```
********************************************************************
************/
#include <iostream>

using namespace std;

const int MAXNODES = 10;

const int INF = 9999;

// Structure to represent an edge

struct edge

{

   int u, v, cost;

};

int fnFindParent(int v, int parent[]);

void fnUnion_ij(int i, int j, int parent[]);

void fnInputGraph(int m, edge e[]);

int fnGetMinEdge(edge e[], int n);

void kruskal(int n, edge e[], int m);

/
********************************************************************
**********
*Function          : main
*Input parameters  :
    *int argc - no of commamd line arguments
    *char **argv - vector to store command line argumennts
*RETURNS           : 0 on success
********************************************************************
**********/
int main( int argc, char **argv)

{

    int n = 6, m = 10;
    edge e[2*MAXNODES] = {{0,1,6},{1,4,3},{4,5,6},{5,3,2},{3,0,5},
{0,2,1},{1,2,5},{3,2,5},{4,2,6},{5,2,4}};

cout << "Enter the number of nodes : ";

cin >> n;

cout << "Enter the number of edges : ";
cin >> m;

fnInputGraph(m, e);

kruskal(n, e, m);

return 0;

}

/
********************************************************************
**********
*Function         : fnFindParent
*Description      : Function to find parent of a given vertex
*Input parameters :
*   int v   - vertex for whom parent has to be found
*   int parent[] - parent vector
*RETURNS          : parent vertex
********************************************************************
**********/

int fnFindParent(int v, int parent[])

{

    while (parent[v] != v)
        v = parent[v];

    return v;

}

/
********************************************************************
**********
*Function   : fnUnion_ij
*Description    : Function to merge two trees
*Input parameters:
*   int i, j - vertices to be merged
*   int parent[] - parent vector
*RETURNS    : no value
********************************************************************
**********/
void fnUnion_ij(int i, int j, int parent[])

{

    if(i < j)
        parent[j] = i;
    else
        parent[i] = j;

}
```

```
/
********************************************************************
**********
*Function         : fnInputGraph
*Description      : Function to read a graph
*Input parameters :
*   int m   - no of edges in the graph
*   edge e[] - set of edges in the graph
*RETURNS          : no value
********************************************************************
**********/
void fnInputGraph(int m, edge e[])

{

    int i, j, k, cost;

    for(k=0; k<m; k++)
    {
        cout << "Enter edge and cost in the form u, v, w : \n";
        cin >> i >> j >> cost;

        e[k].u = i;
        e[k].v = j;
        e[k].cost = cost;
    }
}

/
********************************************************************
**********
*Function         : fnGetMinEdge(
*Description      : Function to find the least cost edge in the edge
set
*Input parameters :
*   edge e[] - set of edges in the graph
*   int n   - no of vertices in the graph
*RETURNS          : index of least cost edge in the edge set
********************************************************************
**********/

int fnGetMinEdge(edge e[], int n)

{

    int i, small, pos;
    small = INF;
    pos = -1;

    for(i=0; i<n; i++)
    {
        if(e[i].cost < small)
        {
            small = e[i].cost;
            pos = i;
        }
    }

    return pos;
}

void kruskal(int n, edge e[], int m)

{

    int i, j, count, k, sum, u, v, t[MAXNODES][2], pos;
    int parent[MAXNODES];
    count = 0;
    k = 0;
    sum = 0;

    for(i=0; i<n; i++)
    {
        parent[i] = i;
    }

    while(count != n-1)
    {
        pos = fnGetMinEdge(e,m);
        if(pos == -1)
        {
            break;
        }
        u = e[pos].u;
        v = e[pos].v;
        i = fnFindParent(u,parent);
        j = fnFindParent(v,parent);

        if(i != j)
        {
            t[k][0] = u;
            t[k][1] = v;
            k++;
            count++;
            sum += e[pos].cost;
            fnUnion_ij(i, j, parent);
        }
        e[pos].cost = INF;
    }

    if(count == n-1)
    {
        cout << "\nSpanning tree exists";
        cout << "\nThe Spanning tree is shown below\n";
        for(i=0; i<n-1; i++)
            cout << t[i][0] << " " << t[i][1] << endl;
        cout << "\nCost of the spanning tree : " << sum;
    }
    else
        cout << "\nThe spanning tree does not exist";
}
```

OUTPUT:

```
Enter the number of nodes : 6

Enter the number of edges : 10

Enter edge and cost in the form u, v, w :

0 1 6

Enter edge and cost in the form u, v, w :

1 4 3

Enter edge and cost in the form u, v, w :

4 5 6

Enter edge and cost in the form u, v, w :

5 3 2

Enter edge and cost in the form u, v, w :

3 0 5

Enter edge and cost in the form u, v, w :

0 2 1

Enter edge and cost in the form u, v, w :

1 2 5

Enter edge and cost in the form u, v, w :

3 2 5

Enter edge and cost in the form u, v, w :

4 2 6

Enter edge and cost in the form u, v, w :

5 2 4

Spanning tree exists
```

```
The Spanning tree is shown below

0 2

5 3

1 4

5 2

1 2

Cost of the spanning tree : 15
```