```
#include<stdio.h>

#define NIL -1
#define MAX 100

struct edge
{
    int x;
    int y;
    int weight;
    struct edge *link;
}*front = NULL;

void create_tree(struct edge tree[]);
void insert_queue(int i, int j, int wt);
struct edge *delete_queue();
int isEmpty();
void make_a_graph();

int vertices;

int main()
{
    int count;
    struct edge tree[MAX];
    int tree_weight = 0;
    make_a_graph();
    create_tree(tree);
    printf("Edges in MST:\n");
    for(count = 1; count <= vertices - 1; count++)
```

```
    {
        printf("%d->", tree[count].x);
        printf("%d\n", tree[count].y);
        tree_weight = tree_weight + tree[count].weight;
    }
    printf("Total Weight of this Minimum Spanning Tree:\t%d\n", tree_weight);
    return 0;
}

void create_tree(struct edge tree[])
{
    struct edge *tmp;
    int y1, y2, root_y1, root_y2;
    int parent[MAX];
    int i, count = 0;
    for(i = 0; i < vertices; i++)
    {
        parent[i] = NIL;
    }
    while(!isEmpty( ) && count < vertices - 1)
    {
        tmp = delete_queue();
        y1 = tmp->x;
        y2 = tmp->y;
        while(y1 != NIL)
        {
            root_y1 = y1;
            y1 = parent[y1];
        }
        while(y2 != NIL)
        {
```

```
            root_y2 = y2;
            y2 = parent[y2];
        }
        if(root_y1 != root_y2)
        {
            count++;
            tree[count].x = tmp->x;
            tree[count].y = tmp->y;
            tree[count].weight = tmp->weight;
            parent[root_y2] = root_y1;
        }
    }
    if(count < vertices - 1)
    {
        printf("Graph is Disconnected. Therefore, Spanning Tree is not possible\n");
        exit(1);
    }
}

void insert_queue(int i, int j, int wt)
{
    struct edge *tmp, *q;
    tmp = (struct edge *)malloc(sizeof(struct edge));
    tmp->x = i;
    tmp->y = j;
    tmp->weight = wt;
    if(front == NULL || tmp->weight < front->weight)
    {
        tmp->link = front;
        front = tmp;
    }
```

```
    else
    {
        q = front;
        while(q->link != NULL && q->link->weight <= tmp->weight)
        {
            q = q->link;
        }
        tmp->link = q->link;
        q->link = tmp;
        if(q->link == NULL)
        {
            tmp->link = NULL;
        }
    }
}

struct edge *delete_queue()
{
    struct edge *tmp;
    tmp = front;
    front = front->link;
    return tmp;
}

int isEmpty()
{
    if(front == NULL)
    {
        return 1;
    }
    else
```

```
    {
        return 0;
    }
}

void make_a_graph()
{
    int count, weight, maximum_edges, origin_vertex, destination_vertex;
    printf("Enter Total Number of Vertices:\t");
    scanf("%d", &vertices);
    maximum_edges = vertices * (vertices - 1)/2;
    for(count = 0; count < maximum_edges; count++)
    {
        printf("Enter Edge [%d] Co-ordinates [-1 -1] to Quit\n", count + 1);
        printf("Enter Origin Point:\t");
        scanf("%d", &origin_vertex);
        printf("Enter Destination Point:\t");
        scanf("%d", &destination_vertex);
        if((origin_vertex == -1) && (destination_vertex == -1))
        {
            break;
        }
        printf("Enter Weight for this Edge:\n");
        scanf("%d", &weight);
        if(origin_vertex >= vertices || destination_vertex >= vertices || origin_vertex < 0 ||
destination_vertex < 0)
        {
            printf("Entered Edge Co - ordinates is Invalid\n");
            count--;
        }
        else
```

```
        {
            insert_queue(origin_vertex, destination_vertex, weight);
        }
    }
}
```