Experiment Number: 6

Problem Statement:-Doing the comparative analysis of Kruskal's & Prim's algorithm by finding out Minimum Cost Spanning Tree.

Aim:- Implementation of Prim's and Kruskal's Algorithm for finding Minimum Cost Spanning Tree.

Theory:-

Kruskal's Algorithm

Kruskal's Algorithm is directly based on the generic Minimum Spanning Tree algorithm. It builds the MST in forest. Initially, each vertex is in its own tree in forest. Then, algorithm considers each edge in turn, order by increasing weight. If an edge (u, v) connects two different trees, then (u, v) is added to the set of edges of the MST, and two trees connected by an edge (u, v) are merged into a single tree on the other hand, if an edge (u, v) connects two vertices in the same tree, then edge (u, v) is discarded.
A little more formally given a connected, undirected and weighted graph with a function
$w: E \rightarrow R$.

- Starts with each vertex being its own component.
- Repeatedly merges two components into one by choosing the light edge that connects them (i.e., the light edge crossing the cut between them).
- Scans the set of edges in monotonically increasing order by weight.
- Uses a disjoint-set data structure to determine whether an edge connects vertices in different components.

Algorithm: -

Start with an empty set A, and select at every stage the shortest edge that has not been chosen or rejected, regardless of where this edge is situated in the graph.

KRUSKAL(V, E, *w*)

```
A ← { }          ▷ Set A will ultimately contains the edges of the MST
for each vertex v in V
   do MAKE-SET(v)
sort E into non-decreasing order by weight w
for each (u, v) taken from the sorted list
   do if FIND-SET(u) = FIND-SET(v)
      then A ← A ∪ {(u, v)}
           UNION(u, v)
return A
```

Analysis: -

Initialize the set A: O(1)
First for loop: |V| MAKE-SETs
Sort E: O(E log E)
Second for loop: O(E) FIND-SETs and UNIONs
Therefore, O(E lg V) time. (If edges are already sorted, O(E α(V)), which is almost linear.)

Prim's algorithm

It starts from an arbitrary vertex (root) and at each stage, adds a new branch (edge) to the tree already constructed; the algorithm halts when all the vertices in the graph have been reached. This strategy is

greedy in the sense that at each step the partial spanning tree is augmented with an edge that is the smallest among all possible adjacent edges.

Algorithm: -

Input: A weighted, undirected graph G=(V, E, w)

Output: A minimum spanning tree T.

T={}.

Let r be an arbitrarily chosen vertex from V.

U = {r}

WHILE | U| < n

DO

Find u in U and v in V-U such that the edge (u, v) is a smallest edge between U-V.

T = TU{(u, v)}

U= UU{v}

Analysis: -

The algorithm spends most of its time in finding the smallest edge. So, time of the algorithm basically depends on how we search this edge.

Straight forward method: Just find the smallest edge by searching the adjacency list of the vertices in V. In this case, each iteration costs O(m) time, yielding a total running time of O(m * n).

Binary Heap:

By using binary heaps, the algorithm runs in O(m log n).

Fibonacci Heap:

By using Fibonacci heaps, the algorithm runs in O(m + n log n) time.

Output: -

Note: In this section you have to take one sample graph example and show the MCST for the corresponding graph using Kruskal as well as Prims Algorithm. While showing the output you should print the start point of edge, end point of edge and cost of that edge and finally total cost of MCST.

Questionnaires

Ques.1	What do you mean by minimum spanning tree?
Ans	A *minimum spanning tree (MST)* or minimum weight spanning tree for a weighted, connected and undirected graph is a spanning tree with weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.
Ques.2	How does krushkals Algorithm works?
Ans	1.Sort all the edges in non-decreasing order of their weight. 2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it. 3. Repeat step#2 until there are (V-1) edges in the spanning tree.
Ques.3	How does Prim's Algorithm Works?
Ans	Prim's algorithm is also a Greedy algorithm. It starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets, and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.
Ques.4	What is the running time for kruskal's and prim's algorithms?
Ans	Kruskal's Algorithm:O(m logn) Prim's Algorithm: O(m + nlogn)

Program

Prims Algorithm

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]={0},min,mincost=0,cost[10][10];
void main()
{
clrscr();
printf("\nEnter the number of nodes:");
        scanf("%d",&n);
        printf("\nEnter the adjacency matrix:\n");
        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                {
```

```
                        scanf("%d",&cost[i][j]);
                        if(cost[i][j]==0)
                                cost[i][j]=999;
                }
        visited[1]=1;
        printf("\n");
        while(ne < n)
                {
                for(i=1,min=999;i<=n;i++)
                        for(j=1;j<=n;j++)
                        if(cost[i][j]<min)
                        if(visited[i]!=0)
                        {
                                min=cost[i][j];
                                a=u=i;
                                b=v=j;
                        }
                        if(visited[u]==0 || visited[v]==0)
                {
                        printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
                        mincost+=min;
                        visited[b]=1;
                }
                cost[a][b]=cost[b][a]=999;
        }
        printf("\n Minimum cost for tree traversal=%d",mincost);
        getch();
}
//      output
Enter the number of nodes:4
Enter the adjacency matrix:
100                                                     100
2                                                       3
100                                                     100
8                                                       5
2                                                       8
100                                                     7
3                                                       5
7                                                       100
Edge 1:(1 2) cost:2
 Edge 2:(2 3) cost:3
 Edge 3:(3 4) cost:5
 Minimum cost for tree traversal=10
```

Kruskals Algorithm

```
#include<stdio.h>
```

```
#include<conio.h>
int parent[50];
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;
}
void main()
{  int i,j;
	int c[50][50],nv,ne=1,u,v,a,b,min,mincost=0;
	clrscr();
	printf("enter no of vertices:");
	scanf("%d",&nv);
 //	printf("enter no. of edges:");
  //	scanf("%d",&ne);
	printf("enter cost adj matrix:");
	for(i=0;i<nv;i++)
	{
		for(j=0;j<nv;j++)
		{
			scanf("%d",&c[i][j]);
		}
	}
	for(i=0;i<nv;i++)
	{
		parent[i]=0;
	}
	while(ne<nv)
	{
		for(i=0,min=999;i<nv;i++)
		{
			for(j=0;j<nv;j++)
			{
```

```
                        if(c[i][j]<min)
                        {
                                min=c[i][j];
                                u=a=i;
                                v=b=j;
                        }
                }
        }

        u=find(u);
        v=find(v);
        if(uni(u,v))
        {
                mincost=mincost+min;
                printf("\n%d to %d minimim cost= %d",a,b,min);
        }
        c[a][b]=c[b][a]=999;
        ne++;
    }
    printf("\nminimum cost of spanning tree is %d",mincost);

    getch();
}
//OUTPUT
Enter no of vertices: 4
Enter cost adj matrix: 999
2
999
8
2
999
8
7
999
3
999
5
8
7
5
999
```

0 to 1 minimim cost= 2
2 to 1 minimim cost= 3
2 to 3 minimim cost= 5
Minimum cost of spanning tree is 10

NAME:HARSHBHAI SOLANKI
ROLL.N0:62
SE-4-D

EXPERIMENT N0-6

AIM: Implementation of Prim's and Kruskal's algorithm for Minimum Cost Spanning Tree.

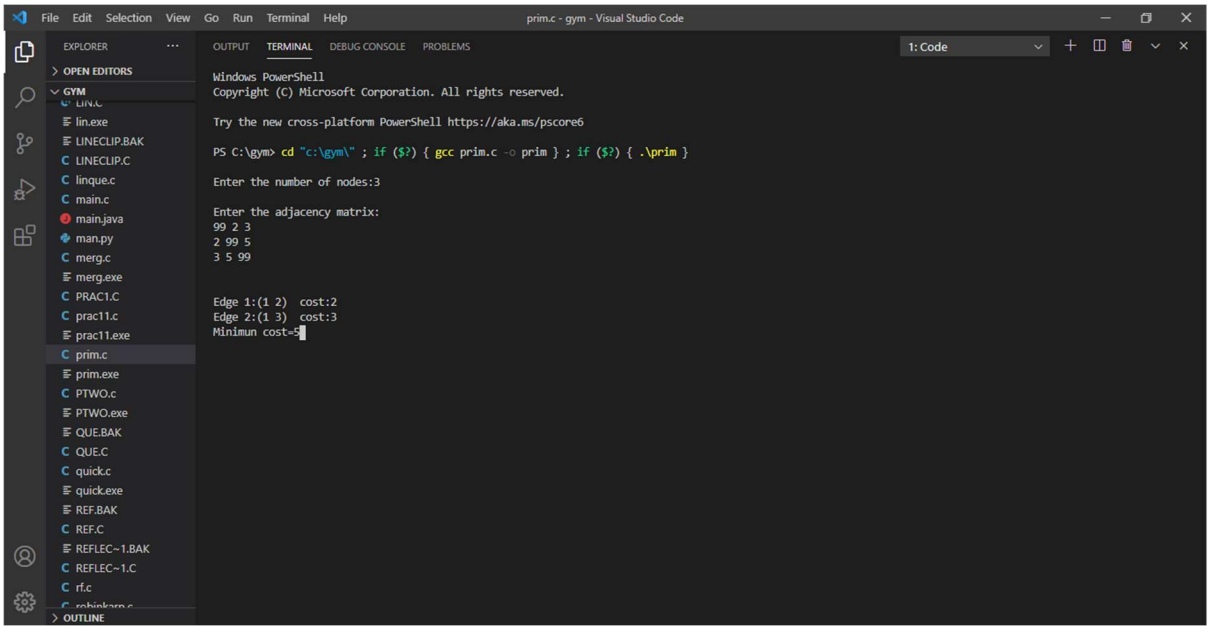
PRIM'S ALGORITHM:

CODE:

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]= {
        0
}
,min,mincost=0,cost[10][10];
void main() {
        printf("\nEnter the number of nodes:");
        scanf("%d",&n);
        printf("\nEnter the adjacency matrix:\n");
        for (i=1;i<=n;i++)
          for (j=1;j<=n;j++) {
                scanf("%d",&cost[i][j]);
                if(cost[i][j]==0)
                   cost[i][j]=999;
        }
        visited[1]=1;
        printf("\n");
        while(ne<n) {
                for (i=1,min=999;i<=n;i++)
```

```
        for (j=1;j<=n;j++)
         if(cost[i][j]<min)
          if(visited[i]!=0) {
              min=cost[i][j];
              a=u=i;
              b=v=j;
          }
          if(visited[u]==0 || visited[v]==0) {
              printf("\nEdge %d:(%d %d)  cost:%d",ne++,a,b,min);
              mincost+=min;
              visited[b]=1;
          }
          cost[a][b]=cost[b][a]=999;
    }
    printf("\nMinimun cost=%d",mincost);
    getch();
}
```

OUTPUT:



KRUSKAL'S ALGORITHM:

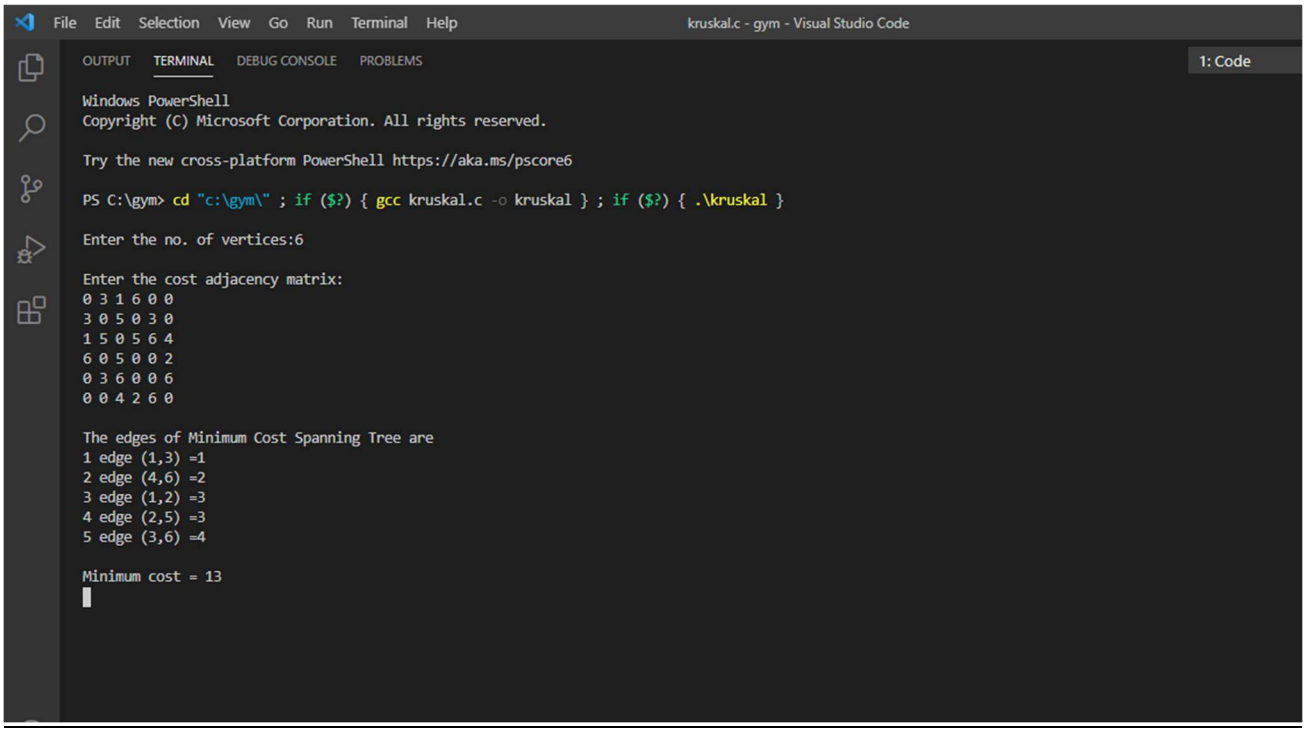
CODE:

```
#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);
        printf("\nEnter the cost adjacency matrix:\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        scanf("%d",&cost[i][j]);
                        if(cost[i][j]==0)
                                cost[i][j]=999;
                }
        }
        printf("\nThe 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);
                mincost +=min;
            }
            cost[a][b]=cost[b][a]=999;
        }
        printf("\nMinimum cost = %d\n",mincost);
        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:



CONCLUSION:

By performing above practical we can conclude the following points:

- Prim’s algorithm has a time complexity of O(V2), V being the number of vertices and can be improved up to O(E + log V) using Fibonacci heaps.
- Kruskal’s algorithm’s time complexity is O(E log V), V being the number of vertices.
- Prim’s algorithm gives connected component as well as it works only on connected graph.
- Kruskal’s algorithm can generate forest(disconnected components) at any instant as well as it can work on disconnected components
- prim's algorithm does not need sorted edges
- kruskal's algorithm need sorted edges
- Prim’s algorithm runs faster in dense graphs.
- Kruskal’s algorithm runs faster in sparse graphs.