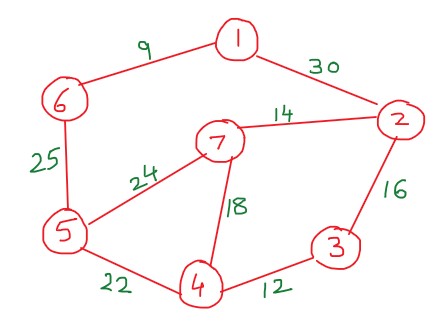
**EXPERIMENT 6**

**Aim:** Implement **Prims** and **Kruskal’s** algorithm for finding Minimum cost spanning tree using **Greedy** Method.

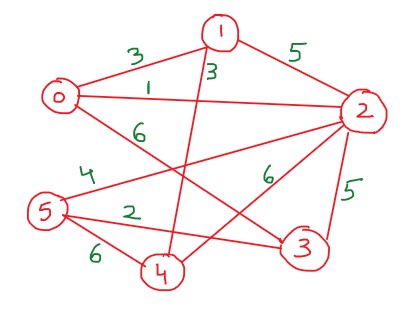
**Problem statement:**

Write a menu driven program to solve the Minimum cost spanning tree problem using Prims and Kruskals algorithm.

**Input :** For Prims algorithm input the below graph as a cost adjacency matrix.



For Kruskals algorithm input the below graph as a cost adjacency matrix.



**Output:** Display the edges that are added to the MST along with their cost Also display the final optimal cost of the MST.

**(Paste your code and output below) Code:**

#include<stdio.h>

#include<stdlib.h>

#define MAX 30

int a,b,u,v,n,i,j,ne=1; int i,j,total\_cost; int visited[10]={0},min,mincost=0,cost[10][10];

typedef struct edge

{

int u,v,w; }edge;

typedef struct edgelist

{ edge data[MAX];

int n; }edgelist;

edgelist elist;

int G[MAX][MAX],n;

edgelist spanlist; void kruskal(); int find(int belongs[],int vertexno); void union1(int belongs[],int c1,int c2); void sort(); void print(); void prim();

int main()

{

int ch; while(1){ printf(" \n 1. Prim's Algo \n 2. Kruskal's Algo\n 3.Exit \n Enter your choice: "); scanf("%d",&ch); switch(ch){ case 1 : prim(); break;

case 2 : kruskal();

break;

case 3 : exit(0); default : printf("\n Enter a valid choice!"); break;

}

}

return 0;

}

void prim()

{ printf("\nEnter the number of nodes:"); 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;

} 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=%d",mincost);

}

void kruskal()

{ int belongs[MAX],i,j,cno1,cno2; elist.n=0; printf("\nEnter number of vertices:"); scanf("%d",&n); printf("\nEnter the cost adjacency matrix:\n"); for(i=0;i<n;i++) for(j=0;j<n;j++){ scanf("%d",&G[i][j]);

} for(i=1;i<n;i++){ for(j=0;j<i;j++)

{

if(G[i][j]!=0)

{

elist.data[elist.n].u=i; elist.data[elist.n].v=j; elist.data[elist.n].w=G[i][j]; elist.n++;

}

}

}

sort(); for(i=0;i<n;i++){ belongs[i]=i;

}

spanlist.n=0; for(i=0;i<elist.n;i++)

{

cno1=find(belongs,elist.data[i].u); cno2=find(belongs,elist.data[i].v); if(cno1!=cno2)

{

spanlist.data[spanlist.n]=elist.data[i]; spanlist.n=spanlist.n+1; union1(belongs,cno1,cno2);

}

}

print();

}

int find(int belongs[],int vertexno)

{ return(belongs[vertexno]);

}

void union1(int belongs[],int c1,int c2)

{

int i;

for(i=0;i<n;i++){ if(belongs[i]==c2){ belongs[i]=c1;

}

}

}

void sort()

{

int i,j;

edge temp; for(i=1;i<elist.n;i++){ for(j=0;j<elist.n-1;j++){ if(elist.data[j].w>elist.data[j+1].w)

{

temp=elist.data[j]; elist.data[j]=elist.data[j+1]; elist.data[j+1]=temp;

}

}

}

}

void print()

{

int i,cost=0; for(i=0;i<spanlist.n;i++)

{

printf("\n%d -- %d : Cost = %d",spanlist.data[i].u,spanlist.data[i].v,spanlist.data[i].w);

cost=cost+spanlist.data[i].w;

} printf("\n\nCost of the spanning tree=%d",cost);

}

**Output:**

