PROJECT REPORT

Submitted by

Dushyant Rao [RA2011028010106],
Avipsha Panigrahi[RA2011028010101],
Shubhra Kumari[RA2011028010101]

Under the Guidance of

Dr. M Shobana
Assistant Professor, Department of Networking and Communication

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

with specialization in cloud computing

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR - 603203

JUNE 2022

SRM INSTITUTION OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that this lab report titled **Cable Networking Using Kruskal** is the bonafide work done by Dushyant Rao (RA2011028010106), Avipsha Panigrahi (RA2011028010101), Shubhra Kumari (RA2011028010093) who carried out the lab project under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

SIGNATURE
Dr. M Shobana
DAA – Course Faculty
Assistant Professor
Department of Networking and Communication

SIGNATURE
Head of Department

TABLE OF CONTENTS

PROBLEM STATEMENT

Most of the cable network companies use the Disjoint Set Union data structure in Kruskal's algorithm to find the shortest path to lay cables across a city or group of cities.

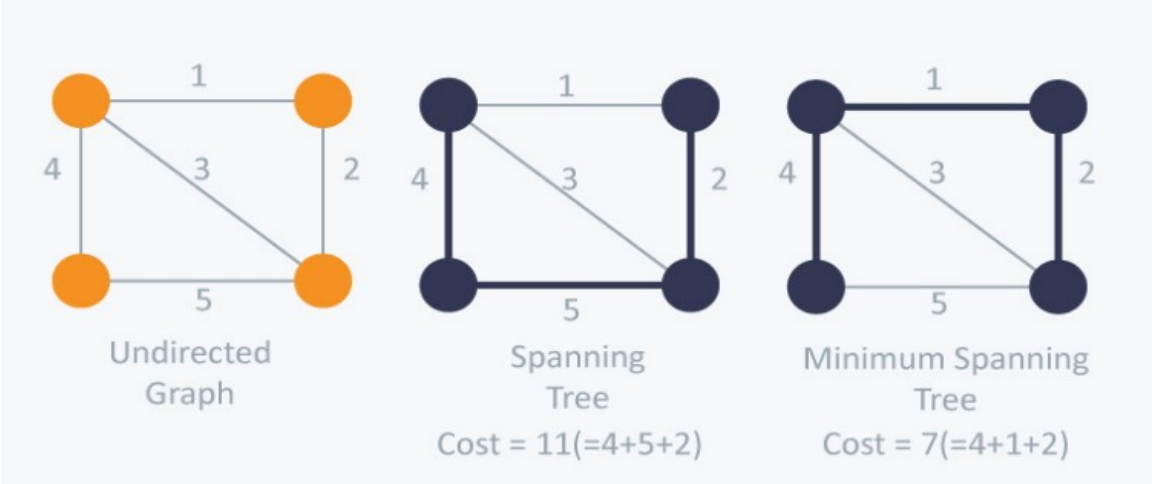
Which leads us to this post on the properties of Disjoint sets union and minimum spanning tree along with their example.

PROJECT DESCRIPTION

- **In this project we use Kruskal's algorithm to find the shortest path to lay cables across a city or group of cities.**
- **Which leads us to this post on the properties of Disjoint sets union and minimum spanning tree along with their example.**
- **Firstly we analyse Kruskal's algorithm through diagrams and spanning trees.**
- **The algorithm steps provide a better insight of the problem, after which an example : India is taken.**
- **The Indian cable network is studied and a minimum spanning tree of the problem is generated.**
- **Finally code and output are printed.**

KRUSKAL'S ALGORITHM

- **Spanning tree is the sum of weights of all the edges in a tree is known as Kruskal's algorithm.**
- **A minimum spanning tree (MST) is one which costs the least among all spanning trees.**

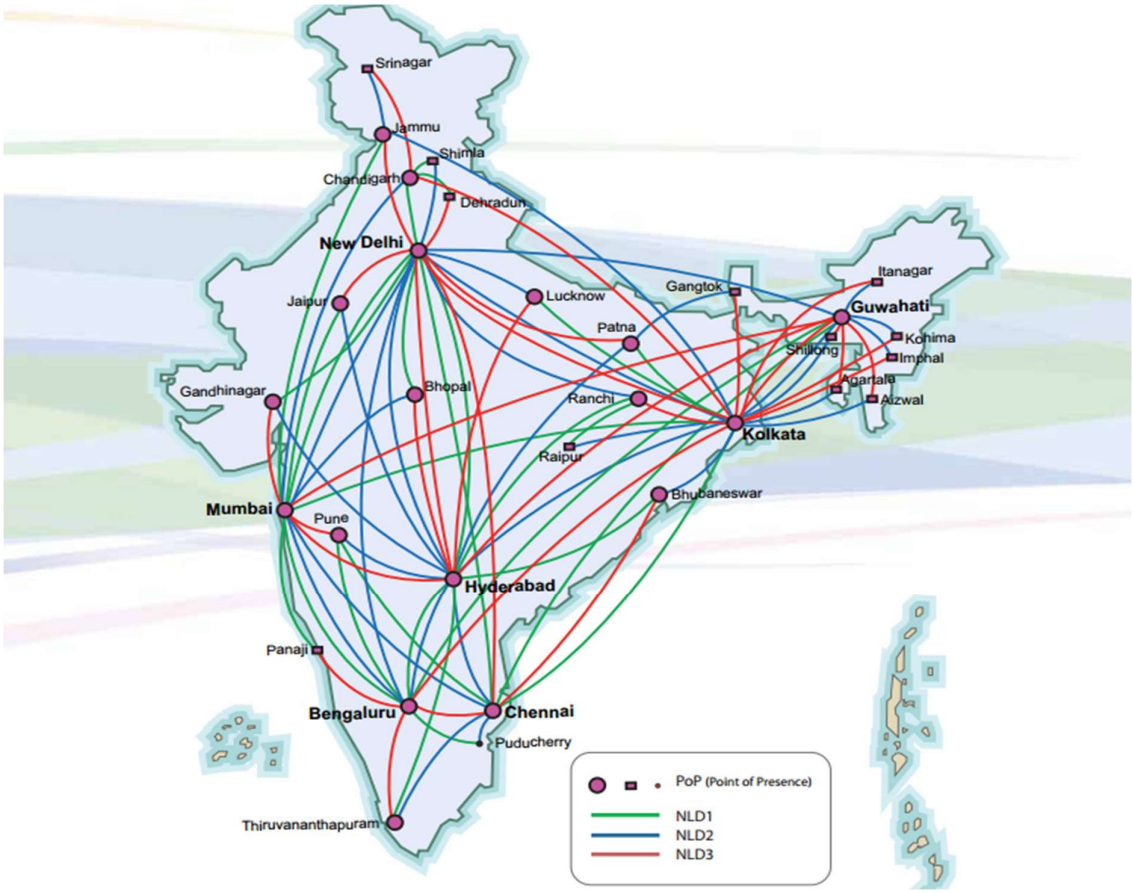


ALGORITHM STEPS

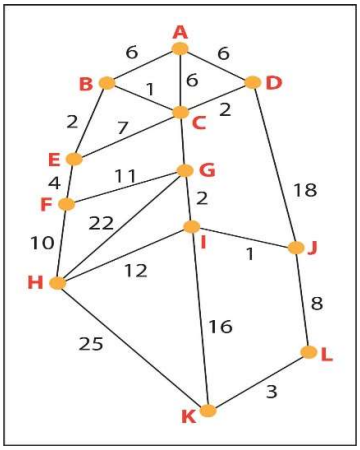
1. **Remove all loops and parallel edges.**
2. **Arrange all the edges on the graph in ascending order. Kruskal's algorithm considers each group as a tree and applies disjoint sets to check how many of the vertices are part of other trees.**
3. **Add edges with least weight; we begin with the edges with least weight/cost. Hence, B, C is connected first considering their edge cost only 1.**

PROBLEM

- **Let us consider a map of INDIA.**



- **let's Let's simplify the map by converting it into a graph as below and naming important locations on the map with letters and distance in meters (x 100).**
- **So for the given map, we have a parallel edge running which is of length 2.4kms(2400mts).**
- **We will remove the parallel road and keep the 1.8km (1800m) length for representation.**



- **Arrange all the edges on the graph in ascending order. Kruskal's algorithm considers each group as a tree and applies disjoint sets to check how many of the vertices are part of other trees.**

B,C	1
I,J	1
B,E	2
C,G	2
G,I	2
C,D	2
K,L	3
E,F	4
A ,B	6
A,C	6
A,D	6
E,C	7
J,L	8
F,H	10
F,G	11
H,I	12
I,K	16
D,J	18
G,H	22
H,K	25

CODE

```
#include<bits/stdc++.h>
using namespace std;

// Creating shortcut for an integer pair
typedef pair<int, int> iPair;

// Structure to represent a graph
struct Graph
{
    int V, E;
    vector< pair<int, iPair> > edges;

    // Constructor
    Graph(int V, int E)
    {
        this->V = V;
        this->E = E;
    }

    // Utility function to add an edge
    void addEdge(int u, int v, int w)
    {
        edges.push_back({w, {u, v}});
    }

    // Function to find MST using Kruskal's
    // MST algorithm
    int kruskalMST();
};

// To represent Disjoint Sets
struct DisjointSets
{
    int *parent, *rnk;
    int n;

    // Constructor.
    DisjointSets(int n)
    {
        // Allocate memory
        this->n = n;
        parent = new int[n+1];
```

```
        rnk = new int[n+1];

        // Initially, all vertices are in
        // different sets and have rank 0.
        for (int i = 0; i <= n; i++)
        {
            rnk[i] = 0;

            //every element is parent of itself
            parent[i] = i;
        }
    }

    // Find the parent of a node 'u'
    // Path Compression
    int find(int u)
    {

        if (u != parent[u])
            parent[u] = find(parent[u]);
        return parent[u];
    }

    // Union by rank
    void merge(int x, int y)
    {
        x = find(x), y = find(y);

        /* Make tree with smaller height
        a subtree of the other tree */
        if (rnk[x] > rnk[y])
            parent[y] = x;
        else // If rnk[x] <= rnk[y]
            parent[x] = y;

        if (rnk[x] == rnk[y])
            rnk[y]++;
    }
};

/* Functions returns weight of the MST*/

int Graph::kruskalMST()
{
    int mst_wt = 0; // Initialize result

    // Sort edges in increasing order on basis of cost
    sort(edges.begin(), edges.end());
```

```
    // Create disjoint sets
    DisjointSets ds(V);

    // Iterate through all sorted edges
    vector< pair<int, iPair> >::iterator it;
    for (it=edges.begin(); it!=edges.end(); it++)
    {
        int u = it->second.first;
        int v = it->second.second;

        int set_u = ds.find(u);
        int set_v = ds.find(v);

        // Check if the selected edge is creating
        // a cycle or not (Cycle is created if u
        // and v belong to same set)
        if (set_u != set_v)
        {
            // Current edge will be in the MST
            // so print it
            cout << u << " - " << v << endl;

            // Update Min Len
            mst_wt += it->first;

            // Merge two sets
            ds.merge(set_u, set_v);
        }
    }

    return mst_wt;
}

// Driver program to test above functions
int main()
{

    int V = 9, E = 14;
    Graph g(V, E);

    // making above shown graph
    g.addEdge(0, 1, 4);
    g.addEdge(0, 7, 8);
    g.addEdge(1, 2, 8);
    g.addEdge(1, 7, 11);
    g.addEdge(2, 3, 7);
    g.addEdge(2, 8, 2);
    g.addEdge(2, 5, 4);
    g.addEdge(3, 4, 9);
```

```
    g.addEdge(3, 5, 14);
    g.addEdge(4, 5, 10);
    g.addEdge(5, 6, 2);
    g.addEdge(6, 7, 1);
    g.addEdge(6, 8, 6);
    g.addEdge(7, 8, 7);

    cout << "Shortest Distances between Cities \n";
    int mst_wt = g.kruskalMST();

    cout << "\nMin Len of cable " << mst_wt;

    return 0;
}
```

OUTPUT

```
/tmp/yJ53k9c1Wm.o
Shortest Distances between Cities
6 - 7
2 - 8
5 - 6
0 - 1
2 - 5
2 - 3
0 - 7
3 - 4
Min Len of cable 37
```