Mostly all location problems are based on all nodes of a tree. However, in data communication networks only some nodes are communicating with data repository at a particular instance of time. In this thesis, labeling schemes and algorithms for some location problems on set of active nodes in a tree are presented. Let \( n \) be the total number of nodes in a tree \( T \) and at particular instance only \( K \) nodes be active. This thesis concentrates on solving location problems on set of \( K \) active nodes such as Median, Center and Diameter. We have also covered both cases such as relative and absolute center and relative and absolute median while considering the center and median problems.

Our results are as follows.

- For any tree with \( n \) nodes, there is a labeling scheme with labels of size \( O(\log^2 n) \) bits per node, such that, given labels of any \( K \) active nodes of this tree, an absolute median of these \( K \) nodes can be found in \( O(K \log K) \) time, using only those labels.

- For any tree with \( n \) nodes, there is a labeling scheme with labels of size \( O(\log^2 n) \) bits per node, such that, given labels of any \( K \) active nodes of this tree, a relative median of these \( K \) nodes can be found in \( O(K^2) \) time, using only those labels.

- For any tree with \( n \) nodes, there is a labeling scheme with labels of size \( O(\log^2 n) \) bits per node, such that, given labels of any \( K \) active nodes of this tree, diameter of these \( K \) active nodes can be found in \( O(K) \) time, using only those labels.

- For any tree with \( n \) nodes, there is a labeling scheme with labels of size \( O(\log^2 n) \) bits
per node, such that, given labels of any $K$ active nodes of this tree, relative center of these $K$ active nodes can be found in $O(K)$ time, using only those labels.

- For any tree with $n$ nodes, there is a labeling scheme with labels of size $O(\log^2 n)$ bits per node, such that, given labels of any $K$ active nodes of this tree, actual center of these $K$ active nodes can be found in $O(K)$ time, using only those labels.
LABELING SCHEMES FOR SOME LOCATION PROBLEMS ON TREES

A thesis submitted
to Kent State University in
partial fulfillment of the requirements
for the degree of Master of Science

by

Nitin Bafna

May, 2007
# TABLE OF CONTENTS

LIST OF FIGURES .................................................. v

ACKNOWLEDGEMENTS ................................................ vii

1 **Introduction** .................................................. 1

2 **Background** .................................................. 5
   2.1 Key concepts ............................................... 5
   2.2 LCA Labeling Scheme ....................................... 13
   2.3 Distance Labeling Scheme ................................ 17
   2.4 LCA Labeling Scheme by Stephen Alstrup .............. 21

3 **Median** ..................................................... 25
   3.1 Actual Median ........................................... 25
   3.2 Relative median .......................................... 29

4 **Diameter** .................................................... 37

5 **Center** ..................................................... 43
   5.1 Relative Center: .......................................... 43
   5.2 Actual Center: ........................................... 47

6 **Conclusion and Future Work** ............................. 51
   6.1 Conclusion ................................................ 51
   6.2 Future Work ............................................... 51
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Active nodes in rooted tree</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Example of median in a Tree</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Example for fact 2</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>Weight of subtree containing median is more than other subtree</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>Diameter of tree is no more than two times of the radius</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>Illustration to the proof of fact 3</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>Eccentricity of a vertex</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>Use of eccentricity of a vertex to find diameter</td>
<td>11</td>
</tr>
<tr>
<td>9</td>
<td>Middle vertices of diametral path constitute the center of tree $T$</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>Center of a tree</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>Example of Least Common Ancestor</td>
<td>13</td>
</tr>
<tr>
<td>12</td>
<td>Explanation of Decoder Algorithm</td>
<td>15</td>
</tr>
<tr>
<td>13</td>
<td>Preprocessing for distance labeling scheme</td>
<td>17</td>
</tr>
<tr>
<td>14</td>
<td>Tree $T$ Before Balancing</td>
<td>19</td>
</tr>
<tr>
<td>15</td>
<td>Balanced tree $H$ for tree $T$</td>
<td>19</td>
</tr>
<tr>
<td>16</td>
<td>Division of tree in disjoint heavy paths[1]</td>
<td>22</td>
</tr>
<tr>
<td>17</td>
<td>Proof for lemma 3</td>
<td>26</td>
</tr>
<tr>
<td>18</td>
<td>Example for explaining algorithm Absolute Median, step 1</td>
<td>30</td>
</tr>
<tr>
<td>19</td>
<td>Example for explaining algorithm Absolute Median, step 2</td>
<td>31</td>
</tr>
<tr>
<td>20</td>
<td>Example for explaining algorithm Absolute Median, step 3</td>
<td>32</td>
</tr>
<tr>
<td>21</td>
<td>Example for explaining algorithm Absolute Median, step 4</td>
<td>33</td>
</tr>
<tr>
<td>22</td>
<td>Example for explaining algorithm Absolute Median, step 5</td>
<td>34</td>
</tr>
<tr>
<td>23</td>
<td>Example of finding relative median of active nodes</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>24</td>
<td>Diameter of tree.</td>
<td>38</td>
</tr>
<tr>
<td>25</td>
<td>Diameter of active nodes in a tree.</td>
<td>38</td>
</tr>
<tr>
<td>26</td>
<td>One end of a diametral path is the deepest point in tree.</td>
<td>39</td>
</tr>
<tr>
<td>27</td>
<td>Two cases for node joining in diameter path.</td>
<td>40</td>
</tr>
<tr>
<td>28</td>
<td>Example of finding diameter of active nodes in a tree.</td>
<td>41</td>
</tr>
<tr>
<td>29</td>
<td>Closest point to the absolute center of tree is a relative center.</td>
<td>44</td>
</tr>
<tr>
<td>30</td>
<td>Closest point to the absolute center of tree is a relative center.</td>
<td>46</td>
</tr>
<tr>
<td>31</td>
<td>Actual center of $K$ active nodes in a tree.</td>
<td>49</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

I am very grateful to my advisor Dr. Feoder Dragan, and the committee members, Dr. Yuri Breitbart and Dr. Hasan Peyravi, for their constant encouragement and direction. I have known Dr. Dragan for a long time and would always cherish his gentle guidance and support that have made this research possible.

I am also grateful to the members of the research team of Dr. Dragan in Department of Computer Science for spending their valuable time with me to get important things done for this thesis.

I would also like to thank my colleagues at Brulant Inc., Cleveland, for understanding for the times I had to be away from Brulant work to focus on my research.

I thank my friends and family for their constant help and support during my thesis completion and always. I thank my mother, Jasoda Bafna, my father, Nemichand Bafna, and my brother, Lalit Bafna to be there with me when I needed them. I would also thank my dearest friend Abha Gupta, for providing me her support, affection and guidance during the completion of this thesis.
CHAPTER 1

Introduction

In centralized data system, data is kept in one or limited number of repositories and all other receivers communicate with repositories to send and receive data. Generally these repositories are located at the median or center of the network formed by receivers. But in lot of cases all receivers do not communicate with repositories simultaneously. At particular time only some of receivers are active. There is fair amount of chances that original center or median of graph may not be center or median with respect to active receivers. For performance boost we will certainly like that data is stored at a location which is median or center of active points at that particular time.

For finding center or median we need efficient network representation which concerns the development of methods and structures for cheaply storing useful information about the network and making it readily and conveniently accessible. In tradition centralized approach, network is represented by adjacency information data structure, e.g. adjacency matrix. We can solve lot of location problems using this information. However in this centralized approach this matrix is the whole source of information and identifiers of vertices does not contain much useful information, except pointing towards rows and columns of that matrix. In contrast vertices can be used for storing more useful information in such a way that it will allow us to find adjacency using only local information on vertices. Breuer and Folkman introduced notion of adjacency labeling schemes [3] [2]. Hence having local informative labels for vertices make good network representation. While designing labels we want to construct short labels such that they carry enough information to solve problems. Sufficient adjacency labeling schemes were explored in [7].
Similar motivation led to development of labeling schemes for finding distance information. This notion is called distance labeling scheme, which enables us to find distance between any two vertices efficiently from their labels. This notion was introduced in [9] and studied further in [4][8].

In this thesis, we are concentrating on rooted tree networks and also we have set of some nodes in a tree which are active at particular instance of time. We will be solving location problems on this active set. As shown in figure 1 we have rooted tree and only some nodes are active. Consider there is data repository in one of the nodes in a tree and at particular instance of time only some nodes are active and communicating with repository. Current data center may decide to move repository to convenient location with respect to active nodes for better performance. That data center will have labels of those active nodes and only using these labels it needs to determine the next convenient location for data repository. In this thesis we design labeling schemes and algorithms for finding convenient location as fast as possible by using small labels. We will be mainly concentrating on median, center
and diameter of active nodes in a tree. While considering median and center we are going
to consider both cases which are, Absolute and Relative median/center. When median or
center can be any node in a tree then that median/center is called absolute median/center.
In contrast, Relative median/center can only be one of the active nodes in a tree.

In chapter 2, we will explain some of the concepts, facts and algorithms related to
location problems. We will also explain least common ancestor (LCA) labeling schemes
and distance labeling scheme which we will use to find median, center and diameter of a
tree. In chapter 3, we will consider problem of finding relative and absolute median of a
tree. In chapter 4, we will explain how to use distance labeling scheme to find diameter
of active nodes in a tree efficiently. In Chapter 5, we will construct labeling schemes for
finding relative and absolute center of active nodes in a tree.

Our results are as follows.

- For any tree with $n$ nodes, there is a labeling scheme with labels of size $O(\log^2 n)$
  bits per node, such that, given labels of any $K$ active nodes of this tree, an absolute
  median of these $K$ nodes can be found in $O(K \log K)$ time, using only those labels.

- For any tree with $n$ nodes, there is a labeling scheme with labels of size $O(\log^2 n)$ bits
  per node, such that, given labels of any $K$ active nodes of this tree, a relative median
  of these $K$ nodes can be found in $O(K^2)$ time, using only those labels.

- For any tree with $n$ nodes, there is a labeling scheme with labels of size $O(\log^2 n)$ bits
  per node, such that, given labels of any $K$ active nodes of this tree, diameter of these
  $K$ active nodes can be found in $O(K)$ time, using only those labels.

- For any tree with $n$ nodes, there is a labeling scheme with labels of size $O(\log^2 n)$ bits
  per node, such that, given labels of any $K$ active nodes of this tree, relative center of
  these $K$ active nodes can be found in $O(K)$ time, using only those labels.
• For any tree with $n$ nodes, there is a labeling scheme with labels of size $O(\log^2 n)$ bits per node, such that, given labels of any $K$ active nodes of this tree, actual center, of these $K$ active nodes can be found in $O(K)$ time, using only those labels.
CHAPTER 2

Background

2.1 Key concepts

Weber function: For any vertex $x$ in a tree $T$, Weber function is the sum of weight of every vertex $y$ in tree $T$ multiplied by it’s distance to $x$[6],

$$F(x) = \sum_{y \in V} w(y)d(x,y).$$

Consider tree shown in figure 2. Let number on edges represent weight of edges and number on nodes represent weight of nodes. Therefore weber function of node $a$,

$F(a) = 2(2) + 2(3) + 2(1) + 2(3) + 2(2) + 2(5) + 2(4) = 40.$

$F(b) = 2(3) + 2(4) + 2(1) + 2(2) + 2(1) + 2(4) + 2(3) = 36.$

Median: Median $M$ of tree $T$ is defined as vertices with minimum Weber function value.

$M = \{x \in V : F(x) \text{ is minimal} \}.$

Let us consider figure 3. $x$ and $y$ are adjacent vertices and we get two subtrees $T_x$ and $T_y$ by deviding tree $T$ over edge $xy$. Weber functions for $x$ and $y$,

Figure 2: Example of median in a Tree.
\[ F(x) = \sum_{v \in T_x} w(v)d_{T_x}(v, x) + \sum_{v \in T_y} w(v)d_{T_y}(v, x). \]

Therefore,
\[ F(x) = \sum_{v \in T_x} w(v)d_{T_x}(v, x) + \sum_{v \in T_y} w(v)[d_{T_y}(v, y) + 1], \]
\[ F(x) = \sum_{v \in T_x} w(v)d_{T_x}(v, x) + \sum_{v \in T_y} w(v)d_{T_y}(v, y) + \sum_{v \in T_y} w(v). \]

Similarly,
\[ F(y) = \sum_{v \in T_y} w(v)d_{T_y}(v, y) + \sum_{v \in T_x} w(v)d_{T_x}(v, x) + \sum_{v \in T_x} w(v). \]

Therefore,
\[ F(x) - F(y) = \sum_{v \in T_y} w(v) - \sum_{v \in T_x} w(v), \]
i.e.,
\[ F(x) - F(y) = [w(T_y) - w(T_x)], \]
where \( W = \sum_{v \in V} w(v), W(T_y) = \sum_{v \in T_y} w(v) \) and \( W(T_x) = \sum_{v \in T_x} w(v) \).

**Fact 1:** If \( w(T_y) > 1/2W \) then median belongs to \( T_y \).

**Proof:** As we know total weight of tree \( T \) is
\[ W = w(T_y) + w(T_x). \] From above we have,
\[ F(x) - F(y) = w(T_y) - w(T_x) \] and
\[ F(x) - F(y) = w(T_y) - W + w(T_x). \]
But as \( w(T_y) > 1/2W \), \( F(x) - F(y) > 0 \), therefore median belongs to subtree \( T_y \), as any vertex in \( T_x \) will have larger Weber function than the one of \( y \).
**Fact 2:** If \( w(T_y) = w(T_x) = \frac{1}{2}W \) then \( F(x) = F(y) \), i.e. \( x, y \) both are medians of tree \( T \).

**Proof:** Let us look at figure 3. Assume \( x \) and \( y \) are not medians of a tree. Let \( m \) be the median of tree in \( T_y \) closest to \( y \). Let \( m' \) be the non-median vertex on path between \( m \) and \( y \), adjacent to \( m \) As shown in figure 3 and 4,

\[
F(m') - F(m) > 0,
\]

i.e.,

\[
w(T_{m'}) < w(T_m).
\]
But then,

\[ w(T_x) \leq w(T_m') < w(T_m) \leq w(T_y) \]

and a contradiction arises.

**Goldman’s algorithm to find median:**

1. Find total weight \( W \) of all vertices in tree.

2. Find Leaf node \( v \) in a tree.

3. Let \( u \) be the father of \( v \).

4. If \( w(v) > \frac{1}{2}W \) then output \( v \) as median.
   
   Else delete \( v \) from tree \( T \) and set \( w(u) = w(u) + w(v) \).

5. Goto step 2.

As explained in above algorithm we start with a leaf of tree. In each step, we add leaf’s weight to its father’s weight until father’s weight becomes more than \( \frac{1}{2}W \). As weight of any vertex becomes more than half of total weight of tree it is median of tree. That means the deleted branch of tree from that vertex have weight more than \( \frac{1}{2}W \). Therefore according to facts explained above that vertex is a median of tree. Clearly complexity of this algorithm to find a median is \( O(n) \).

**Diameter:** Diameter of a tree is the maximum distance between two nodes in a tree. In other words diameter is longest distance possible between two nodes of tree. That pair of nodes is called a diametral pair. Formally, the diameter of \( T \) is \( diam(T) = \max\{d(x,y) | x,y \in V\} \), and the radius of \( T \) is \( rad(T) = \min\{\max\{d(v,u) | u \in V\} | v \in V\} \). The center, \( C(T) \), of \( T \) is all vertices with, \( ecc(v) = \max\{d(u,v) | u \in V\} = rad(T) \), i.e., with smallest eccentricity.
Fact 3: \(2\text{rad}(T) \geq \text{diam}(T) \geq 2\text{rad}(T) - 1.\)

Proof: Let’s first prove that \(\text{diam}(T) \leq 2\text{rad}(T).\) As shown in figure 5, let \(c\) be the center of tree \(T\) and \(x, y\) be the diametral pair of tree. Therefore, \(d(x, y) \leq d(x, c) + d(y, c).\) As for any vertex \(v\) in tree \(T, d(c, v) \leq \text{rad}(T).\) Therefore, \(\text{diam}(T) = d(x, y) \leq 2\text{rad}(T).\) Now let’s prove \(2\text{rad}(T) - 1 \leq \text{diam}(T).\) Assume that \(\text{diam}(T) \leq 2\text{rad}(T) - 2.\) As shown in figure 6 let \(x, y\) be a diametral pair of tree and \(c\) be any middle vertex on path between \(x\) and \(y.\) Consider any vertex \(z\) of \(T\) which is farthest from \(c.\) As shown in figure there are three paths possible between \(z\) and \(c.\) Assume we can reach \(z\) from \(c\) via path 3. Now, \(d(x, z) = d(x, c) + d(c, z),\) but, \(\text{diam}(T) \leq 2\text{rad}(T) - 2\) and \(d(c, y) \leq \text{rad}(T) - 1.\) Hence,
Figure 7: Eccentricity of a vertex.

\[ d(x, c) + d(c, z) > d(x, c) + d(c, y). \] But this is a contradiction for \( x, y \) being a diametral pair. Therefore our assumption \( \text{diam}(T) \leq 2 \text{rad}(T) - 2 \) is wrong. Similarly other cases lead to contradictions. Therefore \( 2 \text{rad}(T) - 1 \leq \text{diam}(T) \).

**Eccentricity of vertex**: Eccentricity of vertex \( v \) is defined as maximum distance from vertex \( v \) to all other vertices in a tree \( T \).

\[ \text{ecc}(v) = \max_{x \in V} d(v, x). \]

For example, in figure 7, \( \text{ecc}(v) = 6 \) assuming all edges have weights 1. Also you can see in figure 7, \( x \) is a center of \( T \) and \( \text{ece}(x) = 5 = \text{rad}(T) \). Eccentricity of center, \( x \), is the minimum of eccentricity of any vertex in tree \( T \).

**Fact 4**: For any vertex \( x \) of tree and it's farthest vertex \( y \), i.e. \( \text{ece}(x) = d(x, y) \), \( \text{ecc}(y) = \text{diam}(T) \) holds.

**Proof**: As shown in figure 8, let \( u, v \) give the diameter of a tree \( T \) and \( y \) is a farthest point to \( x \). Therefore \( d(x, y) \geq d(x, u) \) and \( d(x, y) \geq d(x, v) \).

Now, \( d(x, c) + d(c, y) \geq d(x, c) + d(c, u) \).

Hence, \( d(c, y) \geq d(c, u) \).

But then, \( d(v, y) = d(v, c) + d(c, y) \geq d(v, c) + d(c, u) = d(v, u) = \text{diam}(T) \).
Therefore $y$ is an end point of diameter and eccentricity of $y$ is diameter of tree $T$.

**Algorithm to find Diameter:**

1. Pick any vertex $x$ from set of vertices from tree $T$.

2. Run $BFS(x)$. Find vertex, $y$, farthest to $x$.

3. Run $BFS(y)$. Find vertex, $z$, farthest to $y$.

4. Output $y$, $z$ as a diametral pair of $T$ and $diam(T) = d(y, z)$.

**Fact 5:** Middle vertices of diametral path constitute the center of tree $T$. 
Proof: As shown in figure 9, let $x$ and $y$ be the end points of diametral path of tree $T$ and $c$ be a central vertex on diametral path. Consider point $u$ which is farthest to point $c$ in tree.

Therefore, $d(c, u) \geq d(x, c)$ and $d(c, u) \geq d(c, y)$,

which implies $u$ should be end point of diametral path. As $d(c, u) \geq \text{rad}(T)$ and $d(x, y) = \text{diam}(T) \leq 2 \text{rad}(T)$, we get $d(c, u) = \text{rad}(T)$. Hence we have proved middle vertices of diametral path are centers of a tree.

Let us prove that there are no other central vertices in the tree. As shown in figure 10, let $x$ and $y$ be the end points of a diametral path of tree $T$ and $c$ be a central vertex on diametral path. Assume that point $c_T$ is also center of tree and $d(c, c_T) = r_1$ and radius is $r$. We know that $d(c_T, x) \leq r$ and $d(c_T, y) \leq r$. If $d(x, y) = \text{diam}(T) = 2r$, then $c_T$ must be in middle of path between $x$ and $y$, i.e. $c = c_T$. If $d(x, y) = \text{diam}(T) = 2r - 1$, then $d(c, y) = r$ and $d(c, x) = r - 1$. If $d(c, y) = r - 1$, then again $c = c_T$.

So, we may assume that $d(c_T, x) = d(c_T, y) = r$. But then, since $d(c, y) = r$ too, there is a cycle in tree $T$, which is impossible.
2.2 LCA Labeling Scheme

In this section we are going to explain in detail labeling scheme for finding Least Common Ancestor (LCA) of two vertices in a rooted tree designed by Dr. Peleg [10]. For any two vertices \(v, w\) in a tree, \(z = \text{LCA}(v, w)\) is the least common ancestor of \(v, w\) in a tree. Formally LCA function maps the vertex pair \((v, w)\) into identifier \(I(z)\), where \(I(z)\) is unique identifier of vertex \(z\) in tree [10].

For example, in figure 11, \(\text{LCA}(18, 12) = 16\) and \(\text{LCA}(10, 3) = 11\). 17 is also ancestor of 10 and 3 but 11 is the least common ancestor of pair \((10, 3)\).

**Goal:** Design an efficient labeling scheme that encodes least common ancestor information and decoder, a polynomial time algorithm for inferring the least common ancestor of the two nodes only form their labels.

Now we will see labeling scheme designed by Dr. Peleg [10]. Labels are constructed as follows. First we assign its identifier with interval labels of that vertex. Interval labeling scheme is designed by N. Santoro and R. Khatib [11]. These scheme consists of 2 steps,
first to assign all vertices of tree depth first search numbers starting at root and then label each vertex \( u \in T \) by interval \( \text{Int}(u) = [\text{DFS}(u), \text{DFS}(w)] \), where \( w \) is the last descendent of \( u \) visited by the DFS tour. The resulting labeling scheme has labels of size \( O(\log n) \) bits per node. By assigning interval labels to each vertex they satisfy following property: For every two vertices \( u \) and \( v \) of the tree \( T \), \( \text{Int}(v) \subseteq \text{Int}(u) \) iff \( v \) is descendent of \( u \) in \( T \).

For every vertex \( v \) in the tree, let \( T(v) \) denote the subtree of \( T \) rooted at \( v \). For \( 0 \leq i \leq \text{depth}(v) \), denote \( v \)'s ancestor at level \( i \) of the tree by \( \gamma_i(v) \). In particular, \( \gamma_0(v) = \text{root} \) and \( \gamma_{\text{depth}(v)}(v) = v \). A non root vertex \( v \) with parent \( w \) is called small if its subtree, \( T(v) \), contains at most half the number of vertices contained in its parents' sub-tree, \( T(w) \). Otherwise, \( v \) is large. The root is always small.

For a vertex \( v \) and \( 1 \leq i \leq \text{depth}(v) \), the \( i \)-triple of \( v \) is

\[
Q_i(v) = [(i - 1, \text{Int}((\gamma_{i-1}(v)))), (i, \text{Int}(\gamma_i(v))), (i + 1, \text{Int}(\gamma_{i+1}(v)))].
\]

So, finally in second stage we assign to each vertex \( v \) the label,

\[
L(v) = (\text{Int}(v), \text{Int}(v), \{Q_i(v) | 1 \leq i \leq \text{depth}(v), i \in \text{SAL}(v)\}),
\]

where \( \text{SAL}(v) \) is small ancestor levels of \( v \) which can be defined as,

\[
\text{SAL}(v) = \{i | 1 \leq i \leq \text{depth}(v), \gamma_i(v) \text{ is small}\}. \quad \text{Small ancestors of } v \text{ can be represented as } \text{SA}(v) = \{\gamma_i(v) | i \in \text{SAL}(v)\} [10].
\]

Now we will see how decoder works. Goal of decoder is to find LCA, \( I(z) \), of vertices \( v \) and \( w \) when \( L(v) \) and \( L(w) \) are given, where \( I(z) \) is identifier of their least common ancestor \( z = \text{LCA}(v, w) \).

**Decoder** \( D_{\text{LCA}} \)

1. If \( \text{Int}(w) \subseteq \text{Int}(v) \) then return \( I(v) \). /* \( v \) is ancestor of \( w \) */

2. If \( \text{Int}(v) \subseteq \text{Int}(w) \) then return \( I(w) \). /* \( w \) is ancestor of \( v \) */
3. Extract from $L(v)$ and $L(w)$ the sets $SAL(v)$, $SAL(w)$, $SA(v)$ and $SA(w)$.

4. Let $\alpha$ be the highest level vertex in $SA(v) \cap SA(w)$. Let $K$ be its level, i.e., $\alpha = v^{\gamma_K(v)} = w^{\gamma_K(w)}$. /* $\alpha$ = least common small ancestor of $v$ and $w$ */

5. If $v^{\gamma_{K+1}(v)} \neq w^{\gamma_{K+1}(w)}$ then return $I(\alpha)$.

6. /* $v^{\gamma_{K+1}(v)} = w^{\gamma_{K+1}(w)}$ is also a common (yet large) ancestor of $v$ and $w$ */

   Let $i_v = \min\{i \in SAL(v) \mid i > K\}, i_w = \min\{i \in SAL(w) \mid i > K\}, i_m = \min\{i_v, i_w\}$.

7. Extract $v^{\gamma_{i_{m-1}}(v)}$ from the $i_m$-triple $Q_{i_{m}}(v)$; Return $I(v^{\gamma_{i_{m-1}}(v)})$.

If $v$ is ancestor of $w$ or vice versa, then step 1 or 2 of decoder $D_{LCA}$ will find $LCA(v, w)$ correctly by using Interval labeling property. Dr. Peleg have proved the correctness of this algorithm in [10]. Let us understand algorithm with an example. Consider a tree shown in figure 12. Let’s first find least common ancestor of $L$ and $H$.

$SA(L) = \{L, J, A\}$. 

Figure 12: Explanation of Decoder Algorithm.
Therefore, $SAL(L) = \{5, 4, 0\}$.

$SA(H) = \{H, B, A\}$.

Therefore, $SAL(F) = \{3, 1, 0\}$.

$SA(L) \cup SA(H) = \{A\}$.

Therefore, highest level vertex $\alpha = A$ and it's level $K = 0$,

i.e. $\alpha = \gamma_0(H) = \gamma_0(L) = A$.

Now $\gamma_1(H) \neq \gamma_1(L)$.

Therefore, according to step 5, decoder will return $I(\alpha) = I(A)$. $A$ is least common ancestor of $H$ and $L$.

Let's look at another example from same tree shown in figure 12. We will see now how we can find LCA of nodes $M$ and $P$.

$SA(M) = \{M, J, A\}$.

Therefore, $SAL(L) = \{5, 4, 0\}$.

$SA(P) = \{P, K, A\}$.

Therefore, $SAL(F) = \{5, 4, 0\}$.

$SA(L) \cup SA(F) = \{A\}$.

Therefore, highest level vertex $\alpha = A$ and it's level $K = 0$,

i.e. $\alpha = \gamma_1(M) = \gamma_1(P) = A$

But now in this case $\gamma_2(M) = \gamma_2(P) = C$.

Therefore, step 5 in decoder is not true this time and according to step 6 we are going to find $i_v$ and $i_w$.

$i_v = \min\{5, 4\}$ and $i_w = \min\{5, 4\}$.

Therefore, $i_m = \min\{i_v, i_w\} = 4$.

Therefore, decoder $D_{LCA}$ will return $I(I)$ as

$\gamma_{i_m-1}(v) = \gamma_3(v) = I$.

Therefore, $I$ is least common ancestor of $M$ and $P$. 
This labeling scheme is proved very handy in every algorithm we have designed. We will see how this labeling scheme is effectively used in forthcoming chapters.

**Theorem 1** For a rooted tree \( T \) of \( n \) nodes, we can construct labels of size \( O(\log^2 n) \) bits for each node, such that, given labels of 2 nodes we can find least common ancestor of those two nodes in constant time.

### 2.3 Distance Labeling Scheme

In this section we will explain efficient distance labeling scheme[4] in detail. In this scheme we construct labels of size \( \log^2 n \) bits for nodes, such that given labels of two nodes we can find the distance between them in tree in constant time.

**Goal:** Design short labels for nodes in tree to encode distances and distance decoder, an algorithm for inferring the distance between two nodes only from their labels (in time polynomial in the label length)[4].
Preprocessing: In this step we arbitrarily label each node \( v \) in \( T \) with distinct integers \( I(V) \) from \([1..n]\) as shown in figure 13.

Recursive partitioning and Decomposing: In this step we are going to balance tree. We do balancing because we want to limit the maximum depth of tree to \( \log n \). Balancing of tree helps to limit label size to \( \log^2 n \). Let’s see how we do balancing by recursive partitioning and decomposing. To decompose tree we are going to find tree separator nodes.

Tree Separator: Tree separator is node \( v \) in \( n \)-node tree \( T \), whose removal breaks \( T \) into subtrees of size at most \( n/2 \).

Known fact: Every tree has tree separator that can be found in linear time.

Therefore in this step in tree \( T \) we find tree separator and decompose tree \( T \) in two or more subtrees \( T_1 \cdots T_k \). then continue process for each subtree \( T_1 \cdots T_k \) separately.

Let’s understand this by example shown in figure 13 and figure 14.

Tree \( T \) shown in figure 13 contains 14 nodes. We have arbitrarily labeled each node with distinct integer. In \( T \) node 17 is tree separator. Removing node 17 we can decompose tree \( T \) in to subtrees \( T_1 \) and \( T_2 \). Both \( T_1 \) and \( T_2 \) contain no more than \( n/2 \), i.e. 7, nodes. Similarly in \( T_1 \) node 16 is tree separator. Removing node 16 tree \( T_1 \) decomposes in to three subtrees \( T_{11}, T_{12} \) and \( T_{13} \) containing no more than 3 nodes. In tree \( T_2 \) node 11 is tree separator removal of which decompose tree \( T_2 \) in three subtrees containing no more than 3 nodes. Similarly we decompose each subtree by finding tree separator in the tree. Nodes 9, 5 and 19 are used to further decompose subtrees.

Now we can construct balanced tree as shown in figure 15 by arranging node in order of decompositions. For example as node 17 for first node used for decomposition so it will be root of the balanced tree, \( H \), and there children will be the separator nodes in two subtrees formed by decomposition.
Figure 14: Tree T Before Balancing.

Figure 15: Balanced tree H for tree T.
Labels:

First we will construct LCA labels for nodes as explained in previous section. LCA labels will be $O(\log^2 n)$ bits size and then we can find LCA of 2 nodes in constant time [10]. Therefore for vertex $v$ in $H$, LCA Label will be,

$$v \rightarrow A(v).$$

In distance labeling scheme label of $v$ will be,

$$Label(v) = (A(v), dist_T(v, c_0), dist_T(v, c_1), \ldots, dist_T(v, c_h)).$$

where $c_0, c_1, c_2, \ldots, c_h$ are ancestors of node $v$ in $H$ on height $0, 1, 2, \ldots, h$ respectively.

As $H$ is balanced tree no child subtree will have more than half nodes of it’s parent tree. Therefore hight of balanced tree, $H$, will be maximum $\log n$. Therefore any node in $H$ will not have more than $\log n$ ancestors. Since $\log n$ bits per entry, $Label(v)$ will be of size $O(\log^2 n)$ bits. Let’s look at the decoder algorithm shown in algorithm 1. In first step we

**Input:** Vertices $u$ and $v$ and their $Label(u)$ and $Label(v)$.

**Output:** $dist_T(u, v)$.

Extract $A(u), A(v)$ from $Label(u)$ and $Label(v)$.

Find depth $k$ of $LCA(u, v)$ using $A(u), A(v)$.

Output $dist_T(v, c_k) + dist_T(v, c_k)$ by extracting appropriate entries from labels.

**Algorithm 1:** Distance Decoder

will extract $A(u), A(v)$ from $Label(u)$ and $Label(v)$, which we will use for getting $LCA(u, v)$.

In second step we will find the height of least common ancestor of $u$ and $v$ in tree $H$ shown in figure 15. Once we get height of least common ancestor, distance between $u$ and $v$ is distance of $u$ and $c_k$ in tree $T$ added to distance of $v$ and $c_k$ in tree $T$. These distances can be extracted from $Label(u)$ and $Label(v)$.

Let’s understand decoder algorithm by example. Let $dist_T(A, B)$ represent distance between nodes $A$ and $B$ in tree $T$. Consider each edge represent distance 1 in tree $T$ shown in figure 14. Therefore by refereing $T$ and balanced tree $H$ shown in figure 15, labels of nodes 19 and 5 are
Label(19) = (A(19), 2, 2, 0),
Label(5) = (A(5), 4, 2, 0).

Therefore, LCA(19, 5) = 17 and height of node 17 is $k = 0$.

Therefore, $\text{dist}_T(19, 5) = \text{dist}_T(19, 17) + \text{dist}_T(5, 17) = 2 + 4 = 6$,

where $\text{dist}_T(19, 17)$ and $\text{dist}_T(5, 17)$ can be extracted from labels in constant time. Now we have seen how to find distance of 2 nodes in tree in constant time using $O(\log^2 n)$ bit labels. We will be using these labels in following chapters for finding diameter and center of $K$ active nodes in the tree.

**Theorem 2** [4] For a tree $T$ of $n$ nodes, we can construct labels for each node of size $O(\log^2 n)$ bits, such that, given labels of 2 nodes we can find distance between them in constant time.

2.4 LCA Labeling Scheme by Stephen Alstrup

In this section we are going to see another labeling scheme for finding least common ancestor of two nodes[1]. The special structure of labels used in this scheme enables us to find ancestor at height $k$ of a child. We will first look in detail how it is designed and used for finding least common ancestor of two nodes in this section. Then we will see how to modify it to find center of active nodes in a tree in chapter 5.

**Lemma 1** (Gilbert and Moore [5]). A sequence $< y >_k$ of positive numbers with $n = \Sigma_{i=1}^k y_i$ has an alphabetic code $< b >_k$, where $|b_i| \leq \log n - \log y_i + O(1)$ for all $i$ and $< y >_k$ is a sequence of objects $y_1, y_2, ..., y_k$. An alphabetical sequence of $< y >_k$ is a sequence of binary strings $< b >_k$, $b_i \epsilon \{0, 1\}^*$, where $b_i < b_j$, for all $1 \leq i < j \leq k$.

This lemma states that we can construct an alphabetic sequence satisfying the length bounds in Lemma 1 for an integer sequence $< y >_k$ in $O(k)$ time. That is length of alphabetical code will be less than $\log n$. Detailed proof of the lemma is given in the paper[1].
**Figure 16**: Division of tree in disjoint heavy paths[1].

**Labels**: As shown in figure 16, we divide the tree into disjoint paths. For a tree $T$, let $|T|$ denote the number of nodes of $T$. Let $T_v$ be the subtree rooted by $v$, and $size(v) = |T_v|$. Let $parent(v)$ be the parent of $v$, and $children(v)$ be the set of children of $v$. We classify each node of $T$ as either heavy or light as follows. The root is light. For each internal node $v$, we pick a child $w$ of $v$, where $size(w) = \max\{size(z) | z \in children(v)\}$ and classify $w$ as heavy. We classify each of the remaining children of $v$ as light. We call an edge to a light child a light edge, and an edge to a heavy child a heavy edge. For a node $v$ with a heavy child $w$, let $lsize(v) = size(v) - size(w)$. The nearest ancestor of $v$ which is light (possibly $v$ itself if $v$ is light) is denoted by $apex(v)$. By removing the light edges $T$ is partitioned into paths, which we call heavy paths. A node $w$ belongs to the same heavy path as the nodes of the set, $HP(w) = \{v | v \in T; apex(v) = apex(w)\}$.

Once tree is partitioned we assign heavy label, $hlabel(v)$, to each node $v \in T$ and a light label $llabel(v)$ to each light node $v$. These labels are defined as follows. For the root $r$, $llabel(r)$ is the empty string. Then, for each light node $w \neq r$, $llabel(w)$ is a binary string such that,
• Light Label:

\( l_{\text{label}}(w) \not\in \{ l_{\text{label}}(z) \mid z \neq w; z \in \text{children}(p) \} \), where \( p = \text{parent}(w) \).

Let \( w \) be any node. Then \( h_{\text{label}}(w) \) is a binary string such that,

• Heavy label:

\[ h_{\text{label}}(w) <_{\text{lex}} \min_{\text{lex}} \{ h_{\text{label}}(z) \mid z \neq w; z \in T \cap H_P(w) \} \], where \( \text{lex} \) is the lexicographic order of two strings. Next we assign a label \( l(v) \) to each node \( v \in T \) top down as follows. We define \( l(\text{parent}(r)) \), and \( l_{\text{label}}(r) \), to be the empty string. Then for every node \( v \), \( l(v) = l(\text{parent}(\text{apex}(v))).l_{\text{label}}(\text{apex}(v)).h_{\text{label}}(v) \) (We use the . operator for concatenation of strings).

It follows from the definition that a label \( l(v) \) consists of the concatenation of alternating heavy and light labels, thus \( l(v) = h_1 \cdot l_1 \cdot h_2 \cdot l_2 \cdot \ldots \). For a bit string \( s \), we let \( s[i] \) be the \( i^{th} \) bit of \( s \). In addition to \( l(v) \) we need a label \( k(v) \) of the same length, where \( k(v)[i] = 1 \) if and only if \( l(v)[i] \) is the beginning of either a light or heavy label. The label \( k(v) \) is well defined because \( |l_{\text{label}}(v)|, |h_{\text{label}}(v)| \geq 1 \). The label, \( label(v) \), assigned to a node consists of the concatenation of \( l(v) \) and \( k(v) \). Once heavy and light labels of all the nodes have been computed, one can perform a depth first traversal [12] of the tree and compute \( l(v) \) and \( k(v) \) in linear time. Length of labels always is less than \( \log n \). Proof and more explanation about these labels and their size is given by Stephen Alstrup in his paper[1]. Next Lemma indicates how to find least common ancestor.

**Lemma 2** [1]Let \( x \) and \( y \) be two vertices of \( T \),

1. If \( l(x) = h_1 \cdot l_1 \cdot \ldots \cdot h_i \cdot t \) and \( l(y) = h_1 \cdot l_1 \cdot \ldots \cdot h_i \cdot l_i' \cdot t' \), where \( l_i \neq l_i' \) or \( l_i \cdot t \) is empty or \( l_i' \cdot t' \) is empty, then \( l(LCA(x, y)) = h_1 \cdot l_1 \cdot \ldots \cdot h_{i-1} \cdot l_{i-1} \cdot h_i \).

2. If \( l(x) = h_1 \cdot l_1 \cdot \ldots \cdot h_i \cdot \ldots \) and \( l(y) = h_1 \cdot l - 1 \cdot \ldots \cdot h_i' \cdot \ldots \), where \( h_i' \neq h_i \), then \( l(LCA(x, y)) = h_1 \cdot l_1 \cdot \ldots \cdot h_{i-1} \cdot l_{i-1} \cdot \min_{\text{lex}} \{ h_i, h_i' \} \).
Let $z = LCA(x, y)$. By definition $l(parent(apex(z)))$ is a prefix of both $l(x)$ and $l(y)$. Let $w$ be the heavy child of $z$. If $x \in T_a, y \in T_b, a, b \in children(z) \setminus \{w\}$ or $x$ is an ancestor to $y$, then Case 1 occurs; otherwise Case 2 occurs. Therefore we can find least common ancestor of two nodes in constant time with $O(\log n)$ bit labels. Key point about this label structure is each label contains labels of its all ancestors in itself. We will use same logic to find label of ancestor of a node in the center algorithm in chapter 5.
CHAPTER 3

Median

In this chapter we are going to discuss particular case of median problem. In this case we have set of active nodes, $S$, active at a particular time. We need to find median of only those active nodes. There are two cases possible for median choice,

1. Median can be any node in a tree (Actual Median).
   
   This is a vertex $x$ from $V$ which minimizes $\Sigma_{y \in S} w(y)d(x,y)$.

2. Medina should be one of the active node (Relative Median).
   
   This is a vertex $x$ from $S$ which minimizes $\Sigma_{y \in S} w(y)d(x,y)$.

Let us find solution for both cases separately.

3.1 Actual Median

**Problem Definition:** Given a tree $T$ of $n$ nodes, and $K$ active nodes in a tree at particular time, find labeling scheme to find median of those $K$ nodes. That is we need to label nodes in tree $T$ such that, when we have labels of $K$ active nodes, we can find actual median of those $K$ active nodes.

Two active nodes are called closest neighbors if their least common ancestor is deepest among least common ancestors of any other two active nodes.

**Lemma 3** When addition of weights of two closest neighbor nodes with weights less than $W/2$, is more than or equal to $W/2$ then least common ancestor (LCA) of those nodes is a median of all $K$ active nodes.
**Proof:** As shown in figure 17, let \( L \) be the least common ancestor of nodes \( A \) and \( B \). Nodes \( A \) and \( B \) are the active nodes in a tree such that there are no active nodes in subtrees of \( A \) and \( B \) respectively. Also assume that weight of subtrees starting from \( A \) and \( B \) are less than \( W/2 \), and addition of both weights is more than \( W/2 \). Now as they are closest neighbors and weight of their subtrees are less than \( W/2 \), branches other side of \( L \) will only have weight less than \( W/2 \). Therefore according to Fact 1, \( L \) is Median of all active nodes in a tree.

**Labeling Scheme:** As we have seen in lemma 3, we need to find two closest nodes such that:

1. Weight of their subtrees should be less than \( W/2 \).
2. Addition of their weight is more than or equal to \( W/2 \).

We need closest nodes in a tree because we want to make sure that there is no more weight in subtree of least common ancestor of these two nodes except weight of subtrees of closest neighbors. To find closest neighbor we number all nodes with depth first search (DFS)
numbers in a rooted tree. Later in this chapter we will see how we use this method to find closest neighbors. To find least common ancestor (LCA) we use LCA labeling scheme designed by Dr. Peleg. Using this labeling scheme we can find least common ancestor of two nodes in constant time. Detail explanation of this scheme is given in section 2.2. Our new labeling scheme label will consist of DFS number of that node in addition to $O(\log^2 n)$ bits same as in LCA labeling scheme. Therefore new labels will be of size $O(\log^2 n)$ bits.

**Algorithm:** We have a list of all active nodes. First we sort all active nodes according to their DFS number. We use DFS number as a sort criteria since depth first search always gives us neighbors of node which are closest to that node in the tree on both sides. Now we add weights of two closest neighbors whose LCA is deepest in tree to make sure we are not ignoring weights of active nodes below two nodes in consideration. If addition of weights is more than $W/2$ then we know that least common ancestor of them is Median of tree according to Lemma 3. Otherwise we add weight of one node to another and remove that node from the list and continue doing this until we find two nodes such that addition of their weights is more than $W/2$. As we go from bottom of tree to top for finding median, we can ensure that LCA of those nodes whose weight together is more than $W/2$ is the median of tree. Let us look at the algorithm for more explanation.

**Absolute Median:**

1. Let $S$ : set of all active nodes and $W$ be the total weight of all active nodes in tree.

2. Sort all nodes in $S$ according to DFS number of the nodes.

3. Find depth of $LCA$ of neighbors and store them in Logarithmic Priority Queue $Q$ in order of depth of least common ancestors.

4. while ( $Q$ not empty )
(a) Get the pair \((A, B)\) from \(Q\) whose LCA is deepest, on top of \(Q\).

(b) If \(w(A) + w(B) \geq W/2\) then declare \(LCA(A, B)\) as Median of all active nodes and break.

(c) Else add weight of \(B\) to weight of \(A\).

(d) Delete \(B\) from \(S\) and remove \(LCA(B, C)\) and \(LCA(A, B)\) from \(Q\).

(e) Let \(C\) be new neighbor of \(A\). Find \(LCA(A, C)\) and it’s depth.

(f) Insert \(LCA(A, C)\) in \(Q\), according to it’s depth.

**Explanation of algorithm:** We have a list of all active nodes. We sort them according to their DFS number. Now we know that in this list, neighbors of node in list represent neighbors of node in tree on both sides. Now we find the least common ancestor for every node with each of its neighbors. By doing this we can find out the pair of neighbors who have deepest least common ancestor among all active nodes in a tree. We insert the least common ancestors in priority queue according to their depths. Now we know the LCA on top of queue is deepest in a tree. Therefore we get the pair who constituted that LCA and add their weights. If addition of their weights is more than \(W/2\) then we declare least common ancestor of that pair as Median of active nodes in tree, otherwise we add weight of first node to second node and remove first node from the list. Taking nodes with deepest LCA insures that when we remove any node from the list we processed all active nodes bellow that node in tree. We continue on doing this until we get median of active nodes in tree.

**Runtime:** Sorting according to DFS number will take \(O(K\log K)\) time for \(K\) nodes. Now inside the while loop, we can find LCA of two nodes in constant time with the labels of those nodes. To insert LCA in to priority queue will take \(O(\log K)\) time. The rest steps in while loop are constant time steps. Therefore total time for one iteration of while loop is
$O(\log K)$. As in each iteration we are deleting one node from $S$, while loop in worst case will run maximum $K$ times with $O(\log K)$ operation each time. Therefore runtime of while loop will be $O(K \log K)$. Total run time of algorithm is $O(K \log K)$.

**Explanation by Example:** Consider tree shown in figure 18. As shown in figure all nodes are assigned DFS numbers. We want to find median of active nodes shown in a tree. We can also see sorted list $S$ and Queue status $Q$ at bottom of the figure. Weights of all active nodes are 1. Therefore total weight of tree $W$ is 8. Now as in the algorithm we took out LCA from top of queue $Q$ which is 9. So nodes 7 and 9 constitute LCA 9. Therefore we add weight of 7 in 9. Therefore weight of 9 becomes 2 and then we remove node 7 from the list. Figure 19 shows the tree and queue status after first step. You can see from figures 19 to 21 how we find the next pair to consider and add their weights to check for median. Now let us consider figure 21. Node 13 is on top of queue and nodes 9 (weight = 2) and 12(weight = 2) constitute this LCA. Now when we try to add this two weights it becomes 4 which is equal to $W/2$. Therefore we declare node 13 as median of all active nodes in tree. This step is illustrated in figure 22.

**Theorem 3** For any tree with $n$ nodes, there is a labeling scheme with labels of size $O(\log^2 n)$ bits per node, such that, given labels of any $K$ active nodes of this tree, an absolute median of these $K$ nodes can be found in $O(K \log K)$ time, using only those labels.

### 3.2 Relative median

In this case we want to find out median of active nodes which should be one of the active nodes.

**Problem Definition:** Given a tree $T$ of $n$ nodes and $K$ active nodes in a tree at particular time, find labeling scheme to find median of those $K$ nodes where median is one of the active nodes. That is we need to label nodes in tree $T$ such that, when we have labels of $K$ active nodes
Figure 18: Example for explaining algorithm Absolute Median, step 1.
Figure 19: Example for explaining algorithm Absolute Median, step 2.
Figure 20: Example for explaining algorithm Absolute Median, step 3.
Figure 21: Example for explaining algorithm Absolute Median, step 4.
Figure 22: Example for explaining algorithm Absolute Median, step 5.
nodes, we can find relative median of those $K$ active nodes.

As we all know median is defined as a node which have minimum weber function value in a tree. Similar to that we want to find a median of $K$ active nodes where median is one of them. Therefore we will find weber function for all of them and an active node with minimum weber function value will be the median of all active nodes.

**Relative Median:**

1. Let $S$: set of all active nodes in tree.
2. Find weber function, $F(x)$ for every node in $S$.
3. Declare node as median which have minimum weber function.

**Runtime:** Runtime for the algorithm for retaliative median is $O(K^2)$. As explained in chapter 2 in detail weber function value for an active node with respect to other active nodes can be found in $O(K)$ time using distance labeling scheme. Therefore for $K$ nodes it will be $O(K^2)$ time. Finding node with minimum weber function value will be $O(K)$ time. Therefore total time for algorithm is $O(K^2)$ and we need only distance labels for active nodes as explained in theorem 2.

**Explanation with example** Consider the tree shown in figure 23. Weight of every active node is 1 and also assume that all edges of tree are of 1 unit. As explained in the algorithm let us find out weber function value for every active node.

\[
F(1) = 2 + 2 + 5 + 6 + 7 + 7 + 6 = 35.
\]
\[
F(2) = 2 + 2 + 5 + 6 + 7 + 7 + 6 = 35.
\]
\[
F(5) = 2 + 2 + 3 + 4 + 5 + 5 + 4 = 25.
\]
\[
F(6) = 5 + 5 + 3 + 4 + 3 + 3 + 4 = 27.
\]
\[
F(7) = 7 + 7 + 5 + 4 + 1 + 4 + 3 = 31.
\]
Figure 23: Example of finding relative median of active nodes.

\[ F(9) = 6 + 6 + 4 + 3 + 1 + 3 + 2 = 25. \]
\[ F(11) = 7 + 7 + 5 + 4 + 3 + 4 + 1 = 31. \]
\[ F(12) = 6 + 6 + 4 + 3 + 3 + 2 + 1 = 25. \]

Therefore minimum weber function value is 25. Nodes 5, 9 and 12 have minimum weber function value, therefore any one them can be the median. As shown in figure we have declared node 9 as a median.

**Theorem 4** For any tree with \( n \) nodes, there is a labeling scheme with labels of size \( O(\log^2 n) \) bits per node, such that, given labels of any \( K \) active nodes of this tree, a relative median of these \( K \) nodes can be found in \( O(K^2) \) time, using only those labels.
CHAPTER 4

Diameter

Diameter of tree is defined as farthest distance between any two vertices in a tree. Consider tree shown in figure 24. In this tree vertices $A$ and $J$ are the farthest in a tree. As $\text{dist}(A, J)$ is greater than distance between any other two vertices in tree shown in figure 24, $(A, J)$ is the diametral pair and path between vertices $A$ and $J$ constitutes diameter path.

In this chapter we will focus on special case of trees when only some nodes are active at particular time.

**Problem Definition:** Given a tree $T$ of $n$ nodes and $K$ active nodes in a tree at particular time, find diameter of $K$ active nodes using labeling scheme. That is we need to label nodes in tree $T$ such that, when we have labels of $K$ active nodes, we can find diameter of those $K$ active nodes.

As shown in figure 25 at particular instance only some nodes in a tree are active and now we want to find diameter of only those active nodes. As seen among active nodes nodes $B$ and $F$ are the farthest. Therefore $B, F$ is the diametral pair among active nodes in a tree $T$ and path between nodes $B$ and $F$ constitutes the diameter path of active nodes.

**Lemma 4** One end of a diametral path is the deepest point in tree.

**Proof:** Let’s consider figure 26. Assume that $A, B$ and $D$ are active nodes in tree $T$. Let $AB$ be the diameter of all active nodes in tree, and $D$ be the deepest point among all $K$ active points in tree $T$. As $D$ is the deepest point in tree among all active modes, $D$
Figure 24: Diameter of tree.

Figure 25: Diameter of active nodes in a tree.
Figure 26: One end of a diametral path is the deepest point in tree.

will be the farthest point from least common ancestor (LCA) with respect to other point constituting LCA with $D$.

Now from tree shown in figure 26:

$AC \leq CD$; $C$ is LCA of $A$ and $D$.

$BS \leq SD$; $S$ is LCA of $B$ and $D$.

There are two cases possible in a tree where $D$ is deepest point as shown in figure 27.

First case is when LCA of $A$ and $D$ is deeper than LCA of $A$ and $B$ (fig A). In this case, $AC + CB \leq BC + CD$ (fig A).

The other case is when LCA of $B$ and $D$ is deeper than LCA of $A$ and $B$ (fig B). In this case, $AC + CB \leq CD + AC$ (fig B).

Since $A$, $B$ is diametral pair, $A$, $D$ is also diametral pair. In other words one end of diameter is deepest point in a tree.
Let us see now how we use this lemma to find diameter of all active nodes in a tree. First thing we need is deepest node among all active nodes in a tree. We can store in advance the depth of each node in its label. Once we get that deepest node, next thing we need to find is the farthest node to the deepest node in a tree. To find farthest node we will use distance labeling scheme for labeling nodes in a tree. In this scheme we assign labels of size $O(\log^2 n)$ to each node of tree. Using these labels we can find distance between two nodes in a tree in constant time. Detail explanation on distance labeling scheme is given in section 2.3.

**Algorithm: Diameter**

1. Find the deepest node, $D$, among all active nodes.

2. Find the farthest node, $A$, from $D$ from all active $K$ nodes.

3. Declare $AD$ is a diameter of those $K$ active nodes.

**Explanation:** From the labels we can find out depth of the node in a tree. Therefore we
can go through all active nodes to find deepest node in a tree. Using assigned distance labels we can find distance of each active point from the deepest point we found in a first step. Among all points now we select the point which is farthest from the deepest node. As proved in lemma 4 deepest point is one end of diameter and farthest point from that point will be another end of diameter.

**Run time for algorithm:** Depth of a node can be found in constant time using labels assigned. Deepest point among $K$ nodes can be found in $O(K)$ time. As we have used distance labeling scheme, distance between two nodes can be found in constant time. Therefore distance of all nodes from the deepest node found in first step can be found in $O(K)$ time. Therefore, diameter of all active nodes can be found in $O(K)$ time using $O(\log^2 n)$ size labels.

**Example:** Consider tree shown in figure 28. Node $B$ is the deepest node is tree. Therefore as explained in lemma 4, $B$ is the one end of diameter. Now we need to find the farthest
point to \( B \) among all \((K - 1)\) active points. Assuming each edge has 1 unit length, let us find out diameter of tree.

\[
\text{Dist}(B, A) = 3.
\]

\[
\text{Dist}(B, C) = 4.
\]

\[
\text{Dist}(B, D) = 6.
\]

\[
\text{Dist}(B, E) = 7.
\]

\[
\text{Dist}(B, F) = 7.
\]

\[
\text{Dist}(B, G) = 6.
\]

As, \( \text{dist}(B, E) \geq \text{dist}(B, k) \), where \( k \) is arbitrary active node in tree, \( E \) is the farthest point from \( B \) and \( B, E \) is a diametral pair.

**Theorem 5** For any tree with \( n \) nodes there is a labeling scheme with labels of size \( O(\log^2 n) \) bits per node, such that, given labels of any \( K \) active nodes of this tree, diameter of these \( K \) active nodes can be found in \( O(K) \) time, using only those labels.
CHAPTER 5

Center

The center of a tree is the collection of vertices (called the central vertices) whose eccentricity is least. In other words, it’s the collection of vertices whose longest distance to all other vertices is the smallest. In this chapter we are going to focus on special case where we have only some nodes active in a tree. We need to find center of those active nodes, S. There are two cases possible as follows,

1. Center can be one of the active node (Relative center).
   
   That is a vertex $v \in S$ such that, $ecc(v) = max\{d(v, u) \mid u \in S\}$ is smallest.

2. Center can be any node in a tree (Actual center).
   
   That is a vertex $v \in V$ such that, $ecc(v) = max\{d(v, u) \mid u \in S\}$ is smallest.

Let us find solution for both cases separately.

5.1 Relative Center:

**Problem Definition:** Given a tree $T$ of $n$ nodes, and $K$ active nodes in a tree at particular time, find relative center of $K$ active points using labeling scheme. That is we need to label nodes in tree $T$ such that, when we have labels of $K$ active nodes, we can find relative center of those $K$ active nodes. Relative center refers to the node when center is one of the $K$ active points.

**Lemma 5** Closest point to the absolute center of $K$ nodes of a tree is a relative center.
Proof: Consider figure shown in figure 29. Let us assume that $A$, $B$ is a diametral pair of $K$ nodes of a tree and $C_{actual}$ is the actual center of those $K$ nodes. Let $C_{relative}$ be the relative center. In our case $C_{relative}$ is one of the active nodes in a tree. Therefore $BC_{actual}$ is actual radius of the tree. Let us refer it as $R_{actual}$. Let us also assume that $C$ is the closest active point to $C_{actual}$.

As, $d(c, c_{actual}) = \max\{d(c, A), d(c, B)\} - R_{actual}$

Therefore,

$$\max(d(C, A), d(C, B)) - (R_{actual}) \leq \max(d(C_{relative}, A), d(C_{relative}, B)) - (R_{actual})$$

But, as $C_{relative}$ is relative center of a tree, $d(C, C_{actual}) = d(C_{actual}, C_{relative})$. Therefore closest point to actual center of tree is a Relative center of a tree.

Now, from lemma 5 we can easily find relative center if we know end points of diameter, as actual center is midpoint of diameter path. In previous chapter we have seen how to find diameter in $O(K)$ time.
Labeling Scheme: In this problem we are going to use distance labeling scheme as explained in theorem 2. We will be using $O(\log^2 n)$ bit labels. Using distance labeling scheme labels we can find distance between two nodes in tree in constant time from labels of those two nodes. More explanation is given in section 2.3.

Let us look at algorithm for finding relative center. Let $R_{actual}$ be the actual radius of a tree and $d(A, B)$ be the distance between points $A$ and $B$ in a tree.

Relative Center:

1. Let $S$ : set of all active nodes.

2. Find diameter $D$ of all $K$ active nodes and let $d_1$ and $d_2$ be the end points of diameter.

3. $R_{actual} = D/2$.

4. $min = \infty$.

5. For each node $A$ in $S$.

   (a) let $R_A = \max(d(A, d_1), d(A, d_2))$.

   (b) find $r = R_A - R_{actual}$.

   (c) if($r \leq min$ )then $min = r$ and Center = $A$.

Explanation and Runtime: We know relative center of tree will be active node nearest to actual center of tree. This means, $c_{relative}$ is an active node which gives, $C_{relative} = \{min(\max(d(d_1, v), d(d_2, v))) \mid v \in S\}$, where $S$ is set of all active nodes in tree and $d_1$ and $d_2$ are end points of diameter. In previous chapter we have seen $O(K)$ time algorithm to find diameter of tree. As shown, algorithm finds end points of diameter. Then we find active node which is closest to both end points of diameter and indirectly is closest to actual center. As we have used distance labeling scheme distance between two nodes
Figure 30: Closest point to the absolute center of tree is a relative center.

can be found in constant time. We consider every active node to find closest one to actual center. Therefore, the worst case time for for-loop will be \( O(K) \). Therefore total run time for algorithm will be \( O(K) \) and label size will be \( O(\log^2 n) \) bits.

**Example:** Consider tree shown in figure 30, where all edges are one unit. We need to find relative center of active nodes shown in a tree. According to algorithm we will first find the diameter of a tree. Therefore \( B, H \) is a diametral pair in the tree and path between \( B \) and \( H \) is a diametral path of a tree.

Therefore diameter \( D = 8 \) and, \( R_{actual} = 4 \).

Now let us find \( r \) for each node.

\[
R_A = \max(d(A, B), d(A, H)) = \max(3, 7) = 7; \quad r = 7 - 4 = 3.
\]

\[
R_C = \max(d(C, B), d(C, H)) = \max(5, 5) = 5; \quad r = 5 - 4 = 1.
\]
\[ R_D = \max(d(D, B), d(D, H)) = \max(6, 4) = 6; r = 6 - 4 = 2. \]
\[ R_E = \max(d(E, B), d(E, H)) = \max(7, 3) = 7; r = 7 - 4 = 3. \]
\[ R_F = \max(d(F, B), d(F, H)) = \max(7, 3) = 7; r = 7 - 4 = 3. \]
\[ R_G = \max(d(G, B), d(G, H)) = \max(6, 4) = 6; r = 6 - 4 = 2. \]
As \( R_C \) is minimum, \( C \) is nearest to the actual center. Therefore \( C \) is relative center of the tree as explained in Lemma 5.

**Theorem 6** For any tree with \( n \) nodes, there is a labeling scheme with labels of size \( O(\log^2 n) \) bits per node, such that, given labels of any \( K \) active nodes of this tree, relative center of these \( K \) active nodes can be found in \( O(K) \) time, using only those labels.

5.2 Actual Center:

**Problem Definition:** Given a rooted tree \( T \) of \( n \) nodes, and \( K \) active nodes in a tree at particular time, find actual center of \( K \) active points using labeling scheme. That is we need to label nodes in tree \( T \) such that, when we have labels of \( K \) active nodes, we can find actual center of those \( K \) active nodes.

Actual center refers to the node when center is any node in a tree. As we have seen in chapter 2, central vertices are middle vertices on diametral path. We have already seen how to find diameter of active nodes in a tree. Now what we need is to find middle vertex of diametral path.

Now problem with finding middle vertex of diameter is that we can not move along diameter as it will make runtime of algorithm \( O(n) \) time, if we have \( n \) nodes in a tree. We also know that diameter will pass through least common ancestor (LCA) of two end points of diameter.

Once we know LCA of end points of diameter, we also know on what side of LCA center lies on diametral path.

Therefore now we have labels of two endpoints of diameter and from that we can get label of LCA of end points of diameter. We also know which side of LCA center lies. We know
length of diameter and depth of LCA and end points of diameter. From that we can easily find out depth of center in rooted tree. Now we need to design labeling scheme such that given label of a node and depth of center and center lies on path between that node and root, we can find label of center.

**Labeling Scheme:** We are going to combine labels of distance labeling scheme and LCA labeling scheme by Stephen Alstrup with little modification as explained in section 2.3 and 2.4 respectively. Therefore,

\[ Label(v) = \{\text{distance labels of } v + \text{LCA label of } v \text{ by Stephen Alstrup} + \text{depth of a node}\} \]

Therefore, \[ \text{Size}(Label(v)) = \{\log^2 n + \log n + \log n\} = O(\log^2 n) \] bits.

As we know LCA labels have two parts, \( l(v) \) and \( k(v) \). \( l(v) = h_1 \cdot l_1 \cdot h_2 \cdot l_2 \cdot \ldots \) and \( k(v) \) is array of 0 and 1 same size as of \( l(v) \). \( k(v)[i] = 1 \) if it is start of light or heavy label in \( l(v)[i] \). Now we are going to modify \( k(v) \). In modified labels \( k(v)[i] = 1 \) if \( l(v)[i] \) is finish of label of any vertex. In this labeling scheme label of any node contains labels of all of its ancestors. That is, \( l(A) <_{lex} l(v) \), where \( v \in V \) and \( A \in \{\text{set of all ancestors of node } v\} \). Therefore from \( l(v) \) and \( k(v) \) we can find all ancestors of node \( v \) in top to bottom order when we travel \( k(v) \) from left. Therefore if you know depth of ancestor of a node we can find label of the ancestor. These labels are explained in more detail in section 2.2.

**Algorithm:**

1. Find diameter \( D \) of all \( K \) active nodes and let \( d_1 \) and \( d_2 \) be the end points of diameter.

2. Therefore radius, \( \lceil R = D/2 \rceil \).

3. Find deepest node among \( d_1 \) and \( d_2 \) and name it \( v \).

4. Let depth of center \( D_c = D_v - R \), where \( D_v \) is the depth of \( v \).

5. Find center from label of \( v \) where depth of center is \( D_c \).
Explanation and runtime: As shown in algorithm we find diameter of tree as center will be on diametral path. To find diameter it will take $O(K)$ time as explained in previous chapter. Now we know radius and diameter of active nodes in a tree. If $L$ is LCA of two end points of diameter and $v$ is deepest among end points of diameter, then it is obvious that center will lie on path between $v$ and $L$. Evidently, center will lie on the path between $v$ and root of tree. Therefore we will know the depth of center from depth of $v$. Now as explained above we can find center from the label of $v$ and depth of center in constant time. Therefore total run time of application is $O(K)$.

Example: Consider rooted tree shown in figure 31. Algorithm for finding diameter will return $B$, $D$ as a diametral pair. As $B$ is deepest end node of diameter center will lie on the path between $B$ and root. In this case $LCA(B, D)$ coincides with root of a
tree. Now depth of $B$ is 4 and $Radius = \text{diameter}/2 = 3$. Therefore depth of center is $4 - 3 = 1$. Now we know center is on path between $B$ and root and depth is 1. Therefore from label of $B$ we can find center of tree. For example if label of $B$ looks like, $\text{label}(v) = x, l(v) = 0100110, k(v) = 0101011$, where $x$ is distance label which we use for finding distances. As we know depth of center is 1. Therefore in $k(v)$ we will look position of $i$ where label of ancestor of 1st level finishes as root node has empty $l(v)$. Therefore label of center will be $l(\text{center}) = 01$.

**Theorem 7** For any tree with $n$ nodes, there is a labeling scheme with labels of size $O(\log^2 n)$ bits per node, such that, given labels of any $K$ active nodes of this tree, actual center of these $K$ active nodes can be found in $O(K)$ time, using only those labels.
CHAPTER 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we have developed labeling schemes and efficient algorithms for finding Median, Center and Diameter of active nodes in a tree. We have seen how to find absolute and relative median of active nodes in $O(K\log K)$ and $O(K^2)$ time respectively using $O(\log^2 n)$ bit labels, where $K$ is number of active nodes in a tree with $n$ nodes. We used distance labeling scheme for finding diameter of active nodes in $O(K)$ time. We have also developed labeling schemes and algorithms to find relative and absolute center of a tree in $O(K)$ time with $O(\log^2 n)$ bit labels.

6.2 Future Work

More optimized algorithms and labeling schemes can be found for these location problems. We can also improve performance of the relative median algorithm.
BIBLIOGRAPHY


