CS345 Assignment 3

Chinmay Goyal(180206) and Supreeth Baliga(180801)

1 A Hierarchical Metric

Solution:
We will first try to simplify the tree so that our further calculations become easy. Let us do this.

Lemma 1: In this problem, if an optimal solution exists for any general rooted tree, then we can convert the rooted tree into a binary tree having the same tree distances as in the optimal solution of the rooted tree and thus, this solution is also optimal.

Proof: In the optimal solution, we will have multiple points (say $p_1, p_2, ..., p_k$ where k>2) attached to the same node (say L). We can keep one of these points (say p_1) to be a child of L and we create a node Y and make it the child of L and make all remaining points $p_2, p_3, ...p_k$ to be the children of Y. Assigning h(Y)=h(L) will be valid and now we recursively convert the subtree rooted at Y to be a binary tree. Thus, in this new tree, all pairs of nodes have the same tree distances (τ) and thus, this solution is also optimal.

Note: Here, I am defining degree of a node as its number of children and by tree distances I mean τ.

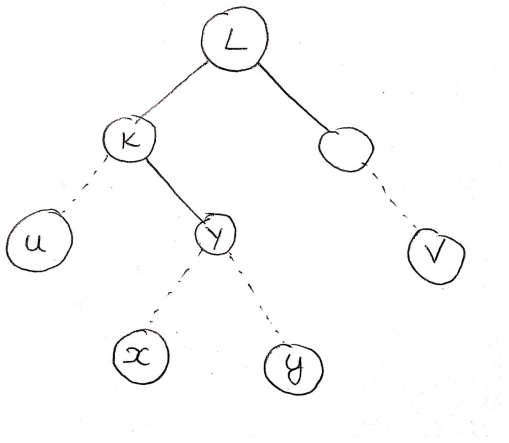
Lemma 2: If an optimal solution exists for a binary tree, then we can construct a full binary tree (each internal node has degree exactly 2) having the same tree distances (τ) as in the optimal solution for the binary tree and hence, this solution is also optimal.

Proof: Let us say we have a binary tree having an optimal solution. We see that internal nodes of degree 1 play no role in the required operations. Let us say we have a internal node L of degree 1. Suppose, we take two points from the subtree of L, then their LCA will be in the subtree rooted at the child of L. Suppose we take a point from the subtree of L and one point from some other part of the tree, then their LCA will be some ancestor of L. Thus, in both cases, L does not appear as the LCA. So basically, we can just delete L and replace it by its child. The tree distances of the remaining nodes remain intact and don't change. Thus, this solution is also optimal. We can also note that this does not violate the height property of the tree, so the new tree is in accordance with all the height properties.

Lemma 3 : Let there be points u,v ($u \neq v$) such that d(u,v) is minimum among all the distances d(i,j) such that $i \neq j$. We claim that there exists an optimal tree T such that $\tau(u, v) = d(u, v)$.

Proof: We prove this by contradiction: Let us assume that there is no such optimal tree T. Then there exists 2 cases :

- $d(u,v) < \tau(u,v)$: This case would be wrong as we want a tree T where $\tau(x,y) \leq d(x,y)$, for all points x and y.
- $d(u,v) > \tau(u,v)$: In this case we prove that we can get a better tree T'. Let us consider the following general case.



Here, let us suppose the value of $\tau(u,v)$ (which is h(L)) is less than d(u,v). So, in this situation, let us make h(L) = d(u,v). Now, heights of all nodes in the subtree of L (like K,Y,..) should be $\leq d(u,v)$. This would not violate any rule since for every pair of nodes in the subtree(say x and y arbitrarily), d(x,y) $\geq$ d(u,v) since d(u,v) was the minimum distance among the points. Similarly, we can say that for the ancestors of L that have h less than d(u,v) but more than $\tau(u,v)$, we can increase their height to d(u,v) and still we won't violate any property but will get a better solution. Thus, we found a better assignment of values (say τ') to the tree distances such that $\tau(u,v) < \tau'(u,v) = d(u,v)$.

Thus, our theorem holds that there is an optimal tree T such that $\tau(u,v)$ = d(u,v).

Lemma 4: There exists an optimal tree in which u and v appear as siblings, where u, v (u $\neq$ v) are points such that d(u,v) is minimum among all the distances d(i,j) such that i $\neq$ j.

Proof: We prove this by using the above figure again. Say, we have an optimal solution as in the figure. So, here, the heights of all nodes in the subtree rooted at L have value $\leq$ d(u,v)(Using Lemma 3). We simply swap the nodes v and Y. Now the values of heights in the subtree rooted at Y will either remain the same or increase (depending on if it violates the height property of tree) but they would not decrease. Decreasing it would simply produce a non-optimal solution. The value in the nodes K and L will also remain the same or increase. Thus, we get an optimal solution with the tree having u and v as siblings.

Theorem: Let there be points u,v (u $\neq$ v) such that d(u,v) is minimum among all the distances d(i,j) such that i $\neq$ j. We claim that there exists an optimal tree T such that $\tau(u,v) = d(u,v)$ and such nodes u and v appear as siblings in the tree T.

Proof: By combining Lemma 3 and Lemma 4 we can directly state the above theorem.

Proof for Greedy Approach :
We follow class model to prove the greedy algorithm. We first define an optimization prob-

lem, say A, then we state a theorem to get some relation between opt(T) and opt(T'). After this we use construction to get a relation between opt(T') and opt(T).

Optimization Problem A: Generate a tree T such that $\tau(x, y) \leq d(x, y)$, for all points x and y. For any other consistent tree T', $\tau(x, y) \geq \tau'(x, y)$, for all points x and y.

We use our theorem to get **a** tree T' with one less point than T. Let us say that, the points with minimum distance d between them is p_1 and p_2, and from theorem we know that there exists an optimal tree T, such that $\tau(p_1, p_2) = d(p_1, p_2)$ and p_1 and p_2, appear as siblings. Let us name their parent node X, and we know that X is also the LCA of p_1 and p_2. We merge these 3 nodes into 1 and let us name the new node p'. In the new tree T', p' acts as a leaf node, hence h(p') = 0, but this does not affect the h value of any other node as still for all the nodes, h(u) $\geq$ h(v), if u is a parent of v and all the leaf nodes have h value 0. After making this modification, we can say that $\tau'(x, p') = opt(\tau(x, p_1)) = opt(\tau(x, p_2))$ (where $x \neq p_1$ *or* p_2) because for any point y other than p_1 and p_2, the LCA of y and p_1 would lie above X and we are not modifying h of any point that is ancestor of X. For all the other nodes, the value of τ between them remains unchanged. Hence for all the nodes u,v(here we are using nodes u and v, that appear in both T and T'):

$$\tau'(u, v) = opt(\tau(u, v)) \tag{1}$$

And we know that in an optimal tree T_1, $\tau_1(u, v) \geq \tau_2(u, v)$, for all consistent trees T_2.
Using this in Equation (1), we get:

$$opt(\tau'(u, v)) \geq \tau'(u, v) \tag{2}$$

Hence :

$$opt(\tau'(u, v)) \geq opt(\tau(u, v)) \tag{3}$$

After performing this merge operation, we no longer have points p_1 and p_2, but we have a new node, p', hence we update the d matrix (d can be represented as a matrix) as follows:

$$d'(u, v) = d(u, v) \text{ where u, v are not } p_1 \text{ or } p_2 \tag{4}$$

$$d'(p', v) = min(d(p_1, v), d(p_2, v)) \tag{5}$$

Now since we will no longer have $d(p_1, p_2)$ in our updated d matrix, we store this value separately in the node p' Now proving the other side, that is constructing a T from an optimal T' Say, we have a point p' in T' which comes as a combination of some points in the original set of points P, we construct a tree T with one more point than, T', hence we convert this leaf node into a node say x and attach the points which formed this point p', as its left and right child say p_1 and p_2. In the new tree T, points p_1 and p_2 are leaf nodes, hence their h value is 0, and for their parent x , h(x) = $d(p_1, p_2)$ (Here height of all the nodes that are the ancestors of the the node x already have h value greater than or equal to $d(p_1, p_2)$, hence we will not be violating any height property)that we stored in the node. In this new tree T, $\tau(u, v) = opt(\tau'(u, v))$, for all nodes $u, v \neq p'$, and $\tau(u, p_1) = \tau(u, p_2) = opt(\tau'(u, p'))$. Hence for all the valid points:

$$\tau(u, v) = opt(\tau'(u, v)) \tag{6}$$

And we know that in an optimal tree T_1, $\tau_1(u, v) \geq \tau_2(u, v)$, for all consistent trees T_2. Using this in Equation (6), we get:

$$opt(\tau(u, v)) \geq \tau(u, v) \tag{7}$$

Hence :

$$opt(\tau'(u,v)) \leq opt(\tau(u,v)) \tag{8}$$

Combining equations (3) and (8), we get:

$$opt(\tau'(u,v)) = opt(\tau(u,v)) \tag{9}$$

Hence, this proves that if we have an optimal tree T then after applying our (theorem)greedy step we get an optimal tree T', and on having an optimal tree T', if we use construction then we get an optimal tree T.

Now, that we have proved all the required lemmas and saw the proof for the greedy approach, we will now look at the description of algorithm.

Algorithm:
Lets say we are currently working on the set of points P having n points and we have to construct tree T as specified. We have a matrix D of distances between each pair of points i.e. d_{p_i,p_j} for each $p_i, p_j \in P$. Now, we find the nodes that are closest i.e. that have the minimum distance in the graph. Say these points are p_x and p_y. We remove p_x and p_y from P and insert a new point p' in P. Now, we have a set P containing (n-1) points. Now, for this new set, we define a new distances matrix D' as follows:

if none of p_i, p_j are p'.
$d'_{p_i,p_j} = d_{p_i,p_j}$
else
$d'_{p',p_i} = d'_{p_i,p'} = min(d_{p_i,p_x}, d_{p_i,p_y})$.

Thus, a smaller instance of the problem is created. This is recursively solved to get its solution. Now, for this recursive problem, let us define the base case. In the base case, there will be just two points in the set P' (say p_1, p_2). The tree is constructed by taking a root node say K. Then we will assign the points in the set P'(contains only two points in the base case) to be the children of K and set h(K) = d_{p_1,p_2}.
Now, lets say solving the smaller instance gave the solution T'. To get the solution for current set of points (P), we built the tree T from T' by simply assigning the points p_x, p_y as children to p' and set h(p') = d_{p_x,p_y}. This completes the algorithm. The proof of correctness of the algorithm is as described in the previous sections. This algorithm is greedy since at every step of the algorithm, we take the two points with minimum distance and apply operations on them to proceed further.

Pseudo Code:

```
HierarchicalMetric(P,D) {
    if (|P|=2) {
        return tree T with root node K and its children
        p1 and p2 and h(K) = d(p1,p2),
        h(p1)=0, h(p2)=0;
    }
    else {
        Find p1 and p2 such that d(p1,p2) is minimum in matrix D;
        Remove p1,p2 from P;
        Insert new point p' in P;
```

```
        Create a new matrix D' of size (|P|)x(|P|);
        //here we have 1 less point in P than in the original situation
        for all points (x,y) in P {
            if x and y are both not p' {
                d'(x,y) = d(x,y);
            }
            else {
                // one of x or y is p'
                if(x=p') d'(p',y) = min(d(p1,y), d(p2,y));
                else d'(x,p') = min(d(x,p1), d(x,p2));
            }
        }
        Tree T <- HierarchicalMetric(P,D');
        Find p' in T and replace by     X
                                       /   \
                                      p1    p2
        Assign h(X) = d(p1,p2) and h(p1)=0, h(p2)=0;
        return T;
    }
}
```

Time Complexity:
Let us call our routine as HierarchicalMetric(P,D). Let us say the set P has n points. Now, before making the recursive call, we find the pair of points with shortest distance between them. This is done in O(n^2). The operation of creating the new set P' is done in O(n). Now, we recursively call HierarchicalMetric(P') where P' has (n-1) points. After getting the solution of P', we find our required node p' in the tree T' (solution of P') in O(n) and do the remaining operations(adding children nodes and modifying h values) in O(1). Thus, we have the recursive relation:

$$Time(n) = c_1 n^2 + c_2 n + Time(n-1) + c_3 n + c_4$$

From this, we get the time complexity as O(n^3). This is clearly a polynomial bound time complexity as specified in the question.