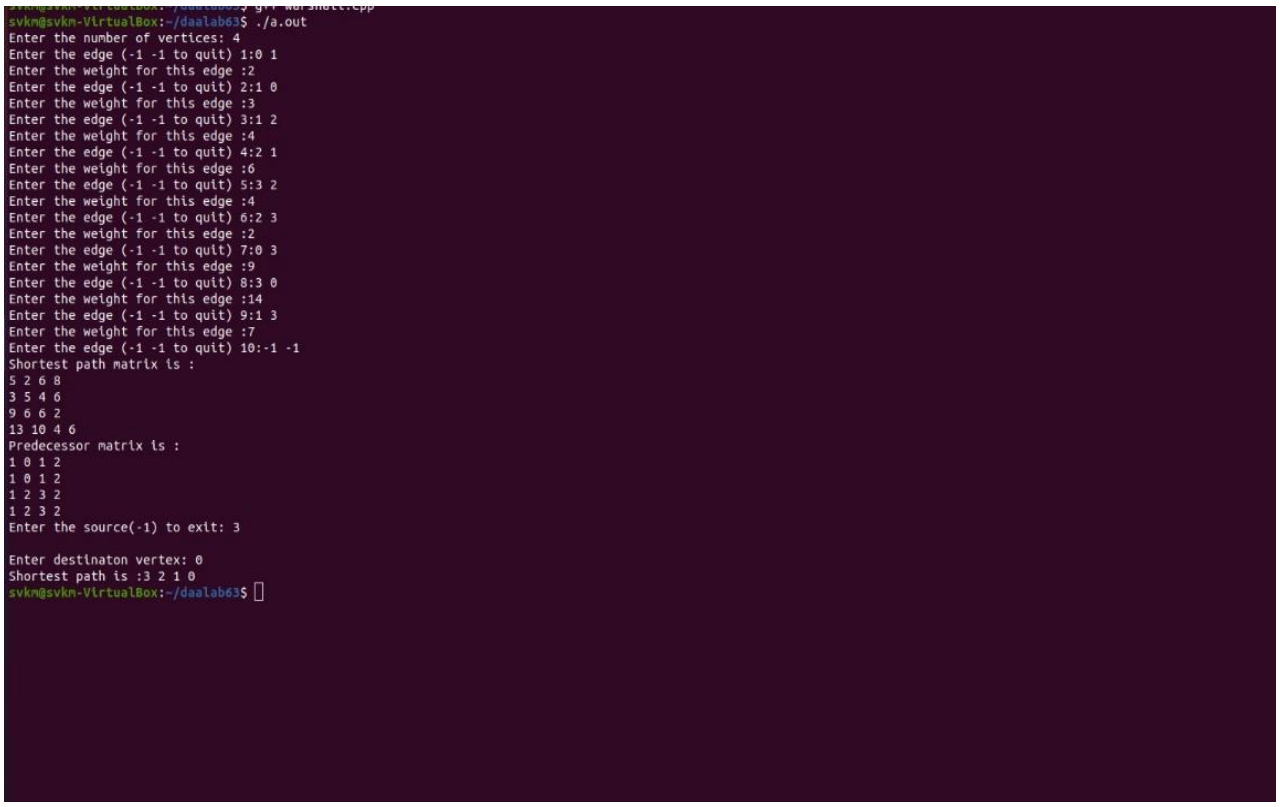
Floyd warshall:

```
#include<iostream>
using namespace std;
#define infi 99999
#define max 100
int n;
int adj[max][max];
int D[max][max];
int pre[max][max];
void createg();
void warshall();
void find_path(int s,int d);
void display(int mat[max][max],int n);
int main(){
int s,d;
createg();
warshall();
while(1){
cout<<"Enter the source(-1) to exit: ";
cin>>s;
if(s==-1)
break;
cout<<"\nEnter destinaton vertex: ";
cin>>d;
if(s<0 || s>n-1||d<0 ||d>n-1){
cout<<"Enter a valid vertex\n";
continue;}
cout<<"Shortest path is :";
find_path(s,d);
cout<<"Length of path is "<<D[s][d];

return 0;

}

}

void createg(){

int i,max_edges,o,d,wt;

cout<<"Enter the number of vertices: ";

cin>>n;

max_edges=n*(n-1);

for(i=1;i<=max_edges;i++){

cout<<"Enter the edge (-1 -1 to quit) "<<i<<":";

cin>>o>>d;

if((o==-1)&&(d==-1))

break;

cout<<"Enter the weight for this edge :";

cin>>wt;


if(o>=n||d>=n|o<0||d<0){

cout<<"Invalid edge\n";

i--;}

else{

adj[o][d]=wt;

}

}

}


void warshall(){

int i,j,k;

for(i=0;i<n;i++)
for(j=0;j<n;j++){
if(adj[i][j]==0){
D[i][j]=infi;
pre[i][j]=-1;
}
else{
D[i][j]=adj[i][j];
pre[i][j]=i;
}}
for(k=0;k<n;k++){
for(i=0;i<n;i++)
for(j=0;j<n;j++)
if(D[i][k]+D[k][j] <D[i][j]){
D[i][j]=D[i][k]+D[k][j];
pre[i][j]=pre[k][j];
}
}
cout<<"Shortest path matrix is :\n";
display(D,n);
cout<<"Predecessor matrix is :\n";
display(pre,n);
}
void find_path(int s,int d){
int i;
int path[max];
int count ;
if(D[s][d]==infi){
cout<<"No path";
return;
}
count=-1;
do{
path[++count]=d;
d=pre[s][d];
}
while(d!=s);
path[++count]=s;
for(i=count ;i>=0;i--)
cout<<path[i]<<" ";
cout<<"\n";
}
void display(int mat[max][max],int n){
int i,j;
for(i=0;i<n;i++){
for(j=0;j<n;j++)
cout<<mat[i][j]<<" ";
cout<<"\n";
}}
```

Output :



```
svkm@svkm-VirtualBox:~/daalab63$ g++ warshall.cpp
svkm@svkm-VirtualBox:~/daalab63$ ./a.out
Enter the number of vertices: 4
Enter the edge (-1 -1 to quit) 1:0 1
Enter the weight for this edge :2
Enter the edge (-1 -1 to quit) 2:1 0
Enter the weight for this edge :3
Enter the edge (-1 -1 to quit) 3:1 2
Enter the weight for this edge :4
Enter the edge (-1 -1 to quit) 4:2 1
Enter the weight for this edge :6
Enter the edge (-1 -1 to quit) 5:3 2
Enter the weight for this edge :4
Enter the edge (-1 -1 to quit) 6:2 3
Enter the weight for this edge :2
Enter the edge (-1 -1 to quit) 7:0 3
Enter the weight for this edge :9
Enter the edge (-1 -1 to quit) 8:3 0
Enter the weight for this edge :14
Enter the edge (-1 -1 to quit) 9:1 3
Enter the weight for this edge :7
Enter the edge (-1 -1 to quit) 10:-1 -1
Shortest path matrix is :
5 2 6 8
3 5 4 6
9 6 6 2
13 10 4 6
Predecessor matrix is :
1 0 1 2
1 0 1 2
1 2 3 2
1 2 3 2
Enter the source(-1) to exit: 3

Enter destinaton vertex: 0
Shortest path is :3 2 1 0
svkm@svkm-VirtualBox:~/daalab63$ 
```

Prisms program :

```
#include<iostream>
using namespace std;
#define MAX  10
#define TEMP  0
#define PERM 1
#define infinity 9999
#define NIL -1
struct edge
{
int u;
int v;
};
int n;
int adj[MAX][MAX];
int predecessor[MAX];
int status[MAX];
int length[MAX];
void create_graph();
void make_tree(int r,struct edge tree[MAX]);
int min_temp();
int main()
{
int wt_tree=0;
int i,root;
struct edge tree[MAX];
create_graph();
cout<<"\nEnter the root vertex : ";
cin>>root;
```

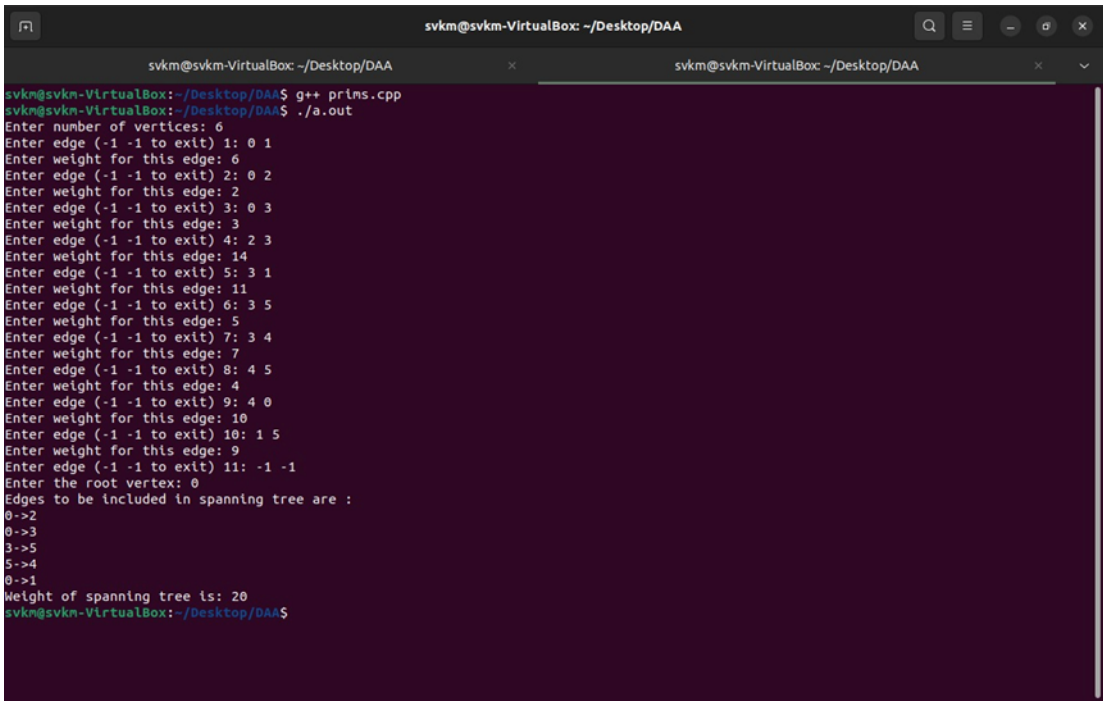
```
make_tree(root,tree);
cout<<"\nEdges to be included in spanning tree are :\n";
for(i=1;i<=n-1;i++)
{
cout<<tree[i].u<<" ";
cout<<tree[i].v;
cout<<"\n";
wt_tree += adj[tree[i].u][tree[i].v];
}
cout<<"The weight of spanning tree is :";
cout<<wt_tree;
return 0;
}
void create_graph()
{
int i,max_edges,origin,destin,wt;
cout<<"\nEnter the number of vertices :";
cin>>n;
max_edges=n*(n-1)/2;
for(i=1;i<=max_edges;i++)
{
cout<<"Enter edge(-1 -1 to quit) "<<i<<" : " ;
cin>>origin>>destin;
if((origin==-1)&&(destin==-1))
break;
cout<<"Enter the weight for this edge :";
cin>>wt;
if(origin>=n||destin>=n||origin<0||destin<0)
{
```

```
cout<<"\nInvalid Edge";
i--;
}
else
{
adj[origin][destin]=wt;
adj[destin][origin]=wt;
}
}
}
void make_tree(int r, struct edge tree[MAX])
{
int current,i;
int count=0;
for(i=0;i<n;i++)
{
predecessor[i]=NIL;
length[i]=infinity;
status[i]=TEMP;
}
length[r]=0;
while(1)
{
current=min_temp();
if(current==NIL)
{
if(count==n-1)
return;
else
```

```
{
cout<<"\nGraph is not connected,No spanning tree is possible";
exit(1);
}
}
status[current]=PERM;
if (current!=r)
{
count++;
tree[count].u=predecessor[current];
tree[count].v=current;
}
for(i=0;i<n;i++)
if (adj[current][i]>0 && status[i]==TEMP)
if (adj[current][i]<length[i])
{
predecessor[i]=current;
length[i]=adj[current][i];
}
}
}
int min_temp()
{
int i;
int min=infinity;
int k=-1;
for(i=0;i<n;i++)
{
if(status[i]==TEMP && length[i]<min)
```

```
{
min=length[i];
k=i;
}
}
return k;
}
```

Output :


svkm@svkm-VirtualBox: ~/Desktop/DAA
svkm@svkm-VirtualBox: ~/Desktop/DAA
svkm@svkm-VirtualBox: ~/Desktop/DAA

```
svkm@svkm-VirtualBox:~/Desktop/DAA$ g++ prims.cpp
svkm@svkm-VirtualBox:~/Desktop/DAA$ ./a.out
Enter number of vertices: 6
Enter edge (-1 -1 to exit) 1: 0 1
Enter weight for this edge: 6
Enter edge (-1 -1 to exit) 2: 0 2
Enter weight for this edge: 2
Enter edge (-1 -1 to exit) 3: 0 3
Enter weight for this edge: 3
Enter edge (-1 -1 to exit) 4: 2 3
Enter weight for this edge: 14
Enter edge (-1 -1 to exit) 5: 3 1
Enter weight for this edge: 11
Enter edge (-1 -1 to exit) 6: 3 5
Enter weight for this edge: 5
Enter edge (-1 -1 to exit) 7: 3 4
Enter weight for this edge: 7
Enter edge (-1 -1 to exit) 8: 4 5
Enter weight for this edge: 4
Enter edge (-1 -1 to exit) 9: 4 0
Enter weight for this edge: 10
Enter edge (-1 -1 to exit) 10: 1 5
Enter weight for this edge: 9
Enter edge (-1 -1 to exit) 11: -1 -1
Enter the root vertex: 0
Edges to be included in spanning tree are :
0->2
0->3
3->5
5->4
0->1
Weight of spanning tree is: 20
svkm@svkm-VirtualBox:~/Desktop/DAA$
```