**Solution 1.**

**Heart of the solution:**

S[i][j] = minimum length of the triangulation considering the convex polygon formed by (xi,yi) and (xj,yj)

S[i][j]= min (S[i][k] + S[k][j] + distance between (xi, yi) and (xj, yj))

Where 1 <= i < k < j <= n

Return: S[0][n] – distance between edge (x0, y0) and (xn, yn)

As (x0, y0) and (xn, yn) is an edge of the original polygon.

**Algo:**

minTriangulationLen()

**1 from**(d=0 to; d<n-1; d++)

2 **for**(**int** l=0; l<(n-d-1); l++)

3 **int** r = l+d+1;

4 s[l][r] = (**float**) Double.*MAX\_VALUE*;

5 **if**(r == l+1)

6 s[l][r] = 0;

7 **for**(**int** k=l+1; k<r; k++)

8 temp = s[l][k] + s[k][r] + dist (l ,r);

9 **if**(s[l][r] > temp)

10 s[l][r] = temp;

11 temp = dist(0, n-1);

12 return((**int**) (s[0][n-1] - temp));

**Running Time:**

O(n3)

**Solution 2**

**Pseudo-Code:**

**DFS-RUN ( G=(V,E), s )**

1. count = 0;

2. seen[v]=false for every vertex v

3. for all vertices

4. if (seen[v]==false)

5. DFS(s)

6. count ++;

7. return count;

**DFS(v)**

1. seen[v]=true

2. for every neighbor u of v

3. if not seen[u] then DFS(u)

**Description:**

The above algorithm runs DFS for every vertex that has not been visited.

Now if the vertex is not visited while DFS traversal then it means it’s on a disjoint part of the graph (or not reachable or not connected) from the vertex initiating DFS. Thus we increment the counter.

Hence the return value is the count.

**Running Time:**

Running time here will be O(n+m) this is because it’s the running time of DFS. And only O(1) steps of procedural changes have been made thus it will remain same as that of original DFS.

**Solution 3**

**Description of Algo:**

The algo will be similar to that of the DFS. However we will do it for a matrix. Here we will keep changing the values of the o to 1 once they are visited. Hence we’ll always get the shortest path to the house(3).

**Algorithm:**

**BFS ( G=(V,E), s )**

1. seen[v]=false, dist[v]=∞ for every vertex v

2. beg=1; end=2; Q[1]=s; seen[s]=true; dist[s]=0;

3. while (beg<end) do

4. head=Q[beg];

5. for every u s.t. (head,u) is an edge and

6. not seen[u] do

7. Q[end]=u; dist[u]=dist[head]+1;

8. seen[u]=true; end++;

9. beg++;

**Running Time:**

Running time will become O(m\*n) this is because the matrix is of size m\*n and the BFS will run in O(m+n) steps.

Couldn’t complete the program dues to time constraints. Have mailed the professor.