```cpp
// Kruskal's Implementation using priority_queue
//Complexity : O(E logE)   E is the no of edges
#include<bits/stdc++.h>
using namespace std;

//declaring all class those we are using
class Graph;
class MinHeap;
class UnionFind;

//Create a structure that represent a node in which there are source, destination and its weight
struct node
{
    int source;
    int dest;
    int weight;
};

//class Graph for creating a graph
class Graph
{
private:
    int rows=6,column=6; //No of rows and columns (no need)
public:

    int totalCost=0;
    void addEdge(int,int,int); //add an edge to the adj Matrix
    void kruskal(); //Perform Kruskal's Algorithm

};

//Class that contains minHeap
class MinHeap
{
private:
    struct node minHeap[500];
    int length;
public:
    void addNode(int ,int ,int);
    void heapify(int);
    void buildHeap();
    void printHeap();
    void deleteMin();
    struct node ExtractMin();
    int isEmpty();
};
//Class for performing disjoint set
class UnionFind
{
private:
    int parent[6];
    int ParentLength=6;

public:
    void initializeParent();
    int findParent(int ); //find the parent of any node
    void unionSet(int ,int ); //perform union of two sets
    void printParent();
};
Graph graph1;
MinHeap minHeap1;
UnionFind unionFind1;

void Graph::addEdge(int source,int dest,int weight)
{
    minHeap1.addNode(source,dest,weight);
}
void MinHeap::addNode(int source,int dest,int weight)
```

```cpp
{
    minHeap[length].dest=dest;
    minHeap[length].source=source;
    minHeap[length].weight=weight;
    length++;

}
void MinHeap::heapify(int parent)
{
    int leftChild=2*parent+1;
    int rightChild=2*parent+2;
    int smallest=parent;
    if(leftChild<length && minHeap[leftChild].weight<minHeap[parent].weight)
    {
        smallest=leftChild;
    }
    if(rightChild<length && minHeap[rightChild].weight<minHeap[smallest].weight)
    {
        smallest=rightChild;
    }
    if(smallest!=parent)
    {
        swap(minHeap[smallest],minHeap[parent]);
        heapify(smallest);
    }
}
void MinHeap::buildHeap()
{
    for(int i=(length-1)/2;i>=0;i--)
    {
        heapify(i);
    }
}
void MinHeap::deleteMin()
{
    if(isEmpty())
    {
        cout<<"Empty\n";
    }
    else{
    swap(minHeap[length-1],minHeap[0]);
    length--;
    heapify(0);
    }
}
int MinHeap::isEmpty()
{
    if(length==0)
    {
        return 1;
    }
    else
    {
            return 0;
    }
}
void MinHeap::printHeap()
{
    cout<<"--Min Heap -- "<<endl;
    for(int i=0;i<length;i++)
    {
        //cout<<minHeap[i].source<<" "<<minHeap[i].dest<<" "<<minHeap[i].weight<<endl;
        cout<<minHeap[i].weight<<" ";
    }
    cout<<endl;
}
struct node MinHeap::ExtractMin()
{
    return minHeap[0];
```

```cpp
};

void UnionFind::initializeParent()
{
    for(int i=0;i<ParentLength;i++)
    {
        parent[i]=-1;
    }
}

int UnionFind::findParent(int index)
{
    //cout<<"inside find Parent \n";
    if(parent[index]==-1)
    {
        return index;
    }
    else
    {
        return findParent(parent[index]);
    }
}
void UnionFind::unionSet(int u,int v)
{
    int parentU=findParent(u);
    int parentV=findParent(v);
    parent[parentV]=parentU;
}
void UnionFind::printParent()
{
    cout<<"Parent array -------"<<endl;
    for(int i=0;i<ParentLength;i++)
    {
        cout<<parent[i]<<" ";
    }
    cout<<endl;
}
//kruskal's perform
void Graph::kruskal()
{
    cout<<"---------------------------Inside Kruskal--------------------------------------------------"<<endl;
    if(minHeap1.isEmpty())
    {
        cout<<"empty\n";
        return;
    }

    struct node temp;
    temp=minHeap1.ExtractMin();
    minHeap1.deleteMin();
    int u =temp.source;
    int v = temp.dest;
    cout<<"Source="<<u<<" destination = "<<v<<" weight = "<<temp.weight<<endl;
    int parentU=unionFind1.findParent(u);
    int parentV= unionFind1.findParent(v);
    cout<<"ParentU = "<<parentU<<" ParentV = "<<parentV<<endl;
    if(parentU!=parentV)
    {
        cout<<"not cycle\n";
        unionFind1.unionSet(u,v);
        totalCost+=temp.weight;
        unionFind1.printParent();
        minHeap1.printHeap();
        kruskal();

    }
    else
    {
        kruskal();
    }
}
int main()
{
    //graph1.initializeAdjMatrix();

    graph1.addEdge(0,1,7);
    graph1.addEdge(0,2,9);
    graph1.addEdge(0,5,14);
    graph1.addEdge(1,2,10);
    graph1.addEdge(1,3,15);
    graph1.addEdge(2,3,11);
    graph1.addEdge(2,5,2);
    graph1.addEdge(3,4,6);
    graph1.addEdge(4,5,9);

    minHeap1.buildHeap();
    minHeap1.printHeap();
    unionFind1.initializeParent();
    graph1.kruskal();
    cout<<"total cost="<<graph1.totalCost<<endl;
}
```