CS 325- Homework Assignment 4

Problem 1: (5 points) Road Trip:

Suppose you are going on a road trip with friends. Unfortunately, your headlights are broken, so you can only drive in the daytime. Therefore, on any given day you can drive no more than d miles. You have a map with n different hotels and the distances from your start point to each hotel x1< x2< ... < xn. Your final destination is the last hotel. Describe an efficient greedy algorithm that determines which hotels you should stay in if you want to minimize the number of days it takes you to get to your destination. What is the running time of your algorithm?

Answer – The best algorithm for this problem would be the Greedy Algorithm. We can travel the farthest for the day within d miles and stay there for a night and then again we can go to the farthest as we can within d miles from the current location and stay there for the next night and so on.

Algorithm :

```
Function roadTrip()
{
        $x_0$ = start point
        // all hotels are covered

for(i=0; i<=n ; i++) {
        While( $x_i$ != $x_n$)
        {
                If( $x_i - x_0$) = d
                        Stay
                If( $x_i - x_0$) > d
                {
                        //Choose the one previous to this location to stay
                        xi-1 – x0
                }
        }
        If( $x_i$ == $x_n$ )
        Choose this as the one to stay and the problem  is done.
} }
```

Running time – If there are n number of hotels on the route, then we need to evaluate this for each of it. Therefore, the running time would be θ(n)

Problem 2: (4 points) CLRS 16-1-2 Activity Selection Last-to-Start

Suppose that instead of always selecting the first activity to finish, we instead select the last activity to start that is compatible will all previously selected activities. Describe how this approach is a greedy algorithm, and prove that it yields an optimal solution.

Answer :

Let us consider a non-empty subproblem S_k and let a_m be an activity in S_k with last finish time. Then a_m is included in some maximum size subset of mutually compatible activities of S_k .

Proof :

- Let A_k be a maximum-size subset of mutually compatible activities in S_k , and let a_j be the activity in A_k with the last start time.
- If $a_j = a_m$, we are done, since we have shown that a_m is in some maximum-size subset of mutually compatible activities of S_k .
- If $a_j \neq a_m$, then $S_m \geq S_j$ as a_m was the one to start at last of all activities.
- Let the set $A_k' = A_k - \{a_j\} \cup \{a_m\}$. Then the activities in A_k' are mutually compatible as a_j was the last to start in A_k. This leads to $S_j \geq S_i$ for all $a_i \in A_k$. Hence $S_m \geq S_i$ for all $a_i \in A_k$ as a_m started after a_j .
- Since $| A_k | = | A_k' |$, we conclude that A_k' is a maximum-size subset of mutually compatible activities of S_k , and it includes a_m which is the greedy choice.

Problem 3: (6 points) MST

Consider an undirected graph G=(V,E) with nonnegative edge weights w(u,v)>=0. Suppose that you have computed a minimum spanning tree T, and that you have also computed shortest paths to all vertices from vertex s ∈ V. Suppose each edge weight is increased by 1: the new weights w'(u,v) = w(u,v) + 1.

(a) Does the minimum spanning tree change? Give an example if it changes or prove it cannot change.

Answer - No, the Minimum spanning tree does not change.

Let T be the minimum spanning tree of graph G.

$T = \{ e_1, e_2, e_3, .. e_{n-1}\}$

Let the total weight of the graph G(V,E) be w(T), which consists of weights w(e).

$w(T) = w(e_1) + w(e_2) + w(e_3) + . . . +w(e_{n-1})$ ------------- (eq1)

Let G'(V,E) be the graph for which the edge weight is increased by 1 from the graph G.

Therefore w'=w (e)+1 for all e ∈ E . We can prove that the minimum spanning tree T of G is the minimum spanning tree of G' as well.

$w(T') = w'(e_1) + w'(e_2) + w'(e_3) + . . . + w'(e_{n-1})$

= $w(e_1) + 1 + w(e_2) + 1 + w(e_3) + 1 + \ldots + w(e_{n-1}) + 1$

$w(T') = w(T) + (n-1)$ ------ from eq1

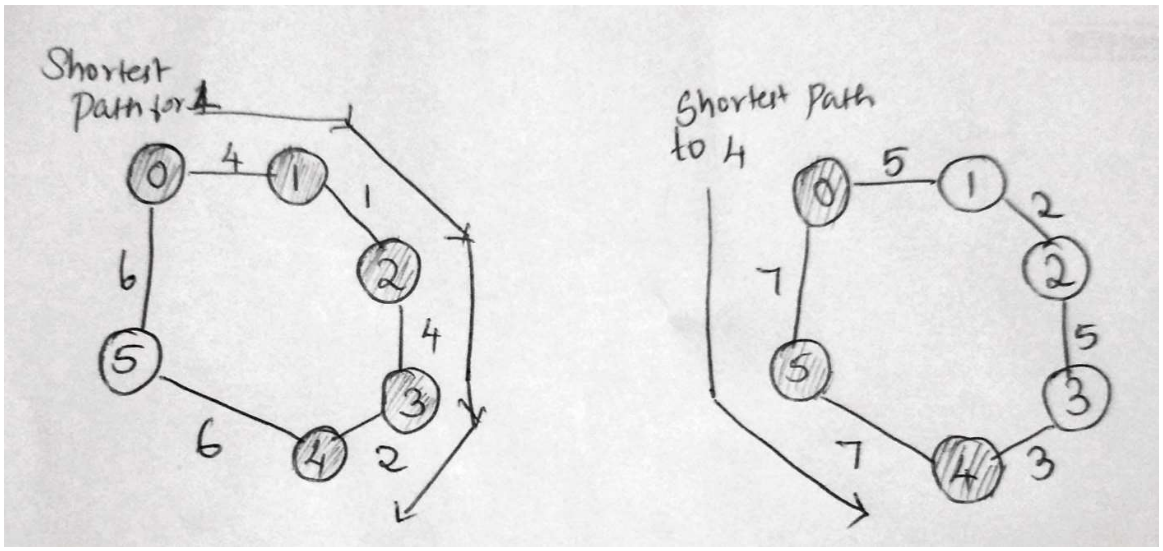
The number of 1's adds up to n-1 since there are n-1 edges. The weight of MST in T' is (n-1) more than the weight of MST T.

If w(T') < w(T) + (n-1), it would mean that w(T) is not an MST of T

Hence any MST in G is an MST in G'.

(b) Do the shortest paths change? Give an example of a graph where the paths change or prove that the paths cannot change.
Answer – Yes, the shortest path would change on the above condition.

Consider the below graph, the shortest path to the vertex 4 is through the edges {4,1,4,2} which gives a total as 11. But when the weights are increased as depicted on the right side of the image, the total adds up to
5+2+5+3 = 15 and on the other side we have 7+ 7 = 14 which is the shortest path.



Problem 4: (15 points Euclidean MST Implementation Implement an algorithm in the language of your choice (C, C++ or Python) that determines the MST in a graph G=(V,E) where the vertices are points (x,y) in the rectangular coordinate system and the weight of the edge between each pair of points is the Euclidean distance between those points. To avoid floating point precision problems in computing the square-root, we will always round the distance to the nearest integer. For all u, v ⍰ V, u = (x1,y1), v = (x2,y2) and w(u,v) = d(u, v) = $nearestint$ √(x1 − x2)2 + (y1 − y2)2.

a) Verbal description of Algorithm

- We can use the Prim's algorithm to find the minimum spanning tree.
- First we initialize an array mstTree that stores the values of the vertex and edge which is included.
- Then we assign the key values to all the vertices and intialise it to ∞
- We pick the first vertex and assign the key value to 0
- While MST does not include all the vertices in the graph, we do the following :
    a. We pick a vertex(u) which is not in MST and that has the lowest key value
    b. We include the vertex u to the MST Tree
    c. The key value for all the adjacent vertices of u are then updated. These adjacent vertices are again computed by taking the ones which has the minimum edge and again comparing it to the key value. If it is less than the key value, we can update it.

b) Pseudocode

```
Function findMST(graph[][])

{

   Int N = Read the total number of Vertices from file

//Read the coordinate values
   x[] = Read from file
   Y[] = Read from file

//Calculating the distance between all the coordinates
  for(int i=0;i<V;i++) {
     for(int j=0; j<V; j++)
     {
               e[i] = sqrt((x1-x2)^2 + (y1-y2)^2))
               graph[i][j]=e[i];
     }

    Int p[];
    Int Key[];
    Boolean mstTree[]; // to update when the vertex is included
      for( i=0 ;I < V ; i++) // where V is the total number of vertex
  {
      Key[] = ∞
      mstTree = false;
   }
 Key[] = 0 // Including the first vertex in the MST
 p[0] = -1; //Setting the parent node of the first node to -1 as it is the root node

for (int count = 0; count < V - 1; count++) {

        int u = minEdge(key, mstTree);

        // Adding the minimum edge

        mstTree[u] = true;

        // Updating key value and parent index of the adjacent vertices which included in MST by
considering the vertices which are not yet covered

        for (int v = 0; v < N; v++)

            //Updating the key only if the graph[u][v] is lesser than the key

            if ((graph[u][v] != 0) && (mstTree[v] == false) && (graph[u][v] < key[v])) {

             p[v] = u;

             key[v] = graph[u][v];

          }  }  }
```

c) Analysis of theoretical running time

As we can see from the pseudocode, there are two for loops which goes on up to V(total number of vertices). Therefore, the theoretical running time of the algorithm is $O(V^2)$ which implies that we need to visit each and every vertices.

When Adjacency list is used, it can be reduced to O(E log V) using binary heaps.