

PROGRAM TITLE:Find the Minimum Spanning Tree of a simple Graph using Kruskal's Algorithm.

PROGRAM ALGORITHM:

```
Kruskal(G)
{
    //input:A weighted connected graph 'G' represented by adjacency matrix
    //output:A minimal spanning tree of G.
    H is a graph with only the vertices of G
    while i≤n-1
    {
        extract an edge from the sorted edge-set of G in the increasing
order
        if adding that edge to H does not form a cycle
            add that edge to H; i=i+1;
        move on to the next edge in the sorted edge-set of G
    }
    H is the reqd MST
}
```

PROGRAM CODE:

```
//C Program to find the Minimum Spanning Tree using Kruskal's Algorithm and
Adjacency Matrix
#include <stdio.h>
#include <stdlib.h>
#define datatype int
int create_graph(datatype **graph,int n)
{
    int i,j,x,c=0;
    datatype w;
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            graph[i][j]=9999;
        }
    }
    printf("\tIdentify the adjoining vertices:");
    for(i=0;i<n;i++)
    {
        printf("\n\tEnter the adjoining vertices of %d.Enter -99 to
stop::",i);
        for(j=0;j<n;j++)
        {
            scanf("%d",&x);
            if(x== -99)
                break;
            else
            {
                if(graph[x][i]==9999)
                {
                    printf("\tEnter corresponding weight::");
                    scanf("%d",&w);
```

```

        graph[i][x]=w;
        c++;
    }
    else
        graph[i][x]=graph[x][i];
    printf("\tEnter next adjoining vertex::");
}
}
}
return c;
}
void print_graph(datatype **graph,int n,datatype **edge)
{
    int i,j,k=0;
    printf("\n\tThe Adjacency Matrix of the Graph is::");
    for(i=-1;i<n;i++)
    {
        printf("\n");
        for(j=-1;j<n;j++)
        {
            if(i==-1)
                printf("%d",j);
            else if(j==-1)
                printf("%d",i);
            else
            {
                printf("%d",graph[i][j]);
                if(i<j && graph[i][j]!=9999)
                {
                    //storing the start and end vertices along with its
weight
                    edge[k][0]=graph[i][j];
                    edge[k][1]=i;
                    edge[k][2]=j;
                    k++;
                }
            }
            printf("\t");
        }
        printf("\n");
    }
}
int modbubblesort(datatype **a,int m)
{
    int i,j,flag;
    datatype tmp;
    for(i=0;i<m;i++)
    {
        flag=0;
        for(j=0;j<m-i-1;j++)
        {
            if(a[j][0]>a[j+1][0])//checking on basis of weight
            {
                flag=1;
                //Swapping the array elements
                tmp=a[j][0];
                a[j][0]=a[j+1][0];

```

```

        a[j+1][0]=tmp;
        tmp=a[j][1];
        a[j][1]=a[j+1][1];
        a[j+1][1]=tmp;
        tmp=a[j][2];
        a[j][2]=a[j+1][2];
        a[j+1][2]=tmp;
    }
}
    if(flag==0)//Indicates that all the elements are in their correct
position and that no swapping has been done
        return 1;
    }
    return 0;
}
void kruskal(datatype **edge,int m,int n)
{
    int i,j,k,r,s,tot=0,markr,marks;
    int *mark = (int*)malloc(n*sizeof(int));
    for(i=0;i<n;i++)
    {
        mark[i]=i;
    }
    printf("\tThe Minimum Spanning Tree is::\n\tWeight\tStart\tEnd\n");
    for(i=0,k=0;i<n-1;k++)
    {
        r=edge[k][1];markr=mark[r];
        s=edge[k][2];marks=mark[s];
        if(mark[r]!=mark[s])
        {
            tot=tot+edge[k][0];
            printf("\t%d\t%d\t%d\n",edge[k][0],edge[k][1],edge[k][2]);
            for(j=0;j<n;j++)
            {
                if(mark[j]==marks)
                    mark[j]=markr;
            }
            i++;
        }
    }
    printf("\tThe Total Weight is::%d\n",tot);
}
int main()
{
    int n,i,m;
    printf("\n\tEnter the number of vertices::");
    scanf("%d",&n);
    datatype **graph= (datatype**)malloc(n*sizeof(datatype*));
    for(i=0;i<n;i++)
    {
        graph[i]=(datatype*)malloc(n*sizeof(datatype));
    }
    m=create_graph(graph,n);
    datatype **edge = (datatype**)malloc(m*sizeof(datatype*));
    for(i=0;i<m;i++)
    {
        edge[i]=(datatype*)malloc(3*sizeof(datatype));
    }
}

```

```

}
print_graph(graph,n,edge);
modbubblesort(edge,m);
printf("The sorted edgeset is::\n\tWeight\tStart\tEnd\n");
for(i=0;i<m;i++)
{
    printf("\t%d\t%d\t%d\n",edge[i][0],edge[i][1],edge[i][2]);
}
kruskal(edge,m,n);
return 0;
}

```

OUTPUT:

```

Enter the number of vertices::7
Identify the adjoining vertices:
Enter the adjoining vertices of 0.Enter -99 to stop::1
Enter corresponding weight::1
Enter next adjoining vertex::3
Enter corresponding weight::4
Enter next adjoining vertex::-99

Enter the adjoining vertices of 1.Enter -99 to stop::0
Enter next adjoining vertex::3
Enter corresponding weight::6
Enter next adjoining vertex::4
Enter corresponding weight::4
Enter next adjoining vertex::2
Enter corresponding weight::2
Enter next adjoining vertex::-99

Enter the adjoining vertices of 2.Enter -99 to stop::1
Enter next adjoining vertex::4
Enter corresponding weight::5
Enter next adjoining vertex::5
Enter corresponding weight::6
Enter next adjoining vertex::-99

Enter the adjoining vertices of 3.Enter -99 to stop::0
Enter next adjoining vertex::1
Enter next adjoining vertex::4
Enter corresponding weight::3
Enter next adjoining vertex::6
Enter corresponding weight::4
Enter next adjoining vertex::-99

Enter the adjoining vertices of 4.Enter -99 to stop::3
Enter next adjoining vertex::1
Enter next adjoining vertex::2
Enter next adjoining vertex::5
Enter corresponding weight::8
Enter next adjoining vertex::6
Enter corresponding weight::7
Enter next adjoining vertex::-99

Enter the adjoining vertices of 5.Enter -99 to stop::4
Enter next adjoining vertex::2

```

Enter next adjoining vertex::6
 Enter corresponding weight::3
 Enter next adjoining vertex::-99

Enter the adjoining vertices of 6.Enter -99 to stop::3
 Enter next adjoining vertex::4
 Enter next adjoining vertex::5
 Enter next adjoining vertex::-99

The Adjacency Matrix of the Graph is::

-1	0	1	2	3	4	5	6
0	9999	1	9999	4	9999	9999	9999
1	1	9999	2	6	4	9999	9999
2	9999	2	9999	9999	5	6	9999
3	4	6	9999	9999	3	9999	4
4	9999	4	5	3	9999	8	7
5	9999	9999	6	9999	8	9999	3
6	9999	9999	9999	4	7	3	9999

The sorted edgeset is::

	Weight	Start End
1	0	1
2	1	2
3	3	4
3	5	6
4	0	3
4	1	4
4	3	6
5	2	4
6	1	3
6	2	5
7	4	6
8	4	5

The Minimum Spanning Tree is::

	Weight	Start End
1	0	1
2	1	2
3	3	4
3	5	6
4	0	3
4	3	6

The Total Weight is::17

DISCUSSION:

1. We use Adjacency Matrix for storing the graph.
2. The edge set is sorted according to their weights using modified Bubble Sort.
3. Another way of approaching the program was to not create the edge set and find the minimum weighted edge after each turn directly from the adjacency matrix but that would have required another matrix to mark the edges already traversed. This would have increased the space required and also the number of computations as the whole matrix would have to be traversed each time.
4. The default weight is set arbitrarily high so that it is not considered in the computations.