Program 4:

Find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm. Write the program in C/C++

```
#include<stdio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10],min,mincost=0,cost[10][10];
int main()
{
printf("Enter the no. of vertices:\n");
scanf("%d",&n);

printf("enter the graph data:\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;
}

for(i=2;i<=n;i++)
visited[i]=0;
printf("The edges of spaning tree are:\n");
visited[1]=1;

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)
               continue;
               else
                  {
                     min=cost[i][j];
                     a=u=i;
                     b=v=j;
                  }
            }

            if(visited[u]==0 || visited[v]==0)
            {
               printf("%d\tEdge\t(%d,%d)=%d\n",ne++,u,v,min);
               mincost+=min;
               visited[v]=1;
            }

         cost[u][v]=cost[v][u]=999;
  }

printf("\n\tMINCOST =%d\n",mincost);
return 1;

}
```

```
Enter the no. of vertices:
3
enter the graph data:
999 10 5
10 999 5
5 5 999
The edges of spaning tree are:
1        Edge     (1,3)=5
2        Edge     (3,2)=5

         MINCOST =10
```

ALGORITHM *Prim(G)*

//Prim's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph $G = \langle V, E \rangle$
//Output: E_T, the set of edges composing a minimum spanning tree of G
$V_T \leftarrow \{v_0\}$ //the set of tree vertices can be initialized with any vertex
$E_T \leftarrow \varnothing$
for $i \leftarrow 1$ **to** $|V| - 1$ **do**
    find a minimum-weight edge $e^* = (v^*, u^*)$ among all the edges (v, u)
    such that v is in V_T and u is in $V - V_T$
    $V_T \leftarrow V_T \cup \{u^*\}$
    $E_T \leftarrow E_T \cup \{e^*\}$
return E_T

Efficiency

- The time efficiency of depends on the data structures used for implementing the priority queue and for representing the input graph.
- Since we have implemented using weighted matrix and unordered array, the efficiency is O(|V²|).
- If we implement using adjacency list and the priority queue for min-heap, the efficiency is O(|E|log|V|).

Program 5:

Find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm. Use Union-Find algorithms in your program. Write the program in C/C++.

```
#include<stdio.h>
int parent[10],min,ne=1,mincost=0,cost[10][10];
int i,j,a,b,u,v,n;
int main()
{
printf("Enter the no. of vertices:\n");
scanf("%d",&n);
printf("Enter the graph data:\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;
}
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=l;
                   b=v=j;
                }

  while (parent[u])
  u=parent[u];
  while(parent[v])
  v=parent[v];
if(u!=v)
{
printf("\n%d\tEdge\t(%d,%d)=%d",ne++,a,b,min);
mincost+=min;
parent[v]=u;
}
cost[a][b]=cost[b][a]=999;
}
printf("\n\tMINCOST=%d\n",mincost);
return 1;
}
```

```
Enter the no. of vertices:
5
Enter the graph data:
0 3 999 7 999
3 0 4 2 999
999 4 0 5 6
7 2 5 0 4
999 999 6 4 0

1        Edge     (2,4)=2
2        Edge     (1,2)=3
3        Edge     (2,3)=4
4        Edge     (4,5)=4
         MINCOST=13
```

ALGORITHM *Kruskal(G)*

```
//Kruskal's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph G = ⟨V, E⟩
//Output: E_T, the set of edges composing a minimum spanning tree of G
sort E in nondecreasing order of the edge weights w(e_{i_1}) ≤ · · · ≤ w(e_{i_|E|})
E_T ← ∅;   ecounter ← 0      //initialize the set of tree edges and its size
k ← 0                        //initialize the number of processed edges
while ecounter < |V| − 1 do
    k ← k + 1
    if E_T ∪ {e_{i_k}} is acyclic
        E_T ← E_T ∪ {e_{i_k}};   ecounter ← ecounter + 1
return E_T
```

- If the graph is represented as an adjacency matrix then the complexity of kruskal algorithm is V^2
- If you use binary heap and adjacency list the complexity can be of the order of ElogV.