Name : Nayanathara P.M.C.

Index No : 210417X

In Class Lab – Week 11 - MST

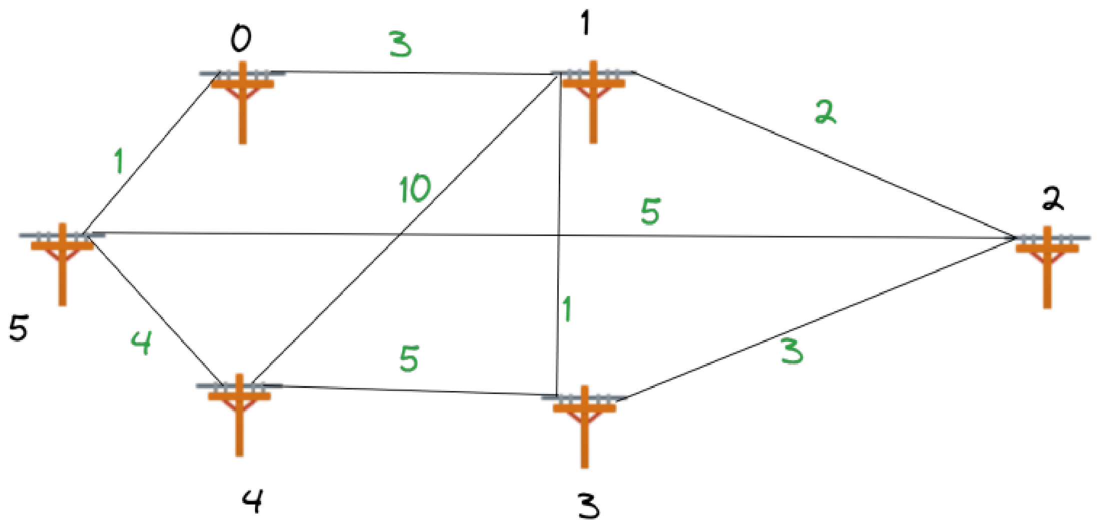
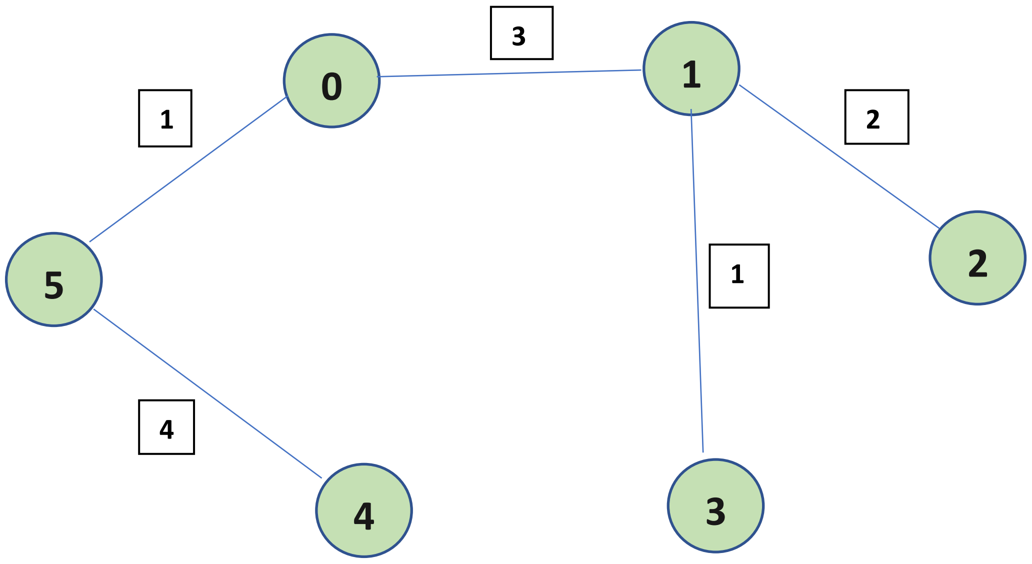


Figure 1: Telephone Pole Graph

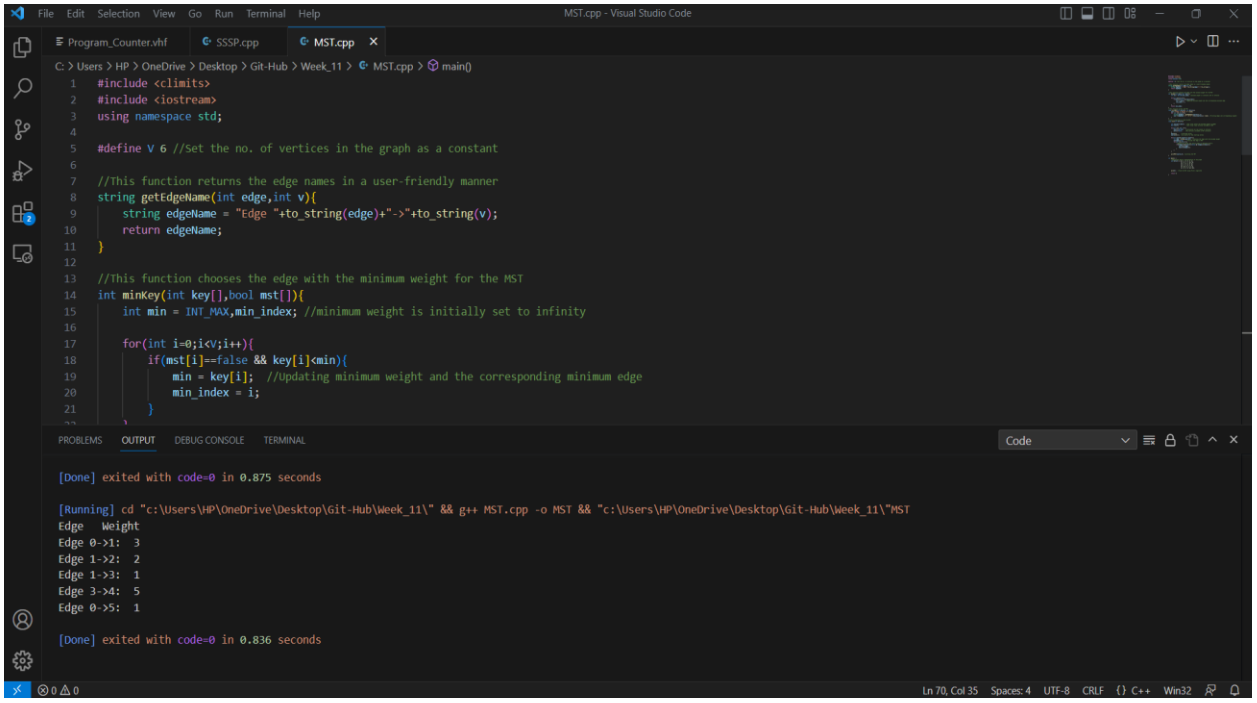
- **Adjacency Matrix**

	0	1	2	3	4	5
0	0	3	0	0	0	1
1	3	0	2	1	10	0
2	0	2	0	3	0	5
3	0	1	3	0	5	0
4	0	10	0	5	0	4
5	1	0	5	0	4	0

- **Minimum Spanning Tree for the graph (Taking Node3 as the start node)**



- **MST using implemented Prim’s algorithm (Taking 0 as the start node)**


File Edit Selection View Go Run Terminal Help
MST.cpp - Visual Studio Code
Program_Counter.vhf
SSSP.cpp
MST.cpp
C: > Users > HP > OneDrive > Desktop > Git-Hub > Week_11 > MST.cpp > main()
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Code
Ln 70, Col 35 Spaces: 4 UTF-8 CRLF {} C++ Win32

```cpp
#include <climits>
#include <iostream>
using namespace std;

#define V 6 //Set the no. of vertices in the graph as a constant

//This function returns the edge names in a user-friendly manner
string getEdgeName(int edge,int v){
    string edgeName = "Edge "+to_string(edge)+"->"+to_string(v);
    return edgeName;
}

//This function chooses the edge with the minimum weight for the MST
int minKey(int key[],bool mst[]){
    int min = INT_MAX,min_index; //minimum weight is initially set to infinity

    for(int i=0;i<V;i++){
        if(mst[i]==false && key[i]<min){
            min = key[i];  //Updating minimum weight and the corresponding minimum edge
            min_index = i;
        }
```

```
[Done] exited with code=0 in 0.875 seconds

[Running] cd "c:\Users\HP\OneDrive\Desktop\Git-Hub\Week_11\" && g++ MST.cpp -o MST && "c:\Users\HP\OneDrive\Desktop\Git-Hub\Week_11\"MST
Edge   Weight
Edge 0->1:  3
Edge 1->2:  2
Edge 1->3:  1
Edge 3->4:  5
Edge 0->5:  1

[Done] exited with code=0 in 0.836 seconds
```

MST s obtained in Question 2 and Question 3 are not the same.

A graph has only one minimum spanning tree if each edge has a distinct weight. This is because if two edges have the same weight, then they can be swapped in and out of the MST without changing the weight of the tree. As a result, there would be multiple MSTs, each with a different set of edges.

➢ Time complexity between Prim's algorithm and Kruskal's algorithm

1. Prim's algorithm –
   - Time complexity : **O(V^2) or O(V log V)** depending on the implementation.
   - In the standard implementation of Prim's algorithm using an adjacency matrix, the time complexity is O(V^2), where V is the number of vertices in the graph. Prim's algorithm iterates V times, and in each iteration, it performs a key update and extract-min operation, which takes O(V) time.
   - However, using a priority queue to efficiently extract the minimum key can reduce the time complexity to O(V log V). in the basic implementation. The extract-min operation can be done in O(log V) time, resulting in an overall time complexity of O(V log V).
   - More efficient for dense graphs.

2. Kruskal's algorithm –
   - Time complexity : **O(V log E) or O(E log V)**
   - E- No. of edges , V- No. of vertices
   - This algorithm sorts the edges based in their weights and then processes them in ascending order.
   - The sorting takes O(E log E) when efficient algorithms like quicksort or merge sort are used. After sorting, the algorithm performs a disjoint set of union-find operation to determine whether an edge creates a cycle in the MST. This operation takes O(log V) time, and is performed on each of the E edges.
   - Thus, the overall time complexity is O(E log E) OR O(E log V)
   - More efficient for sparse graphs.

➢ Git-Hub Repository link

https://github.com/ChethmiNayanathara/Data-Structures-and-Algorithms