lg n / \lg \lg n)$ time only. In addition, our data structures for path reporting queries require $O(n)$ words of space and $O(\lg n + occ \lg n)$ query time, or $O(n \lg \lg n)$ words of space and $O(\lg n + occ \lg \lg n)$ query time, where occ is the output size. However, the best known results for two-dimensional orthogonal range reporting queries [11], using the same amount of space, achieve $O(\lg^{\epsilon} n + occ \lg^{\epsilon} n)$ and $O(\lg \lg n + occ \lg \lg n)$ query time, respectively. Thus we have the first open problem: Can we improve the query time of the data structures for path queries while preserving the space cost?

Let us examine the space cost of the data structures for range queries more carefully. Brodal et al.'s [9, 10] data structures for range median and selection queries and Bose et al.'s [8] data structures for two-dimensional range counting and reporting queries require $n \lg n + o(n \lg n)$ bits of space only, while our data structures support path queries using $\Omega(n \lg n)$ bits of space, where the coefficient is much larger than one. Thus our second open problem is: Can we design data structures for path queries with $n \lg n + o(n \lg n)$ bits of space and efficient query time?

The technique of tree extraction gives us a new direction to maintain labeled or weighted trees. We believe that the usage of this technique is not limited to the path queries we have considered in this thesis. Thus our final open problem is: Can we apply this technique to other types of queries on labeled or weighted trees?

References

[1] Alfred V. Aho and John E. Hopcroft. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1974.

[2] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. On finding lowest common ancestors in trees. *SIAM J. Comput.*, 5(1):115–132, 1976.

[3] Noga Alon and Baruch Schieber. Optimal preprocessing for answering on-line product queries. Technical report, Tel Aviv University, 1987.

[4] Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. New data structures for orthogonal range searching. In *FOCS*, pages 198–207, 2000.

[5] Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *LATIN*, pages 88–94, 2000.

[6] Jon Louis Bentley. Decomposable searching problems. *Inf. Process. Lett.*, 8(5):244–251, 1979.

[7] Philip Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(1-3):217–239, 2005.

[8] Prosenjit Bose, Meng He, Anil Maheshwari, and Pat Morin. Succinct orthogonal range search structures on a grid with applications to text indexing. In *WADS*, pages 98–109, 2009.

[9] Gerth Stølting Brodal, Beat Gfeller, Allan Grønlund Jørgensen, and Peter Sanders. Towards optimal range medians. *Theor. Comput. Sci.*, 412(24):2588–2601, 2011.

[10] Gerth Stølting Brodal and Allan Grønlund Jørgensen. Data structures for range median queries. In *ISAAC*, pages 822–831, 2009.

[11] Timothy M. Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Symposium on Computational Geometry*, pages 1–10, 2011.

[12] Timothy M. Chan and Mihai Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *SODA*, pages 161–173, 2010.

[13] Bernard Chazelle. Filtering search: A new approach to query-answering. *SIAM J. Comput.*, 15(3):703–724, 1986.

[14] Bernard Chazelle. Computing on a free tree via complexity-preserving mappings. *Algorithmica*, 2:337–361, 1987.

[15] Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17(3):427–462, 1988.

[16] Bernard Chazelle. Lower bounds for orthogonal range searching: I. the reporting case. *J. ACM*, 37(2):200–212, 1990.

[17] Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. Syst. Sci.*, 47(3):424–436, 1993.

[18] Travis Gagie, Simon J. Puglisi, and Andrew Turpin. Range quantile queries: Another virtue of wavelet trees. In *SPIRE*, pages 1–6, 2009.

[19] Richard F. Geary, Rajeev Raman, and Venkatesh Raman. Succinct ordinal trees with level-ancestor queries. *ACM Transactions on Algorithms*, 2(4):510–534, 2006.

[20] Beat Gfeller and Peter Sanders. Towards optimal range medians. In *ICALP (1)*, pages 475–486, 2009.

[21] Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *SODA*, pages 841–850, 2003.

[22] Torben Hagerup. Parallel preprocessing for path queries without concurrent reading. *Inf. Comput.*, 158(1):18–28, 2000.

[23] Meng He, J. Ian Munro, and Gelin Zhou. Path queries in weighted trees. In *ISAAC*, pages 140–149, 2011.

[24] Joseph JáJá, Christian Worm Mortensen, and Qingmin Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *ISAAC*, pages 558–568, 2004.

[25] Allan Grønlund Jørgensen and Kasper Green Larsen. Range selection and median: Tight cell probe lower bounds and adaptive data structures. In *SODA*, pages 805–813, 2011.

[26] Danny Krizanc, Pat Morin, and Michiel H. M. Smid. Range mode and range median queries on lists and trees. *Nord. J. Comput.*, 12(1):1–17, 2005.

[27] D. T. Lee and C. K. Wong. Quintary trees: A file structure for multidimensional database systems. *ACM Trans. Database Syst.*, 5(3):339–353, 1980.

[28] George S. Lueker. A data structure for orthogonal range queries. In *FOCS*, pages 28–34, 1978.

[29] Yakov Nekrich. Orthogonal range searching in linear and almost-linear space. *Comput. Geom.*, 42(4):342–351, 2009.

[30] Mark H. Overmars. Efficient data structures for range searching on a grid. *J. Algorithms*, 9(2):254–275, 1988.

[31] Mihai Pătraşcu. Lower bounds for 2-dimensional range counting. In *STOC*, pages 40–46, 2007.

[32] Mihai Pătraşcu. Unifying the landscape of cell-probe lower bounds. *SIAM J. Comput.*, 40(3):827–847, 2011.

[33] Mihai Patrascu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *STOC*, pages 232–240, 2006.

[34] Holger Petersen and Szymon Grabowski. Range mode and range median queries in constant time and sub-quadratic space. *Inf. Process. Lett.*, 109(4):225–228, 2009.

[35] Robert Endre Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *J. Comput. Syst. Sci.*, 18(2):110–127, 1979.

[36] Andrew Chi-Chih Yao. Should tables be sorted? *J. ACM*, 28(3):615–628, 1981.