MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

Skill Based Project report

on

Minimum Cost Graph & Single Shortest Path

Submitted By:

Satya Singh Chandel

0901CA211055

Mentor:

Dr. Anshu Chaturvedi

(Professor)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE

GWALIOR - 474005 (MP) est. 1957

July-December 2021

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

CERTIFICATE

This is certified that **Satya Singh Chandel** (0901CA211055) has submitted the project report titled **Minimum Cost Graph & Single Shortest Path** on problem of **Data Structure & Algorithms** under the mentorship of **Dr. Anshu Chaturvedi** as the requirement of skill based mini project.

[illegible]

Dr. Anshu Chaturvedi

Computer Science and Engineering

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV ,Bhopal)

DECLARATION

I hereby declare that the work being presented in this project report, for the fulfilment of partial requirement of the skills based mini project in 2nd year of Master of Computer Application in Computer Science and Engineering at Madhav Institute of Technology & Science, Gwalior is an authenticated and original record of my work under the mentorship of **Dr. Anshu Chaturvedi**, (Professor), MITS Gwalior.

I declare that I have not submitted the matter embodied in this report anywhere else.

[illegible]

Satya Singh Chandel

0901CA211055
2021-2023 Year,
Master of Computer Application,
Computer Science and Engineering

ACKNOWLEDGEMENT

The full semester project has proved to be pivotal to my career. I am thankful to my institute ,**Madhav Institute of Technology and Science** to allow me to continue my disciplinary project. I extend my gratitude to the Director of the institute ,**Dr. R. K. Pandit** and Dean Academics ,**Dr. Manjaree Pandit** for this.

I would sincerely like to thank my department, **Department of Computer Science and Engineering**, for allowing me to explore this project. I humbly thank **Dr. Manish Dixit**, Professor and Head, Department of Computer Science and Engineering, for his continued support during the course of this engagement, which eased the process and formalities involved.

I am sincerely thankful to my faculty coordinator. I am grateful to the guidance of **Dr. Anshu Chaturvedi**, (Professor), Computer Science and Engineering, for her continued support and guidance throughout the project. I am also very thankful to the faculty and staff of the department.

[illegible]

SATYA SINGH CHANDEL
0901CA211055
1st Year,
Master of Computer Application,
Computer Science and Engineering

Abstract

Minimum Cost Graph is a approach to connect each node with every other node and similarly for the other N nodes but in the worst case the time complexity will be NN.

The other way is to find the cost of every pair of vertices with the Euclidean distance

The task is to connect the graph such that every node has a path from any node with minimum cost.

A directed graph, which may contain cycles, where every edge has weight, the task is to find the minimum cost of any simple path from a given source vertex ‘s’ to a given destination vertex ‘t’. Simple Path is the path from one vertex to another such that no vertex is visited more than once. If there is no simple path possible then return INF(infinite).

CONTENTS

Minimum Cost Graph

Input: N = 3, edges [] [] = {{1, 1}, {1, 1}, {2, 2}, {3, 2}}

Output: 1.41421

Since (2, 2) and (2, 3) are already connected.

So we try to connect either (1, 1) with (2, 2)

or (1, 1) with (2, 3) but (1, 1) with (2, 2)yields the minimum cost.

Input: N = 3, edge [] [] = {{1, 1}, {2, 2}, {3, 3}}

Output: 2.82843

Approach

The brute force approach is to connect each node with everyother node and similarly for the other N nodes but in the worst case the time complexity will be NN.

The other way is to find the cost of every pair of vertices withthe Euclidean distance and those pairs which are connected will have the cost as 0.

After knowing the cost of each pair we will apply the Kruskal Algorithm for the minimum spanning tree and it will yield the minimum cost for connecting the graph. Note that for KruskalAlgorithm, you have to have the knowledge of Disjoint
Set Union (DSU).

Below is the implementation of the above approach:

```
#include <bits/stdc++.h>
usingnamespacestd;

const int N = 500 + 10;

int arr[N], sz[N];

void initialize()
{
    for(int i = 1; i< N; ++i) {
        arr[i] = i;
        sz[i] = 1;
    }
}
```

```
int root(int i)
{
        while(arr[i] != i)i =
                arr[i];
        return i;
}

void Union(int a, int b)
{
        a = root(a);b =
        root(b);

        if(a != b) {
                if(sz[a] <sz[b])swap(a,
                        b);

                sz[a] += sz[b];arr[b] = a;
        }
}

double minCost(vector<pair<int, int>>& p)
{


        int n = (int)p.size();


        vector<pair<double, pair<int, int>>> cost;


        for(int i = 0; i < n; ++i) {
                for(int j = 0; j < n; ++j) {
                        if( i != j) {

                                int x = abs(p[i].first - p[j].first)
                                            + abs(p[i].second - p[j].second);


                            if(x == 1) {
                                    cost.push_back({ 0, { i + 1, j + 1 } });
                                    cost.push_back({ 0, { j + 1, i + 1 } });
                            }
                            else{
                                    int a = p[i].first - p[j].first;
                                    int b = p[i].second - p[j].second;a *= a;
                                    b *= b;
                                    double d = sqrt(a + b);
                                    cost.push_back({ d, { i + 1, j + 1 } });
                                    cost.push_back({ d, { j + 1, i + 1 } });
                            }
                        }
                }
        }


        sort(cost.begin(), cost.end());


        initialize();

        double ans = 0.00;
        for(auto i : cost) {
                double c = i.first;
                int a = i.second.first;
                int b = i.second.second;


                if(root(a) != root(b)) {Union(a, b);
                        ans += c;
                }
        }

        return ans;
}

int main()
{


        vector<pair<int, int>> points = {

                {  1,   1  },
                {  2,   2  },
                {  2,   3  }
        };


        cout<<minCost(points)<<endl;
cout<< “Satya Singh Chandel – 0901CA211055”;
        return0;
```

```
}
```

Output:

```
1.41421
```

Satya Singh Chandel Enrollment – 0901CA211055

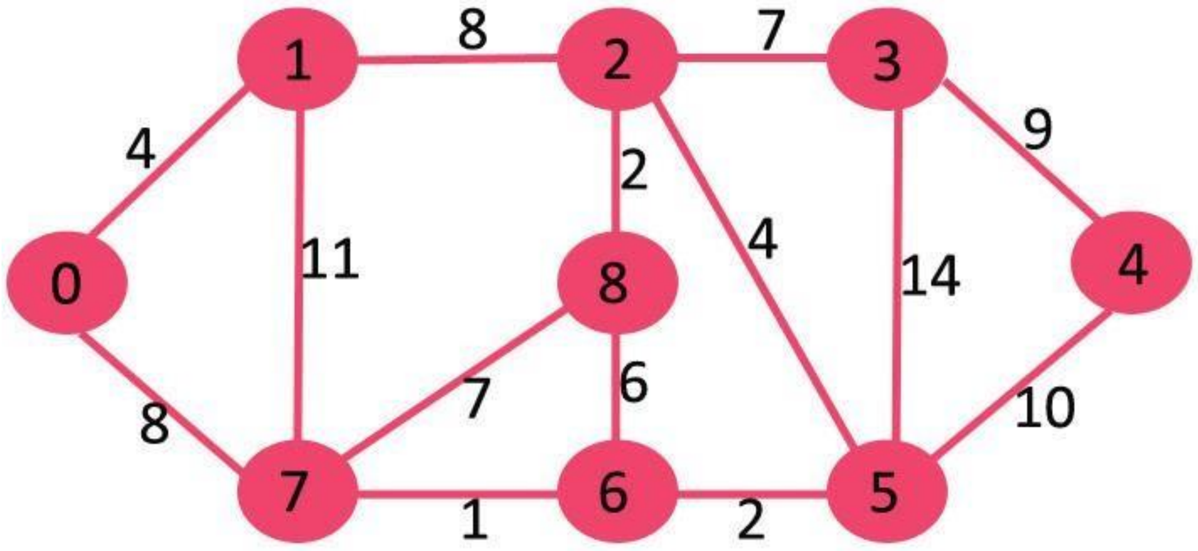
Time Complexity: O(N*N)

Auxiliary Space: O(N*N)

SINGLE SOURCE SHORTEST PATH

Given a graph and a source vertex in the graph, find theshortest paths from the source to all vertices in the given graph.

Examples:
Input: src = 0, the graph is shown below.
Source to all vertices in the given graph.



Output: 0 4 12 19 21 11 9 8 14

Explanation: The distance from 0 to 1 = 4.

The minimum distance from 0 to 2 = 12. 0->1->2

The minimum distance from 0 to 3 = 19. 0->1->2->3

The minimum distance from 0 to 4 = 21. 0->7->6->5->4

The minimum distance from 0 to 5 = 11. 0->7->6->5

The minimum distance from 0 to 6 = 9. 0->7->6

The minimum distance from 0 to 7 = 8. 0->7

The minimum distance from 0 to 8 = 14. 0->1->2->8

Dijkstra shortest path algorithm using Prim'sAlgorithm in O(V2):

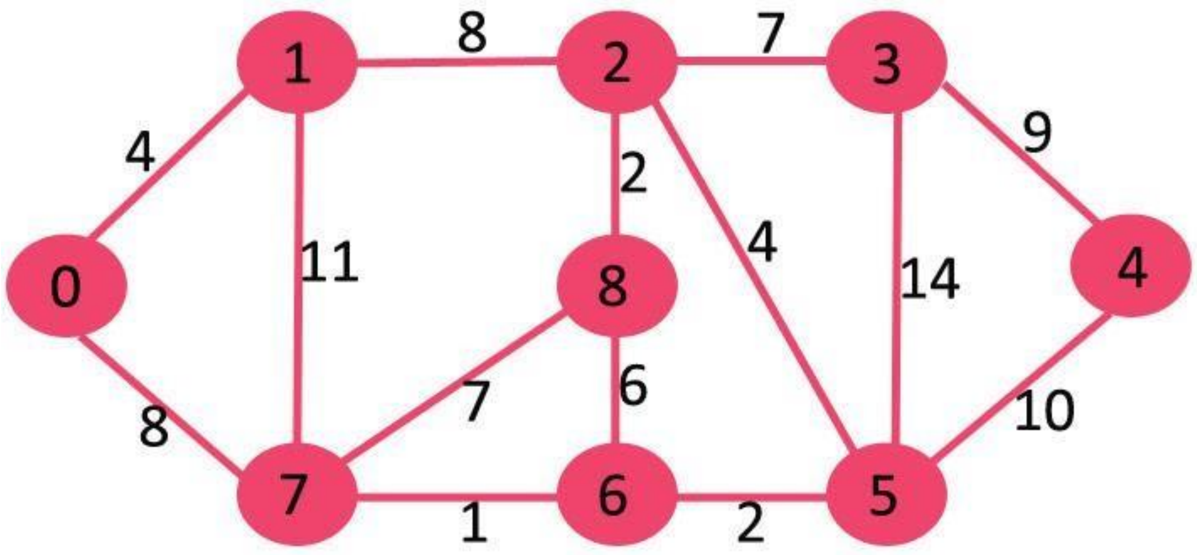
Dijkstra's algorithm is very similar to Prim's algorithm forminimum spanning tree.

Like Prim's MST, generate a SPT (shortest path tree) with a given source as a root. Maintain two sets, one set contains vertices included in the shortest-path tree, other set includes vertices not yet included in the shortest-path tree. At every step of the algorithm, find a vertex that is in the other set (set not yet included) and has a minimum distance from the source.

Follow the steps below to solve the problem:

- Create a set **sptSet** (shortest path tree set) that keeps track of vertices included in the shortest-path tree, i.e., whose minimum distance from the source is calculated and finalized. Initially, this set is empty.
- Assign a distance value to all vertices in the input graph. Initialize all distance values as **INFINITE**. Assign the distance value as 0 for the source vertex so that it is picked first.
- While **sptSet** doesn't include all vertices
  - Pick a vertex **u** which is not there in **sptSet** and has a minimum distance value.
  - Include u to **sptSet**.
  - Then update distance value of all adjacent verticesof u.
    - To update the distance values, iterate throughall adjacent vertices.
    - For every adjacent vertex v, if the sum of the distance value of u (from source) and weightof edge u-v, is less than the distance value ofv, then update the distance value of v.

Below is the illustration of the above approach:

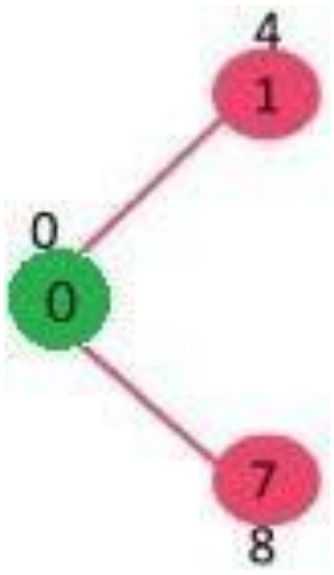


To understand the Dijkstra’s Algorithm lets take a graph and find the shortest path from source to all nodes.

Consider below graph and src = 0

Step 1:

- Now pick the vertex with a minimum distance value. Thevertex 0 is picked, include it in sptSet. So sptSet becomes {0}. After including 0 to sptSet, update distancevalues of its adjacent vertices.
- Now pick the vertex with a minimum distance value. Thevertex 0 is picked, include it in sptSet. So sptSet becomes {0}. After including 0 to sptSet, update distancevalues of its adjacent vertices.
- Adjacent vertices of 0 are 1 and 7. The distance values of1 and 7 are updated as 4 and 8.
- The following subgraph shows vertices and their distance values, only the vertices with finite distance values are shown. The vertices included in SPT are shown in green Color.

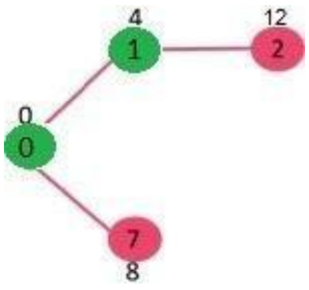


Step 2:

- Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). The vertex 1 ispicked and added to sptSet.

- So sptSet now becomes {0, 1}. Update the distance values of adjacent vertices of 1.

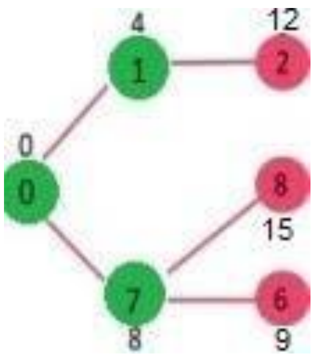
- The distance value of vertex 2 becomes 12.



Step 3:

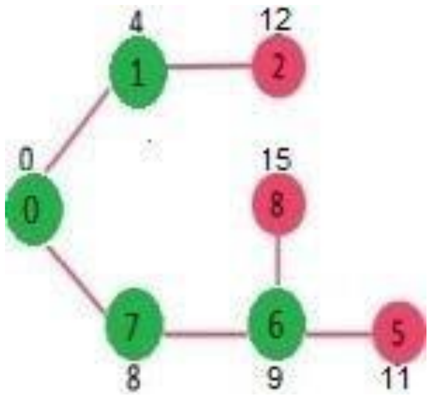
- Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 7 is picked. So sptSet now becomes {0, 1, 7}.

- Update the distance values of adjacent vertices of 7. The distance value of vertex 6 and 8 becomes finite (15 and 9 respectively).

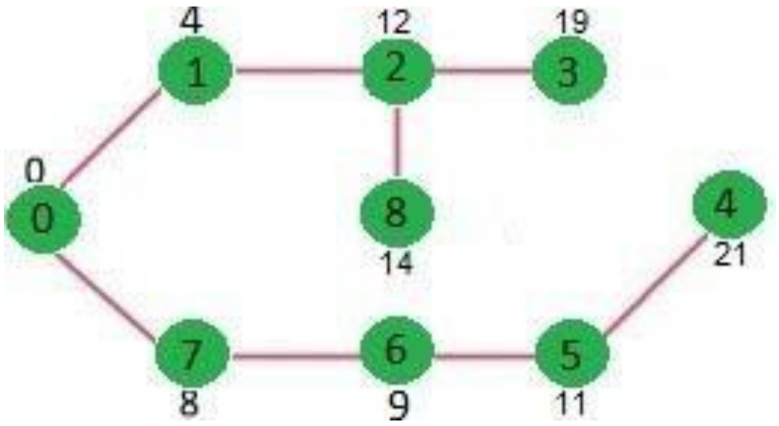


Step 4:

- Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 6 is picked. So sptSet now becomes {0, 1, 7, 6}.
- Update the distance values of adjacent vertices of 6. The distance value of vertex 5 and 8 are updated.



We repeat the above steps until sptSet includes all vertices of the given graph. Finally, we get the following Shortest Path Tree (SPT).



Below is the implementation of the above approach:

```
#include <iostream>
usingnamespacestd;
#include <limits.h>

#define V 9

int minDistance(int dist[], bool sptSet[])
{

    int min = INT_MAX, min_index;

    for(intv = 0; v < V; v++)
        if(sptSet[v] == false&&dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

void printSolution(intdist[])

    cout<<"Vertex \t Distance from Source"<<endl;
    for(inti = 0; i< V; i++)
        cout<<i<<" \t\t\t\t"<<dist[i] <<endl;
}

void dijkstra(int graph[V][V], int src)
{
    int dist[V];
i

    bool sptSet[V];
```

```
      for(int i = 0; i< V; i++)
            dist[i] = INT_MAX, sptSet[i] = false;


      dist[src] = 0;


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

            int u = minDistance(dist, sptSet);


            sptSet[u] = true;


            for(intv = 0; v < V; v++)


                  if(!sptSet[v] && graph[u][v]
                        &&dist[u] != INT_MAX
                        &&dist[u] + graph[u][v] <dist[v])dist[v] =
                        dist[u] + graph[u][v];
      }

      printSolution(dist);
}

int main()
{

      int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                              { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                              { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                              { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                              { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                              { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                              { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                              { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                              { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
    dijkstra(graph, 0);
cout<< "Satya Singh Chandel Enrollment – 0901CA211055";
      return 0;
}
```

Output:

```
Vertex
                Distance from Source

  0                  0
  1                  4
  2                  12
  3                  19
  4                  21
  5                  11
  6                  9
  7                  8
  8                  14
```

Satya Singh Chandel Enrollment – 0901CA211055

Time Complexity: O(V2)
Auxiliary Space: O(V)