**PUNE INSTITUTE OF COMPUTER TECHNOLOGY, PUNE - 411043 Department of Computer Engineering** 

**S.No.-27, Pune Satara Road, Dhankawadi, Pune-411043**

**Data Structures and Algorithms Laboratory**

**Batch-IV (H4)**

**Name: Rushikesh Vidye**

**Roll No.: 21471**

**Class: SE4**

**Assignment No. 7**

**Title:** 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.

**Software Requirement:**

a) OS : Microsoft Windows 10.

b) Browser: Google Chrome.

c) VS Code.

**Hardware Requirement:**

a) Processor: Intel Core i5-8265U.

b) Ram: 8 GB DDR4 2800Mhz.

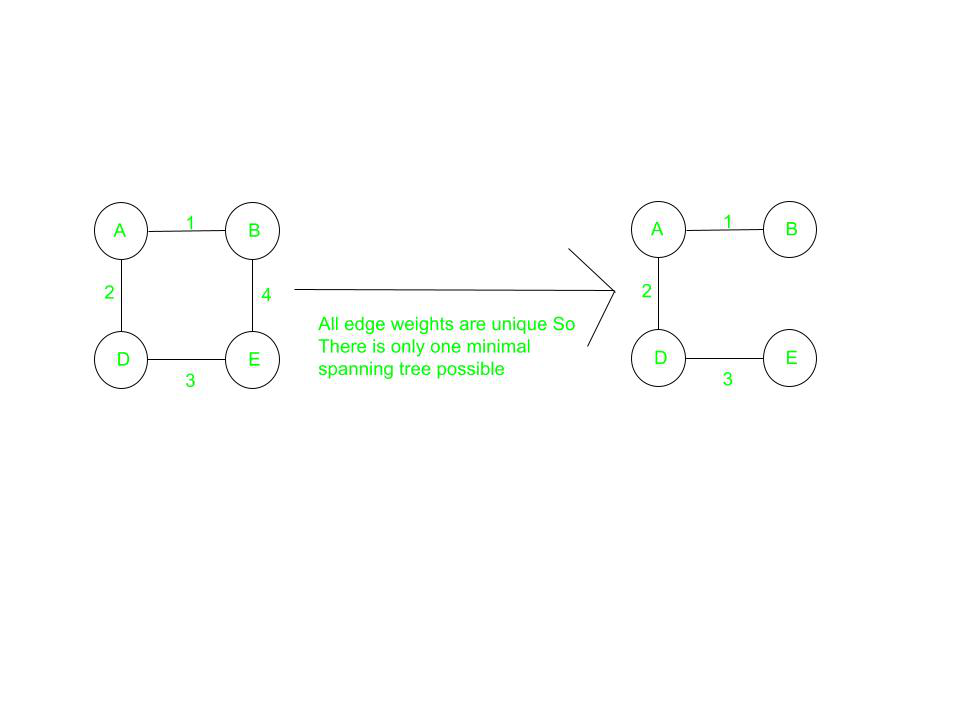
**Theory:**

**Minimum Spanning Tree:**

For a connected and undirected graph, a spanning tree of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have multiple spanning trees. A Minimum Spanning Tree (MST) or minimum weight spanning tree for a weighted, connected, undirected graph is a spanning tree having a weight less than or equal to the weight of every other possible spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

If G(V, E) is a graph then every spanning tree of graph G consists of (V – 1) edges, where V is the number of vertices in the graph and E is the number of edges in the graph. So, (E – V + 1) edges are not a part of the spanning tree. There may be several minimum spanning trees of the same weight. If all the edge weights of a graph are the same, then every spanning tree of that graph is minimum.

For example:



Minimum Cost Edge: If the minimum cost edge of a graph is unique, then this edge is included in any MST. For example, in the above figure, the edge AB (of the least weight) is always included in MST.

If a new edge is added to the spanning tree then it will become cyclic because every spanning tree is minimally acyclic. In the above figure, if edge AD or BC is added to the resultant MST, then it will form a cycle.

The spanning tree is minimally connected, i.e., if any edge is removed from the spanning tree it will disconnect the graph. In the above figure, if any edge is removed from the resultant MST, then it will disconnect the graph.

**Prim’s Algorithm:**

The idea behind Prim’s algorithm is simple, a spanning tree means all vertices must be connected. So, the two disjoint subsets (discussed above) of vertices must be connected to make a Spanning Tree. And they must be connected with the minimum weight edge to make it a Minimum Spanning Tree.

**Algorithm:**

Create Graph: int V, Int E :

1. Accept number of Vertex and Edges.
2. Initialize V\*V AdjMatrix with 999.

Add Edge: Int Start, Int E, Int Weight:

1. Accept Start and E (Considering Bidirectional Edge)
2. Set element at AdjMatix[start][e] = Weight
3. Set element at AdjMatrix[e][start] = Weight

Minimum :

* 1. Create a set mstSet that keeps track of vertices already included in MST.
  2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
  3. While mstSet doesn’t include all vertices
     1. Pick a vertex u which is not there in mstSet and has minimum key value.
     2. Include u to mstSet.
     3. Update key value of all adjacent vertices of u. To update the key values, iterate through all adjacent vertices. For every adjacent vertex v, if weight of edge u-v is less than the previous key value of v, update the key value as weight of u-v

The idea of using key values is to pick the minimum weight edge from cut. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.

**Time Complexity:**

|  |  |  |
| --- | --- | --- |
| Sr.No | Methods | Complexity |
| 1 | Create() | O(n^2) |
| 2 | AddEdge() | O(1) |
| 3 | Minimum() | O(n) |

**Conclusion:**

Hence we learnt the implementation of BFS and DFS traversing techniques with Graph using adjacent matrix and adjacent list.

**Code:**

|  |
| --- |
| #pragma once  #include<iostream>  #include<vector>  using namespace std;  class SpaningTree{      int v,e,w;      int \*\*adjMatrix;      int \* visited;      public :      SpaningTree(int v,int e);      void addEdge(int start,int e,int weight);      void minimumTree();      void printAdjMatrix();};  SpaningTree::SpaningTree(int v,int e){      this->v = v;      this->e = e;      adjMatrix = new int\*[v];      visited = new int[v];      for(int row = 0 ; row < v; row++){          visited[row] = 0;          adjMatrix[row] = new int[v];          for(int column = 0;column < v; column++)          {adjMatrix[row][column] = 999; }}}  void SpaningTree::addEdge(int start,int e,int weight){      adjMatrix[start][e] = weight;      adjMatrix[e][start] = weight;}  void SpaningTree::printAdjMatrix(){       for(int row = 0 ; row < v; row++){          for(int column = 0;column < v; column++){              cout<<adjMatrix[row][column]<<"\t";  }          cout<<endl;}}  void SpaningTree::minimumTree(){      int p=0,q=0,total=0,min;      visited[0] = 1;      for(int count=0;count<(v-1);count++){          min=999;          for(int i=0;i<v;i++){              if(visited[i]==1) {                  for(int j=0;j<v;j++){                      if(visited[j]!=1) {                          if(min > adjMatrix[i][j]) {                              min=adjMatrix[i][j];                              p=i;                              q=j;                          }}}}}          visited[p]=1;          visited[q]=1;          total=total+min;          cout<<"Minimum cost connection is "<<(p)<<" -> "<<(q)<<"  with charge : "<<min<< endl;}      cout<<"The minimum total cost of connections of all branches is: "<<total<<endl;}  #include"header.h"  int main(void)  {      SpaningTree st = SpaningTree(5,7);      st.printAdjMatrix();      st.addEdge(0,1,200);      st.addEdge(0,3,600);      st.addEdge(1,2,300);      st.addEdge(1,4,500);      st.addEdge(2,4,700);      st.addEdge(3,4,900);      cout<<endl<<endl;      st.printAdjMatrix();      st.minimumTree();  } |

**Output:**

|  |
| --- |
| 999 999 999 999 999  999 999 999 999 999  999 999 999 999 999  999 999 999 999 999  999 999 999 999 999  999 200 999 600 999  200 999 300 999 500  999 300 999 999 700  600 999 999 999 900  999 500 700 900 999  Minimum cost connection is 0 -> 1 with charge : 200  Minimum cost connection is 1 -> 2 with charge : 300  Minimum cost connection is 1 -> 4 with charge : 500  Minimum cost connection is 0 -> 3 with charge : 600  The minimum total cost of connections of all branches is: 1600 |