**Sandesh Ghimire**

**CSCI 301 section 3**

**Computer Science 2**

**Project 10**

**23rd November**

**Introduction**

Kruskal’s Algorithm is one of the algorithms that finds the minimum spanning tree (MST) of an undirected edge-weighted graph. It examines the edges of the graph according to the ascending order of their weight.

**Data Structures:**

The data structures used in this project is Array. We used 1D as well as 2D array to do this program. 2D array is used to make matrix that holds the weight of the graph. We also used file stream to open a .txt file and extract the dates from there.

**Functions:**

The program contains a class named Partition in the header file " *Part.h* “. The class list contains many public functions that are used to manipulate the Kruskal’s algorithm:

*Partition()* : This is the constructor for the Partition class.

*void uf\_find (int x):* It returns the root of the tree to which x belongs.

*void uf\_union (int x, int y):* This function forms the union of the distinct sets containing x and y. It takes two parameter both integer.

*void kruskal(int N, int cst[][100]):* This function performs the kurkal’s algorithm. It takes two parameter one is integer and another is a 2D array.

**The Main Program:**

**The main** function asks the input to the user first. It asks for the number of vertices they want. After that the program reads the file name “vertices.txt”. It takes out the vertex and their weight and store them to a variable. After doing all this things the main function calls the kruskal() function to perform Kruskal algorithm.

**Code**

**Main.cpp**

#include <iostream>

#include "Part.h"

#include"Part.cpp"

#include<iomanip>

#include<cstdlib>

#include <cmath>

#include <fstream>

using namespace std;

int main(){//The main function

int inp, x;

int j, i, data;

Partition p; //declaring object of class

ifstream myfile;

cout<<"Enter the number of vertices you want: ";//asking input to the user

cin>>inp;

int cost[7][100];//matrix

for (i = 0; i<inp; i++) { //starting the loop

for (j = 0; j<inp; j++) {

cost[i][j] = INT\_MAX;

}

}

myfile.open ("vertex.txt");//opening the file

if (!myfile) {//display error message

cout << "Unable to open file";

exit(1); // terminate with error

}

while (!myfile.eof())//starting loop if there is no error

{

myfile >> i;//loading data from txt file

myfile >> j;

myfile >> data;

cost[i][j] = data;//keeping the weight in matrix

cost[j][i] = data;

}

myfile.close();//closing the file

p.kruskal(inp, cost);//calling the function

return 0;

}

**Part.cpp**

#include <iostream>

#include "Part.h"

using namespace std;

Partition::Partition(){ //the constructor

for (int i=0;i<N;++i)

{

p[i]=-1;

}

}

int Partition::uf\_find(int x)//find function

{

while (p[x] != x)//starting loop

x = p[x];//put the value in x

return x;//return x

}

void Partition::uf\_union(int x, int y)//unifing the tree

{

int root= uf\_find(x);//call find function

int new\_child = uf\_find(y);

p[root] = new\_child;//keep value inside array

}

void Partition::kruskal(int N, int cst[][100])//Kruskal algorithm

{

int minWeight = 0;

for (int i = 0; i < N; i++)//initaizing the value

p[i] = i;

int count = 0;

cout << "Minimum spanning tree is:" << endl;

while (count < N - 1)//starting the pre condition

{

int Min = INT\_MAX, value = -1, value2 = -1;

for (int i = 0; i < N; i++)//starting the for loop

{

for (int j = 0; j < N; j++)//nested for loop

{

if (uf\_find(i) != uf\_find(j) && cst[i][j] < Min) { //applying condition to check min value

Min = cst[i][j];//put the main value here

value = i;//edge

value2 = j;//edge

}

}

}

uf\_union(value, value2);//calling union function

count++;

cout << "(v" << value << ",v" << value2 << "); cost = " << Min << endl;//desplaying the message

minWeight += Min;

}

cout<<"Total weight = "<< minWeight;//output the total cost

}

**Part.h**

#include <iostream>

#ifndef PART\_H

#define PART\_H

using namespace std;

class Partition{

public://public functions of the class

static const int N=100;//static constant

Partition();//constructor

int uf\_find(int x);//find function

void uf\_union(int x, int y);//function to unify the tree

void kruskal(int , int[][100]);//the main algorithm

private:

int p[N];//array that contains the values

};

#endif // PART\_H

**User Document**

The program can be used by the user by entering the number of vertices they want.

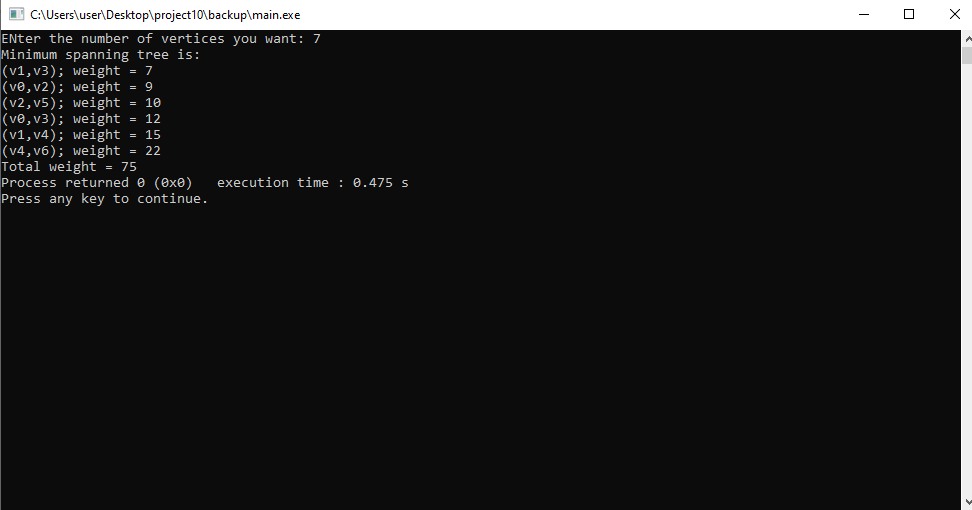
To compile the program simply enter:

*g++ -o main main.cpp*

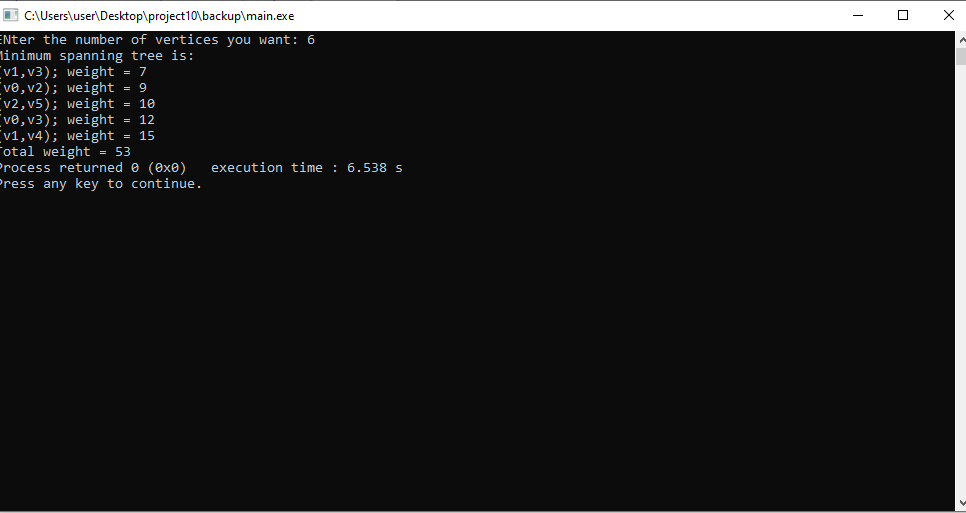
To run the program, enter. */main,* then you can input an integer. After entering an integer, the program will display the MST.

**Testing**

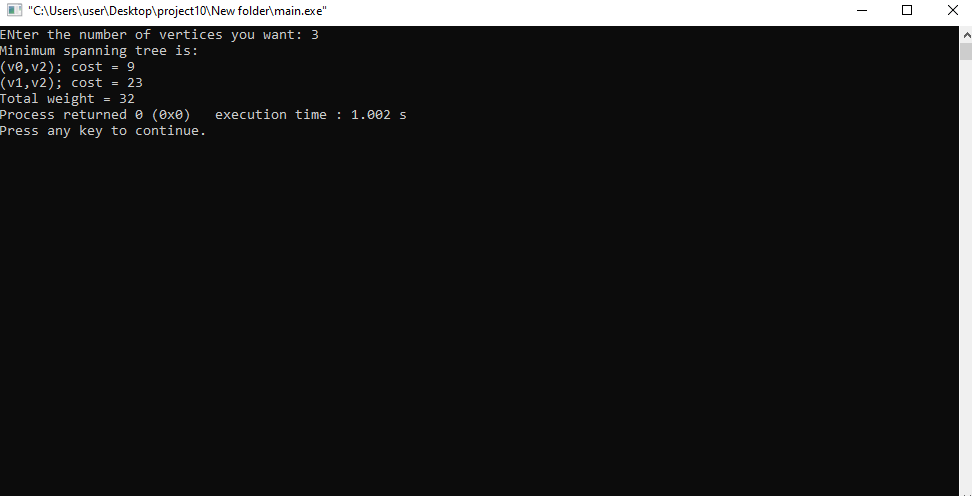
-Test 1



Test 2-



Test 3:



**Summary**

In this project, we have implemented an algorithm that finds the MST of an undirected edge-weighted graph that is called The Kruskal’s Algorithm. It examines the edges of the graph according to the ascending order of their weight. The program reads an input from the user. That determines the number of vertices the tree contain. I used three files to keep my codes organized and easy to access. The problems and their solutions involving Kruskal’s Algorithm that we learnt in class helped a to solve the given question. According to the class there can be worst case while using this program. So, we can make it more efficient if we use weight balancing and path compression technique.

Under what circumstances can we be sure that Kruskal's Algorithm and Prim's Algorithm will identify the same minimum spanning tree?

Yes, because MST is always the same. The way of finding it is different in both algorithms. Kruskal’s algorithm uses the vertices that has minimum weight first to calculate MST where as prime can use any vertices.