ADA LAB

Assignment – 5

*Sub Code: CSE-228*

*Date: 24/02/2021*

**Ayush Patel**

**191112253**

*CSE-2*

Department of

Computer Science and Engineering

**Under Guidance of**

**Professor: Manish Pandey**



**Maulana Azad**

**National Institute of Technology,**

**BHOPAL – 462 003 (INDIA)**

Contents

[QUES 1: Prim’s Algorithm (For Adjacency Matrix Representation) 2](#_Toc65075147)

[Code 1 2](#_Toc65075148)

[Output 1 3](#_Toc65075149)

[Summary 1 4](#_Toc65075150)

[QUES 2: Kruskal’s Minimum Spanning Tree Algorithm 5](#_Toc65075151)

[Code 2 5](#_Toc65075152)

[Output 2 6](#_Toc65075153)

[Summary 2 7](#_Toc65075154)

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

## Code 1

#include <bits/stdc++.h>

*//Prims Algorithm For Minimum Spanning Tree*

using namespace **std**;

#define **I** **INT\_MAX**

**int** cost[][8] =

    {{**I**, **I**, **I**, **I**, **I**, **I**, **I**, **I**},

     {**I**, **I**, 25, **I**, **I**, **I**, 5, **I**},

     {**I**, 25, **I**, 12, **I**, **I**, **I**, 10},

     {**I**, **I**, 12, **I**, 8, **I**, **I**, **I**},

     {**I**, **I**, **I**, 8, **I**, 16, **I**, 14},

     {**I**, **I**, **I**, **I**, 16, **I**, 20, 18},

     {**I**, 5, **I**, **I**, **I**, 20, **I**, **I**},

     {**I**, **I**, 10, **I**, 14, 18, **I**, **I**}};

**int** near[8] = {**I**, **I**, **I**, **I**, **I**, **I**, **I**, **I**};

**int** t[2][6];

**int main**()

{

**int** i, j, k, u, v, n = 7, min = **I**;

   for (i = 1; i <= n; i++)

   {

      for (j = i; j <= n; j++)

      {

         if (cost[i][j] < min)

         {

            min = cost[i][j];

            u = i;

            v = j;

         }

      }

   }

   t[0][0] = u;

   t[1][0] = v;

   near[u] = near[v] = 0;

   for (i = 1; i <= n; i++)

   {

      if (near[i] != 0)

      {

         if (cost[i][u] < cost[i][v])

            near[i] = u;

         else

            near[i] = v;

      }

   }

   for (i = 1; i < n - 1; i++)

   {

      min = **I**;

      for (j = 1; j <= n; j++)

      {

         if (near[j] != 0 && cost[j][near[j]] < min)

         {

            k = j;

            min = cost[j][near[j]];

         }

      }

      t[0][i] = k;

      t[1][i] = near[k];

      near[k] = 0;

      for (j = 1; j <= n; j++)

      {

         if (near[j] != 0 && cost[j][k] < cost[j][near[j]])

            near[j] = k;

      }

   }

   for (i = 0; i < n - 1; i++)

   {

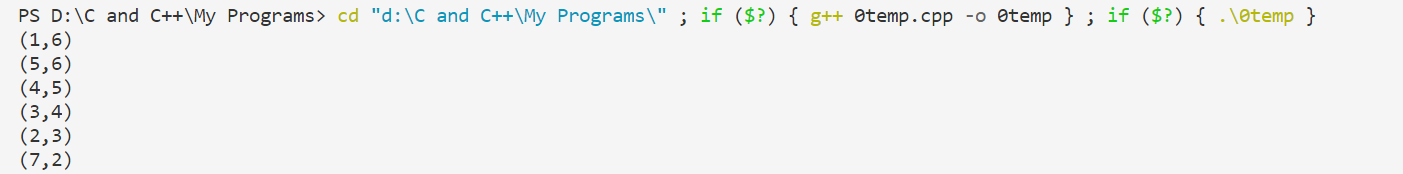
      cout **<<**"("**<<** t[0][i] **<<** "," **<<** t[1][i] **<<**")"**<<** **endl**;

   }

   return 0;

}

## Output 1



## Summary 1

**Time Complexity:** of the above program is O(V^2). If the input [graph is represented using adjacency list](https://www.geeksforgeeks.org/archives/27134), then the time complexity of Prim’s algorithm can be reduced to O(E log V) with the help of binary heap.

# QUES 2: Kruskal’s Minimum Spanning Tree Algorithm

## Code 2

#include <bits/stdc++.h>

*//Kruskal’s Minimum Spanning Tree Algorithm*

using namespace **std**;

#define **ld** long double

#define **I** 65535

int edge[9][3]={{1,2,28},{1,6,10},{2,3,16},{2,7,14},{3,4,12},

{4,5,22},{4,7,18},{5,6,25},{5,7,24}};int sett[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 (sett[u] < sett[v])

   {

      sett[u] += sett[v];

      sett[v] = u;

   }

   else

   {

      sett[v] += sett[u];

      sett[u] = v;

   }

}

int **find**(int u)

{

   int x = u, v = 0;

   while (sett[x] > 0)

   {

      x = sett[x];

   }

   while (u != x)

   {

      v = sett[u];

      sett[u] = x;

      u = v;

   }

   return x;

}

int t[2][7];

int **main**()

{

   int u = 0, v = 0, i, j, k = 0, min = 65535, n = 9;

   i = 0;

   while (i < 6)

   {

      min = 65535;

      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++;

      }

      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 2



## Summary 2

**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).