```
/*
Name : Mahesh Gaikwad
Roll no : SA21
ASSIGNMENT 7
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.
*/
#include <iostream>
#include <vector> #include
<algorithm> using
namespace std; struct
Edge
{
int src; int
dest; int
weight;
};
class Graph
{ int
V;
vector<Edge> edges; public:
Graph(int vertices) : V(vertices) {} void
addEdge(int src, int dest, int weight)
{
edges.push_back({src, dest, weight});
}
int findParent(int parent[], int i)
{
return (parent[i] == i) ? i : findParent(parent, parent[i]);
```

```
}

void unionSet(int parent[], int rank[], int x, int y)
{
int xRoot = findParent(parent, x);
int yRoot = findParent(parent, y);
if (rank[xRoot] < rank[yRoot])
parent[xRoot] = yRoot; else if
(rank[xRoot] > rank[yRoot])
parent[yRoot] = xRoot; else
{
parent[yRoot] = xRoot; rank[xRoot]++;
}
}
void kruskalMST()
{
sort(edges.begin(), edges.end(), [](const Edge &a, const Edge &b)
{ return a.weight < b.weight; });
vector<pair<int, int>> result;
int parent[V], rank[V] = {0}; for
(int i = 0; i < V; ++i) parent[i] =
i; int minCost = 0; for (auto
edge : edges)
{
int uRoot = findParent(parent, edge.src); int
vRoot = findParent(parent, edge.dest); if
(uRoot != vRoot)
{
```

```
result.push_back({edge.src, edge.dest});
minCost += edge.weight;
unionSet(parent, rank, uRoot, vRoot);
}
}
for (auto edge : result) cout << edge.first + 1 << "->" <<
edge.second + 1 << endl; cout << "Min. Cost: " <<
minCost << endl;
}
};

int main()
{ int V,
E;
cout << "Enter no. of offices: "; cin >> V; cout << "Enter no. of
connections: "; cin >> E; Graph g(V); cout << "Enter connections
(source destination weight):" << endl;
for (int i = 0; i < E; ++i)
{
int src, dest, weight; cin >>
src >> dest >> weight;
g.addEdge(src - 1, dest - 1, weight);
}
cout << "Minimum spanning tree:" << endl;
g.kruskalMST();
}
```

OUTPUT :

```
Enter no. of offices4
Enter no. of connections: 5
```

Enter connections (source destination weight):

1 2 4

1 3 3

1 4 6

2 4 4

3 4 7

Minimum spanning tree:

1->3

1->2

2->4

Min. Cost: 11