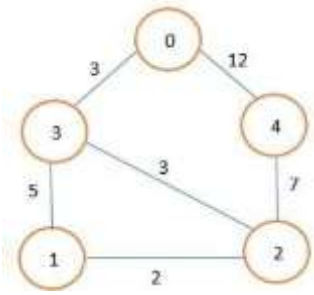
EX.NO : 11(A)	PRIM'S MINIMUM SPANNING TREE
DATE :	

PROGRAM STATEMENT:

To find the sum of weights of the edges of the M inimum Spanning Tree



ALGORITHM:

1. Start the program.
2. Initialize a priority queue to store edges, each represented by (weight, vertex).
3. Choose an arbitrary starting vertex and mark it as visited.
4. Add all edges connected to the starting vertex into the priority queue.
5. While the priority queue is not empty, pop the edge with the minimum weight and if the destination vertex is not in the MST, add it and update the priority queue with edges from the newly visited vertex.
6. End the program by calculating the sum of the edge weights in the MST.

PROGRAM:

```
void Graph::addEdge(int u, int v, int w)
{
    adj[u].push_back(make_pair(v, w));
    adj[v].push_back(make_pair(u, w));
}
```

OUTPUT :

	Input	Expected	Got	
✔	4 5 0 1 5 1 3 15 1 2 10 0 2 8 2 3 20	Prim's MST edges are: 0 - 1 0 - 2 1 - 3 MST cost = 28	Prim's MST edges are: 0 - 1 0 - 2 1 - 3 MST cost = 28	✔
✔	7 9 0 1 28 1 2 16 2 3 12 3 6 18 1 6 14 6 4 24 3 4 22 4 5 25 5 0 10	Prim's MST edges are: 2 - 1 3 - 2 4 - 3 5 - 4 0 - 5 1 - 6 MST cost = 99	Prim's MST edges are: 2 - 1 3 - 2 4 - 3 5 - 4 0 - 5 1 - 6 MST cost = 99	✔
✔	5 6 0 3 3 0 4 12 3 2 3 4 2 7 2 1 2 3 1 5	Prim's MST edges are: 2 - 1 3 - 2 0 - 3 2 - 4 MST cost = 15	Prim's MST edges are: 2 - 1 3 - 2 0 - 3 2 - 4 MST cost = 15	✔

Passed all tests! ✔

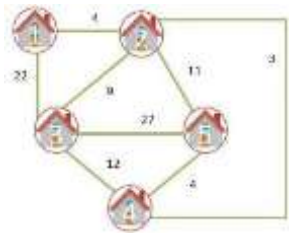
RESULT:

Thus, the C++ program to find the sum of weights of the edges of the Minimum Spanning Tree is created successfully.

EX.NO : 11(B)	KRUSKAL'S MINIMUM SPANNING TREE
DATE :	

PROGRAM STATEMENT:

Given a houses of a city consisting of N 2D coordinates {x, y} where each coordinate represents the location of each house, the task is to find the minimum cost to connect all the houses of the city.



ALGORITHM:

1. Start the program.
2. For each pair of houses, compute the Euclidean distance between them and store it as an edge with weight (distance).
3. Use Kruskal's algorithm or Prim's algorithm to find the Minimum Spanning Tree (MST).
4. Sort the edges by their weight (increasing order).
5. Use a Union-Find (Disjoint-Set) data structure to detect and avoid cycles while adding edges to the MST.
6. The sum of the selected edges' weights in the MST will give the minimum cost to connect all houses.
7. End the program.

PROGRAM:

```
voidaddEdge(int x, int y, int w)
    {
    edgelist.push_back({ w, x, y});
    }
```

OUTPUT :

	Input	Expected	Got	
✔	4 5 0 1 10 1 3 15 2 3 4 2 0 6 0 3 5	2 3 4 0 3 5 0 1 10 Minimum Cost Spanning Tree: 19	2 3 4 0 3 5 0 1 10 Minimum Cost Spanning Tree: 19	✔
✔	3 3 1 2 5 1 3 6 3 2 1	3 2 1 1 2 5 Minimum Cost Spanning Tree: 6	3 2 1 1 2 5 Minimum Cost Spanning Tree: 6	✔
✔	5 8 1 2 4 1 3 22 2 5 11 2 3 9 3 5 27 4 5 4 3 4 12 2 4 3	2 4 3 1 2 4 4 5 4 2 3 9 Minimum Cost Spanning Tree: 20	2 4 3 1 2 4 4 5 4 2 3 9 Minimum Cost Spanning Tree: 20	✔

Passed all tests! ✔

RESULT:

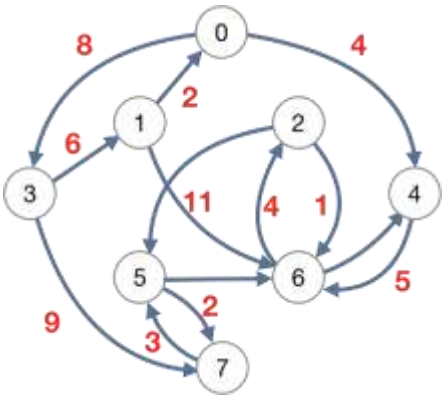
Thus, the C++ program to find the minimum cost to connect all the houses of the city Tree is created successfully.

EX.NO : 11(C)	SHORTEST PATH - DIJKSTRA'S ALGORITHM
DATE :	

PROGRAM STATEMENT:

There is a directed weighted connected graph. You are given a positive integer n which denotes that the graph has n nodes labeled from 1 to n, and an array edges where each edges[i] = [ui, vi, weighti] denotes that there is an edge between nodes ui and vi with weight equal to weighti.

Correct the given C++ function to find the dijkstra'a shortest path from every node to all other nodes in the given graph.



ALGORITHM:

1. Start the program.
2. Initialize the distance for the source vertex to 0 and all other vertices to infinity.
3. Insert the source vertex into a set with its distance as the key.
4. While the set is not empty:
    a. Extract the vertex with the minimum distance from the set.
    b. For each neighboring vertex, check if the current path offers a shorter distance than previously known. If so, update the distance and insert it back into the set.
5. After processing all vertices, print the shortest distance from the source to each vertex.
6. End the program.

PROGRAM:

```
void Graph::shortestPath(int src)
{
set< pair<int, int> > setds;
vector<int> dist(V, INF);
setds.insert(make_pair(0, src));
dist[src] = 0;
while(!setds.empty())
      {
      pair<int, int> tmp =*(setds.begin());
                setds.erase(setds.begin());
                int u=tmp.second;
                list< pair<int, int> >::iterator i;
                for (i= adj[u].begin(); i != adj[u].end(); ++i)
                {
                        int v= (*i).first;
                        intweight =(*i).second; if(dist[v] >dist[u] + weight)
                          {
                                  if(dist[v] != INF)
                                  setds.erase(setds.find(make_pair(dist[v], v)));
                                 dist[v] = dist[u] + weight; setds.insert(make_pair(dist[v], v));
                          }
                }
      }
    dist[]
    printf("Vertex Distance from Source\n");
    for (int i= 0; i< V; ++i)
    printf("%d %d\n", i, dist[i]);
    }
```

OUTPUT :

	Input	Expected	Got	
✔	9 14 0 1 4 0 7 8 1 2 8 1 7 11 2 3 7 2 8 2 2 5 4 3 4 9 3 5 14 4 5 10 5 6 2 6 7 1 6 8 6 7 8 7	Vertex Distance from Source 0 0 1 4 2 12 3 19 4 21 5 11 6 9 7 8 8 14	Vertex Distance from Source 0 0 1 4 2 12 3 19 4 21 5 11 6 9 7 8 8 14	✔
✔	5 5 0 1 5 0 2 2 2 3 1 0 3 6 2 4 5	Vertex Distance from Source 0 0 1 5 2 2 3 3 4 7	Vertex Distance from Source 0 0 1 5 2 2 3 3 4 7	✔
✔	4 4 0 1 100 2 0 100 1 3 600 2 3 200	Vertex Distance from Source 0 0 1 100 2 100 3 300	Vertex Distance from Source 0 0 1 100 2 100 3 300	✔
✔	5 6 0 1 2 0 4 8 1 4 2 1 2 3 2 3 1 3 4 1	Vertex Distance from Source 0 0 1 2 2 5 3 5 4 4	Vertex Distance from Source 0 0 1 2 2 5 3 5 4 4	✔
✔	8 14 0 1 2 0 4 4 4 6 5 2 6 1 6 2 4 2 5 11 1 6 11 6 4 5 5 7 2 7 5 3 3 7 9 0 3 8 1 0 2 3 1 6	Vertex Distance from Source 0 0 1 2 2 10 3 8 4 4 5 19 6 9 7 17	Vertex Distance from Source 0 0 1 2 2 10 3 8 4 4 5 19 6 9 7 17	✔

Passed all tests! ✔

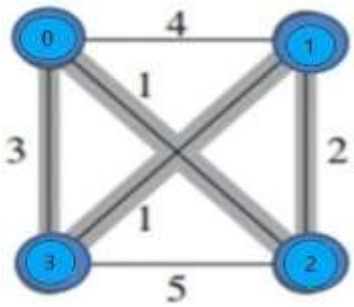
RESULT:

Thus, the C++ program to find the dijkstra'a shortest path from every node to all other nodes in the given graph is created successfully.

EX.NO : 11(D)	APPLICATION OF GRAPH
DATE :	

PROGRAM STATEMENT:

A traveller needs to visit all cities from a list, where distances between all cities are known and visited once. What is the shortest possible routes that he visits each city exactly once and returns to the origin city?. Find the TSP solution using CPP function.



ALGORITHM:

1. Start the program.
2. Initialize a list `ver` containing all vertices except the starting vertex `p`.
3. Set `m_p` to infinity to store the minimum path weight.
4. Generate all permutations of the vertices in `ver`:
    a. For each permutation, calculate the total path weight by traversing through the vertices in the order of the permutation.
    b. Add the weight to complete the cycle by returning to the starting vertex `p`.
5. Update `m_p` with the minimum path weight found.
6. After checking all permutations, return the minimum path weight.

PROGRAM:

```
int TSP(int p)
  {
  vector<int> ver;
  for(int i= 0; i<V; i++)
  {
  if(i!=p)
      ver.push_back(i);
  }
  int m_p=INT_MAX;
  do {
```

```
    int  cur_pth=  0;
    int k = p;
    for(int i=0;i<(int)ver.size();i++)
    {
     cur_pth += adjMatrix[k][ver[i]];
    k = ver[i];
    }
    cur_pth+= adjMatrix[k][p];
    m_p = min(m_p, cur_pth);
  }
  while (next_permutation(ver.begin(), ver.end()));
  return m_p;
}
```

OUTPUT:

	Input	Expected	Got	
✔	4 6 0 1 10 1 2 35 2 0 15 1 3 25 3 2 30 3 0 20	Adjacency Matrix Representation 0 10 15 20 10 0 35 25 15 35 0 30 20 25 30 0 The result is: 80	Adjacency Matrix Representation 0 10 15 20 10 0 35 25 15 35 0 30 20 25 30 0 The result is: 80	✔
✔	3 4 0 1 7 1 1 8 2 1 4 2 0 5	Adjacency Matrix Representation 0 7 5 7 8 4 5 4 0 The result is: 16	Adjacency Matrix Representation 0 7 5 7 8 4 5 4 0 The result is: 16	✔
✔	4 6 0 1 5 2 0 10 2 1 20 3 0 15 2 3 35 1 3 30	Adjacency Matrix Representation 0 5 10 15 5 0 20 30 10 20 0 35 15 30 35 0 The result is: 75	Adjacency Matrix Representation 0 5 10 15 5 0 20 30 10 20 0 35 15 30 35 0 The result is: 75	✔

RESULT:

Thus, the C++ program to find the TSP solution using CPP function is created successfully.