ADA Lab

Assignment - 5

Dated: 24-02-2021

Sub Code: CSE-228

**Vivek Kumar Ahirwar**

**191112419**

**CSE - 3**

Department of

Computer Science and Engineering

Subject Coordinator:

Prof. Manish Pandey



**Maulana Azad**

**National Institute of Technology,**

**BHOPAL – 462 003 (INDIA)**

Contents

[Problem 1: Prim’s Algorithm (For Adjacency Matrix Representation) 2](#_Toc65081203)

[Code 2](#_Toc65081204)

[Output 4](#_Toc65081205)

[Analysis 4](#_Toc65081206)

[Problem 2: Kruskal’s Minimum Spanning Tree Algorithm 5](#_Toc65081207)

[Code 5](#_Toc65081208)

[Output 6](#_Toc65081209)

[Analysis 7](#_Toc65081210)

# Problem 1: Prim’s Algorithm (For Adjacency Matrix Representation)

## Code

#include <bits/stdc++.h>

using **namespace** std;

#define V 6

**class** adjMat

{

**public:**

**int** graph[V][V];

**int** parent[V], key[V];

**bool** mstSet[V];

    adjMat()

    {

        for (**int** i = 0; i < V; i++)

        {

            for (**int** j = 0; j < V; j++)

            {

                graph[i][j] = 0;

            }

        }

        for (**int** i = 0; i < V; i++)

        {

            key[i] = INT\_MAX;

            mstSet[i] = false;

        }

    }

**void** add\_edge(**int** i, **int** j, **int** wt)

    {

        graph[i][j] = wt;

        graph[j][i] = wt;

    }

**int** minKey(**int** key[], **bool** mstSet[])

    {

**int** min = INT\_MAX, min\_index;

        for (**int** v = 0; v < V; v++)

        {

            if (!mstSet[v] && key[v] < min)

                min = key[v], min\_index = v;

        }

        return min\_index;

    }

**void** printMST()

    {

**int** wt=0;

        cout << "Edge       Weight\n";

        for (**int** i = 1; i < V; i++)

        {

            cout << parent[i] << " -- " << i << "       " << graph[i][parent[i]]

                 << "\n";

            wt += graph[i][parent[i]];

        }

        cout<<"Minimum Cost of Spanning Tree: "<<wt<<endl;

    }

**void** prim()

    {

        key[0] = 0;

        parent[0] = -1;

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

        {

**int** u = minKey(key, mstSet);

            mstSet[u] = true;

            for (**int** v = 0; v < V; v++)

            {

                if ((graph[u][v] && mstSet[v] == false) && graph[u][v] < key[v])

                {

                    parent[v] = u;

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

                }

            }

        }

        printMST();

    }

} g;

**int** main()

{

    g.add\_edge(0, 1, 1);

    g.add\_edge(0, 2, 9);

    g.add\_edge(1, 3, 2);

    g.add\_edge(1, 2, 4);

    g.add\_edge(2, 3, 3);

    g.add\_edge(3, 4, 5);

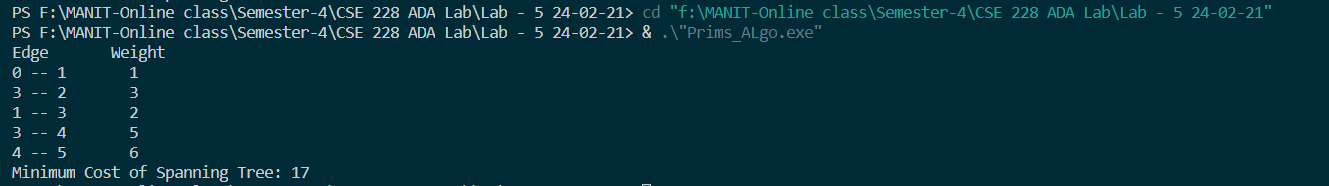
    g.add\_edge(4, 5, 6);

    g.prim();

    return 0;

}

## Output



## Analysis

**Time Complexity:** of the above program is O(V^2). If the input graph is represented using adjacency list, then the time complexity of Prim’s algorithm can be reduced to O(E log V) with the help of binary heap.

# Problem 2: Kruskal’s Minimum Spanning Tree Algorithm

## Code

*//Kruskal’s Minimum Spanning Tree Algorithm*

#include <iostream>

using **namespace** std;

#define I INT\_MAX

**int** edge[9][3] = {{1, 2, 29}, {1, 6, 9}, {2, 3, 16}, {2, 7, 14}, {3, 4, 13}, {4, 5, 21}, {4, 7, 19}, {5, 6, 20}, {5, 7, 26}};

**int** set[8] = {-1, -1, -1, -1, -1, -1, -1, -1};

**int** included[9] = {0, 0, 0, 0, 0, 0, 0, 0, 0};

**void** join(**int** u, **int** v)

{

    if (set[u] < set[v])

    {

        set[u] += set[v];

        set[v] = u;

    }

    else

    {

        set[v] += set[u];

        set[u] = v;

    }

}

**int** find(**int** u)

{

**int** x = u, v = 0;

    while (set[x] > 0)

    {

        x = set[x];

    }

    while (u != x)

    {

        v = set[u];

        set[u] = x;

        u = v;

    }

    return x;

}

**int** t[2][7];

**int** main()

{

**int** u = 0, v = 0, i, j, k = 0, min = INT\_MAX, n = 9;

    i = 0;

    while (i < 6)

    {

        min = INT\_MAX;

        for (j = 0; j < n; j++)

        {

            if (included[j] == 0 && edge[j][2] < min)

            {

                u = edge[j][0];

                v = edge[j][1];

                min = edge[j][2];

                k = j;

            }

        }

        if (find(u) != find(v))

        {

            t[0][i] = u;

            t[1][i] = v;

            join(find(u), find(v));

            included[k] = 1;

            i++;

*// cout<<u<<" "<<v<<" "<<find(u)<<" "<<find(v);*

        }

        else

        {

            included[k] = 1;

        }

    }

    cout << "Spanning Tree\n";

    for (i = 0; i < 6; i++)

    {

        cout << "(" << t[0][i] << ", " << t[1][i] << ")"

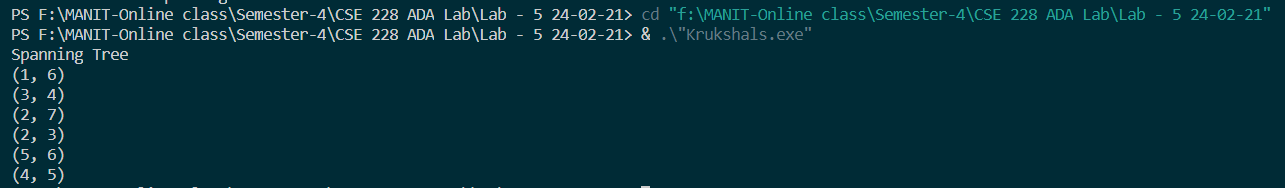
             << "\n";

    }

    return 0;

}

## Output



## Analysis

**Time Complexity:** O(ElogE) or O(ElogV). Sorting of edges takes O(ELogE) time. After sorting, we iterate through all edges and apply find-union algorithm. The find and union operations can take atmost O(LogV) time. So overall complexity is O(ELogE + ELogV) time. The value of E can be atmost O(V2), so O(LogV) are O(LogE) same. Therefore, overall time complexity is O(ElogE) or O(ElogV).