G. H. Raisoni College Of Engineering And Management, Wagholi Pune

2021- 2022

Assignment no :- 10

Department	CE [SUMMER 2022 (Online)]		
Term / Section	III/B	Date Of **submission**	13-12-2021
Subject Name /Code	Data Structures and Algorithms/ UCSL201/UCSP201		
Roll No.	SCOB77	Name	Pratham Rajkumar pitty
Registration Number	2020AC0E1100107		

Experiment No 10

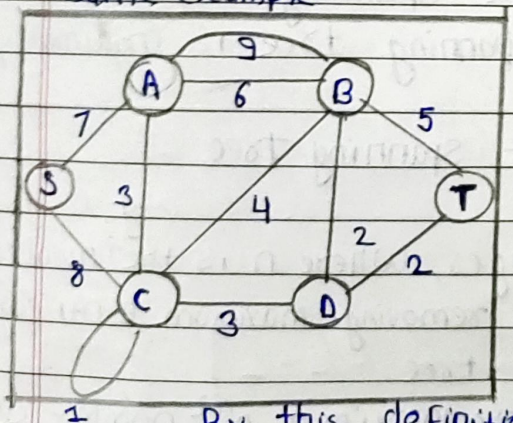
 # Aim:- You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. Solve the problem by suggesting appropriate data structures

 # Theory:-

Prim's algorithm to find minimum cost Spanning tree (as Kruskal's algorithm) uses the greedy approach. Prim's algorithm shares a similarity with the Shortest path first algorithms.

Prim's algorithm, in contrast with Kruskal's algorithm, treats the nodes as a single tree and keeps on adding new nodes to the spanning tree from the given graph.

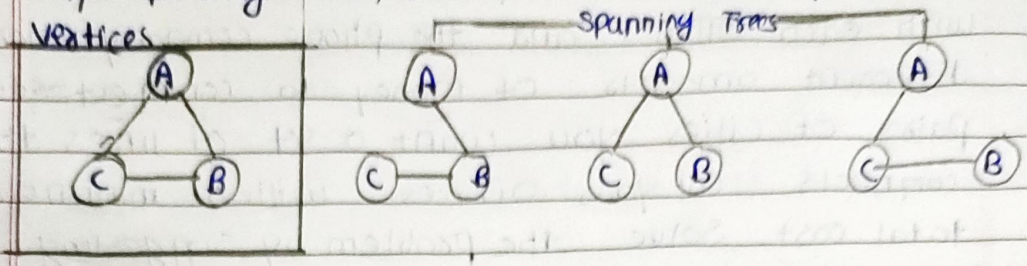
To contrast with kruskal's algorithm and to understand prim's algorithm better, we shall use the same example.



A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected.

By this definition, we can draw a conclusion

that every connected and undirected graph G has at least one Spanning tree. A disconnected graph does not have any Spanning tree, as it cannot be spanned to all its vertices.



We found three spanning trees off one complete graph. A complete undirected graph can have maximum n^{n-2} number of spanning trees, where n is the number of nodes. In the above addressed example, n is 3, hence $3^{3-2}=3$ spanning tree are possible.

General properties of spanning tree

- A connected graph G can have more than one spanning tree.
- All possible spanning trees of graph G, have the same number of edges and vertices.
- The spanning tree does not have any cycle (loops).
- Removing one edge from the spanning tree will make the graph disconnected. i.e the spanning tree is minimally connected.
- Adding one edge to the spanning tree will create a circuit or loop. i.e the spanning tree is maximally acyclic.

Mathematical properties of spanning tree

- Spanning tree has n-1 edges, where n is the number of nodes.
- From a complete graph, by removing maximum e-n+1 edges, we can construct a spanning tree.
- A complete graph can have maximum n^{n-2} number of spanning trees.

Application of Spanning Tree

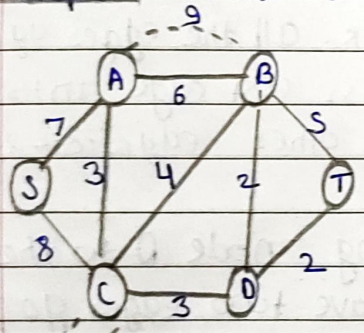
- Civil Network Planning
- Computer Network Routing protocol
- Cluster Analysis

Minimum Spanning tree (MST)

In a weighted graph, a minimum spanning tree is a spanning tree that has minimum weight that all other spanning trees of the same graph. In real-world situations, this weight can be measured as distance, congestion, traffic load or any arbitrary value.

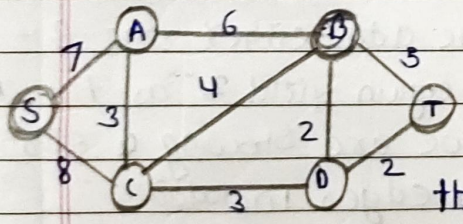
- MST Algorithm :→
  - Kruskal's Algorithm
  - Prim's Algorithm

Step 1 :→ Remove all loops and parallel edges



Remove all loops and parallel edges from the given graph. In case of parallel edges, keep the one which has the least cost associated and remove all others

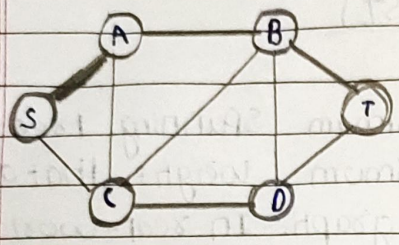
Step 2 :→ Choose any arbitrary node as root node.



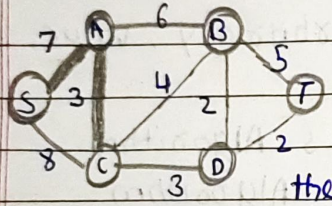
In this case, we chose S node as the root node of Prim's spanning tree. This node is arbitrarily chosen, so any node can be the root node. One may wonder why any vertex can be a root node. So the answer is, in the spanning tree all the

nodes of a graph are included and because it is connected then there must be at least one edge which will join it to the rest of the tree.

Step 3- check outgoing edges and select the one with less cost.

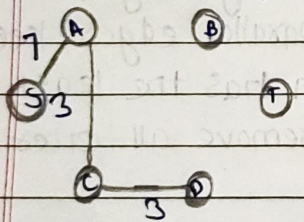


After choosing the root node S, we see that S,A and S,C are two edges with weight 7 and 8, respectively. We choose the edge S,A as it is lesser than the other.

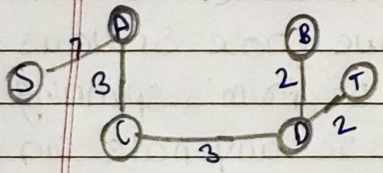


Now, the tree S-7-A is treated as one node and we check for all edges going out from it. We select the one which has the lowest cost and include it in the tree.

After this step, S-7-A-3-C tree is formed. Now we'll again treat it as a node and will check all the edges again. However, we will chose only the least cost edge. In this case, C-3-D is the new edge, which is less than other edges' cost 8, 6, 4 etc.



After adding node D to the spanning tree, we now have two edges going out of it having the same cost, i.e D-2-T and D-2-B. Thus, we add either one, But the next step will again yield 2 as the least cost. Hence, we are showing a spanning tree with both edges included.



MST = 7 + 3 + 3 + 2 + 2
MST = 17

Program code :-

```
#include <iostream>
#include <iomanip>
using namespace std;

const int MAX = 10;
class EdgeList; // forward declaration
class Edge      // USED IN KRUSKAL
{
   int u, v, w;

public:
   Edge() {} // Empty Constructor
   Edge(int a, int b, int weight)
   {
      u = a;
      v = b;
      w = weight;
   }
   friend class EdgeList;
   friend class PhoneGraph;
};
//---- EdgeList Class ----------
class EdgeList
{
   Edge data[MAX];
   int n;

public:
   friend class PhoneGraph;
```

```
    EdgeList()
    {
        n = 0;
    }
    void sort();
    void print();
};
//----Bubble Sort for sorting edges in increasing weights' order ---//
void EdgeList::sort()
{
    Edge temp;
    for (int i = 1; i < n; i++)
        for (int j = 0; j < n - 1; j++)
            if (data[j].w > data[j + 1].w)
            {
                temp = data[j];
                data[j] = data[j + 1];
                data[j + 1] = temp;
            }
}
void EdgeList::print()
{
    int cost = 0;
    for (int i = 0; i < n; i++)
    {
        cout << "\n"
             << i + 1 << " " << data[i].u << "--" << data[i].v << " = " << data[i].w;
        cost = cost + data[i].w;
    }
    cout << "\nMinimum cost of Telephone Graph = " << cost;
}
```

```
//------------ Phone Graph Class----------------
class PhoneGraph
{
   int data[MAX][MAX];
   int n;

public:
   PhoneGraph(int num)
   {
      n = num;
   }
   void readgraph();
   void printGraph();
   int mincost(int cost[], bool visited[]);
   int prim();
   void kruskal(EdgeList &spanlist);
   int find(int belongs[], int vertexno);
   void unionComp(int belongs[], int c1, int c2);
};
void PhoneGraph::readgraph()
{
   cout << "Enter Adjacency(Cost) Matrix : \n";
   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < n; j++)
         cin >> data[i][j];
   }
}
void PhoneGraph::printGraph()
{
   cout << "\nAdjacency (COST) Matrix : \n";
```

```
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << setw(3) << data[i][j];
        }
        cout << endl;
    }
}
int PhoneGraph::mincost(int cost[], bool visited[]) // finding vertex with minimum cost
{
    int min = 9999, min_index; // initialize min to MAX value(ANY) as temporary
    for (int i = 0; i < n; i++)
    {
        if (visited[i] == 0 && cost[i] < min)
        {
            min = cost[i];
            min_index = i;
        }
    }
    return min_index; // return index of vertex which is not visited and having minimum cost
}
int PhoneGraph::prim()
{
    bool visited[MAX];
    int parents[MAX];
    int cost[MAX]; // saving minimum cost
    for (int i = 0; i < n; i++)
    {
        cost[i] = 9999; // set cost as infinity/MAX_VALUE
        visited[i] = 0; // initialize visited array to false
```

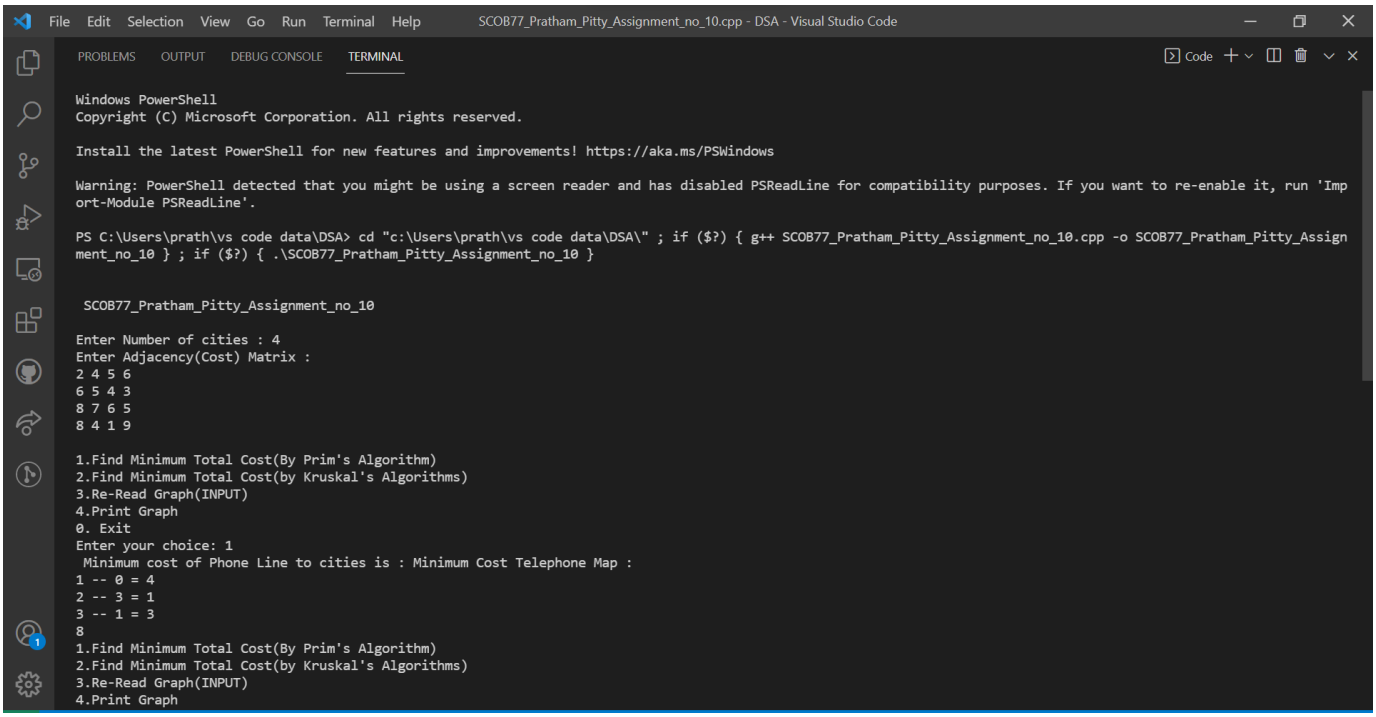
```
    }
    cost[0] = 0;     // starting vertex cost
    parents[0] = -1; // make first vertex as a root
    for (int i = 0; i < n - 1; i++)
    {
        int k = mincost(cost, visited);
        visited[k] = 1;

        for (int j = 0; j < n; j++)
        {
            if (data[k][j] && visited[j] == 0 && data[k][j] < cost[j])
            {
                parents[j] = k;
                cost[j] = data[k][j];
            }
        }
    }
    cout << "Minimum Cost Telephone Map : \n";
    for (int i = 1; i < n; i++)
    {
        cout << i << " -- " << parents[i] << " = " << cost[i] << endl;
    }
    int mincost = 0;
    for (int i = 1; i < n; i++)
        mincost += cost[i]; // data[i][parents[i]];
    return mincost;
}
//------- Kruskal's Algorithm
void PhoneGraph::kruskal(EdgeList &spanlist)
{
    int belongs[MAX]; // Separate Components at start (No Edges, Only vertices)
```

```
   int cno1, cno2;   // Component 1 & 2
   EdgeList elist;
   for (int i = 1; i < n; i++)
      for (int j = 0; j < i; j++)
      {
         if (data[i][j] != 0)
         {
            elist.data[elist.n] = Edge(i, j, data[i][j]); // constructor for initializing edge
            elist.n++;
         }
      }
   elist.sort(); // sorting in increasing weight order
   for (int i = 0; i < n; i++)
      belongs[i] = i;

   for (int i = 0; i < elist.n; i++)
   {
      cno1 = find(belongs, elist.data[i].u); // find set of u
      cno2 = find(belongs, elist.data[i].v); ////find set of v
      if (cno1 != cno2)                  // if u & v belongs to different sets
      {
         spanlist.data[spanlist.n] = elist.data[i]; // ADD Edge to spanlist
         spanlist.n = spanlist.n + 1;
         unionComp(belongs, cno1, cno2); // ADD both components to same set
      }
   }
}
void PhoneGraph::unionComp(int belongs[], int c1, int c2)
{
   for (int i = 0; i < n; i++)
   {
```

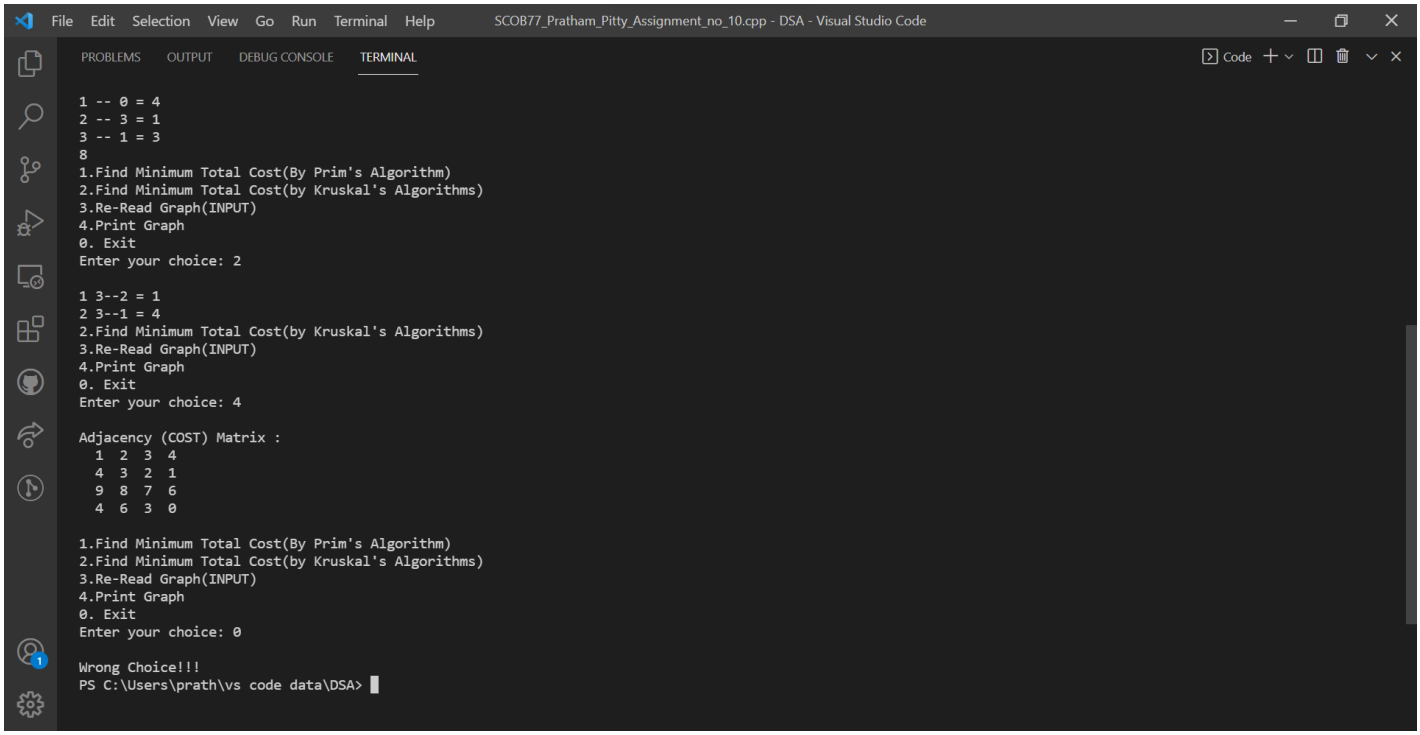
```
        if (belongs[i] == c2)
            belongs[i] = c1;
    }
}
int PhoneGraph::find(int belongs[], int vertexno)
{
    return belongs[vertexno];
}

//--------- MAIN PROGRAM----------------------------------
int main()
{
    cout << "\n\n SCOB77_Pratham_Pitty_Assignment_no_10 \n\n";

    int vertices, choice;
    EdgeList spantree;
    cout << "Enter Number of cities : ";
    cin >> vertices;
    PhoneGraph p1(vertices);
    p1.readgraph();
    do
    {
        cout << "\n1.Find Minimum Total Cost(By Prim's Algorithm)"
            << "\n2.Find Minimum Total Cost(by Kruskal's Algorithms)"
            << "\n3.Re-Read Graph(INPUT)"
            << "\n4.Print Graph"
            << "\n0. Exit"
            << "\nEnter your choice: ";
        cin >> choice;
        switch (choice)
        {
```

```
        case 1:
            cout << " Minimum cost of Phone Line to cities is : " << p1.prim();
            break;
        case 2:
            p1.kruskal(spantree);
            spantree.print();
            break;
        case 3:
            p1.readgraph();
            break;
        case 4:
            p1.printGraph();
            break;
        default:
            cout << "\nWrong Choice!!!";
        }
    } while (choice != 0);

    return 0;
}
```

Output :-



```
File  Edit  Selection  View  Go  Run  Terminal  Help   SCOB77_Pratham_Pitty_Assignment_no_10.cpp - DSA - Visual Studio Code

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                                                    Code

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

Warning: PowerShell detected that you might be using a screen reader and has disabled PSReadLine for compatibility purposes. If you want to re-enable it, run 'Imp
ort-Module PSReadLine'.

PS C:\Users\prath\vs code data\DSA> cd "c:\Users\prath\vs code data\DSA\" ; if ($?) { g++ SCOB77_Pratham_Pitty_Assignment_no_10.cpp -o SCOB77_Pratham_Pitty_Assign
ment_no_10 } ; if ($?) { .\SCOB77_Pratham_Pitty_Assignment_no_10 }


 SCOB77_Pratham_Pitty_Assignment_no_10

Enter Number of cities : 4
Enter Adjacency(Cost) Matrix :
2 4 5 6
6 5 4 3
8 7 6 5
8 4 1 9

1.Find Minimum Total Cost(By Prim's Algorithm)
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
0. Exit
Enter your choice: 1
 Minimum cost of Phone Line to cities is : Minimum Cost Telephone Map :
1 -- 0 = 4
2 -- 3 = 1
3 -- 1 = 3
8
1.Find Minimum Total Cost(By Prim's Algorithm)
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
```



```
File  Edit  Selection  View  Go  Run  Terminal  Help   SCOB77_Pratham_Pitty_Assignment_no_10.cpp - DSA - Visual Studio Code

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                                                    Code

1 -- 0 = 4
2 -- 3 = 1
3 -- 1 = 3
8
1.Find Minimum Total Cost(By Prim's Algorithm)
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
0. Exit
Enter your choice: 2

1 3--2 = 1
2 3--1 = 4
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
0. Exit
Enter your choice: 4

Adjacency (COST) Matrix :
  1  2  3  4
  4  3  2  1
  9  8  7  6
  4  6  3  0

1.Find Minimum Total Cost(By Prim's Algorithm)
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
0. Exit
Enter your choice: 0

Wrong Choice!!!
PS C:\Users\prath\vs code data\DSA>
```