Title: Prim’s and Kruskal’s Algorithm

Sravani Tangeda

AP20110010174

sravani_tangeda@srmap.edu.in

Objective:

The main objective of this experiment is to implement Prim’s and Kruskal’s algorithms.

Problem Statement:

When given a network as input, Prim's and Kruskal’s minimum spanning tree algorithms determine the subset of its edges that creates a tree from the graph that has the least amount of weights added to each vertex among all possible trees.

Prim’s algorithm begins to construct the shortest-spanning tree from any vertex in the graph. Kruskal’s algorithm begins to construct the shortest spanning tree from the vertex having the lowest weight in the graph.

Given a weighted, undirected, and connected graph G, the objective is to find the minimum spanning tree G' for G using Prim's and Kruskal's Algorithm.

Algorithm:

Prim’s:

STEP 1: Choose a Node and add it to MST
STEP 2: Check all its adjacent nodes; if they are already present in MST or not, add edge weight to that edge into Min-Heap.
STEP 3: Add the smallest weighted edge and its node in Min-Heap to MST. Repeat step 2 and step 3 for Node added.
STEP 4: Repeat steps 2, step 3 for the recently added node.

Kruskal’s:

STEP 1: Sort the given edges
STEP 2: Check each edge in sorted order if it forms a cycle with already selected edges. If not Add it to the list of MSTS. If it does move to the next edge.
STEP 3: Run steps 1 and 2 till v-1 edges are selected (v=no.of.vertices).

Code:

Prim’s Code:

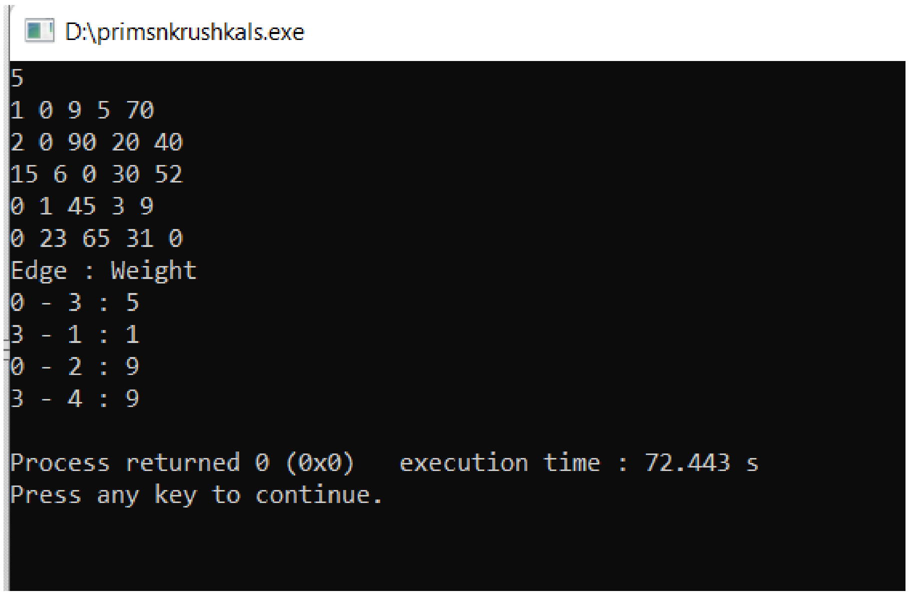
```
#include<stdio.h>

#include<stdbool.h>
```

```
#define INF 9999999

int main() {
    int V;
    scanf("%d",&V); // number of vertices in graph
    int G[V][V];
for(int i=0;i<V;i++)
{
 for(int j=0;j<V;j++)
 {
   scanf("%d",&G[i][j]); // create a 2d array of size VXV
 }
}
  int no_edge;  // number of edge

  // create a array to track selected vertex
  // selected will become true otherwise false
  int selected[V];

  // set selected false initially
  memset(selected, false, sizeof(selected));

  // set number of edge to 0
  no_edge = 0;

  // the number of egde in minimum spanning tree will be
  // always less than (V -1), where V is number of vertices in
  //graph
  selected[0] = true;
```

```
int x;  //  row number
int y;  //  col number

// print for edge and weight
printf("Edge : Weight\n");

while (no_edge < V - 1) {
  //For every vertex in the set S, find the all adjacent vertices
  // , calculate the distance from the vertex selected at step 1.
  // if the vertex is already in the set S, discard it otherwise
  //choose another vertex nearest to selected vertex  at step 1.

  int min = INF;
  x = 0;
  y = 0;

  for (int i = 0; i < V; i++) {
    if (selected[i]) {
      for (int j = 0; j < V; j++) {
        if (!selected[j] && G[i][j]) {  // not in selected and there is an edge
          if (min > G[i][j]) {
            min = G[i][j];
            x = i;
            y = j;
          }
        }
      }
    }
  }
  printf("%d - %d : %d\n", x, y, G[x][y]);
```

```
    selected[y] = true;

    no_edge++;

  }


  return 0;

}
```

Output:



D:\primsnkrushkals.exe

```
5
1 0 9 5 70
2 0 90 20 40
15 6 0 30 52
0 1 45 3 9
0 23 65 31 0
Edge : Weight
0 - 3 : 5
3 - 1 : 1
0 - 2 : 9
3 - 4 : 9

Process returned 0 (0x0)   execution time : 72.443 s
Press any key to continue.
```

Explanation:

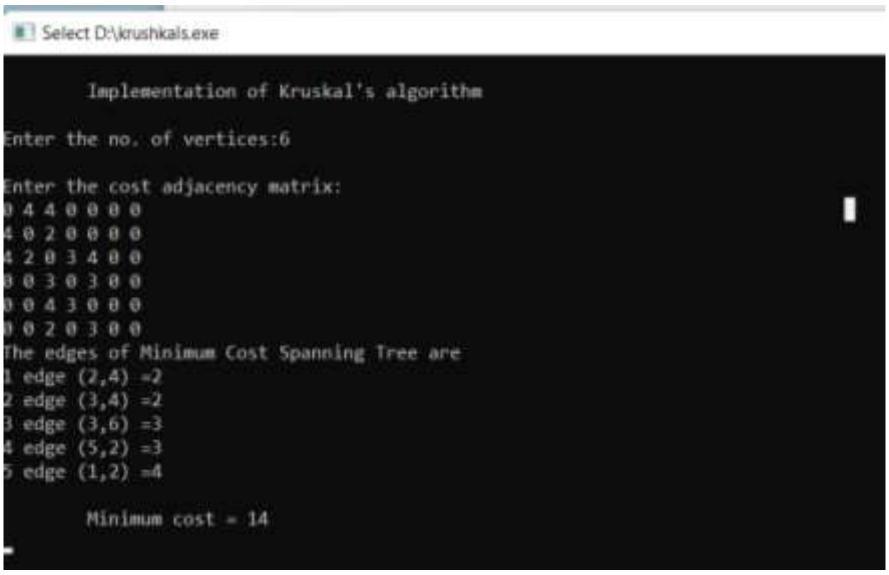
The number of vertices is taken as input which is 5 and the adjacency matrix in the form of the array is entered by the user which indicates the weights from edge to edge. From the start vertex, all the minimum cost edges are traversed and the edges selected along with their corresponding weights are printed which gives minimum cost.

Kruskal’s Code:

```
#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

int i,j,k,a,b,u,v,n,ne=1;

int min,mincost=0,cost[9][9],parent[9];

int find(int);

int uni(int,int);
```

```
void main()
{
        printf("\nEnter the no. of vertices:");
        scanf("%d",&n);     //No of Vertices
        printf("\nEnter the cost adjacency matrix:\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        scanf("%d",&cost[i][j]);   //Input the cost of adjacency matrix
                        if(cost[i][j]==0)
                                cost[i][j]=999;
                }
        }
        printf("The edges of Minimum Cost Spanning Tree are\n");
        while(ne < n)
        {
                for(i=1,min=999;i<=n;i++)
                {
                        for(j=1;j <= n;j++)
                        {
                                if(cost[i][j] < min)
                                {
                                        min=cost[i][j];
                                        a=u=i;
                                        b=v=j;
                                }
                        }
                }
                u=find(u);
                v=find(v);
                if(uni(u,v))
                {
                        printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);  //Print the edge and
weight
                        mincost +=min;
                }
                cost[a][b]=cost[b][a]=999;
        }
        printf("\n\tMinimum cost = %d\n",mincost);   //Print the minimum cost
        getch();
}
int find(int i)
{
        while(parent[i])
        i=parent[i];
        return i;
}
int uni(int i,int j)
{
        if(i!=j)
        {
                parent[j]=i;
                return 1;
        }
        return 0;
}
```

Output:


Select D:\krushkals.exe

```
       Implementation of Kruskal's algorithm

Enter the no. of vertices:6

Enter the cost adjacency matrix:
0 4 4 0 0 0 0
4 0 2 0 0 0 0
4 2 0 3 4 0 0
0 0 3 0 3 0 0
0 0 4 3 0 0 0
0 0 2 0 3 0 0
The edges of Minimum Cost Spanning Tree are
1 edge (2,4) =2
2 edge (3,4) =2
3 edge (3,6) =3
4 edge (5,2) =3
5 edge (1,2) =4

       Minimum cost = 14
```

Explanation:

The number of vertices is taken as input which is 6 and the adjacency matrix in the form of the 2D array is entered by the user which indicates the weights from edge to edge. The edges are sorted based on the cost of the edges. The starting vertex is the vertex of the edge with minimum weight. The edges are traversed in this way and give the minimum cost. Here the edges and corresponding weights are printed and the minimum cost obtained is 14.

Problems Faced:

Initially, I found it difficult to implement Prim's and Kruskal's algorithms and know the difference and best approach among them.

Conclusion:

From implementation, it indicates that the least spanning tree problem can be appropriately solved using Kruskal and Prim. The Kruskal algorithm prioritized edge finding, whereas the Prim algorithm prioritized node searching. A graph with more nodes and fewer edges will benefit from the Kruskal algorithm more than the Prim algorithm, according to test results. The prim method will perform better than Kruskal if the graph contains more edges and fewer nodes because it is based on node searching.