- **Graph.h**

```
#ifndef GRAPH_H_
#define GRAPH_H_

class Graph {
        int weight[20][20];
        int directed;
        int dist[20];
        int path[20];
        int vn,en;
        std::string str[20];
public:
        Graph();
        void createGraph();
        void dijkstra();
        void display();
        virtual ~Graph();
};

#endif /* GRAPH_H_ */
```

- **Graph.cpp**

```
#include<iostream>
#include "Graph.h"
using namespace std;
Graph::Graph() {
	// TODO Auto-generated constructor stub
	cout<<"Enter 1:Directed graph"<<endl<<"0:Undirected graph::";
	cin>>directed;
	cout<<endl;
	do{
		cout<<"Enter number of vertices (Max 20):";
		cin>>vn;
		cout<<"Enter number of edges (Max 20):";
		cin>>en;
	}while(vn<1 || vn>20 || en<1 || en>20);

	for(int i=0;i<vn;i++)
		for(int j=0;j<vn;j++)
			weight[i][j]=0;
}
//==============create graph===========================
void Graph::createGraph(){
	int a,b,w;
	cout<<endl<<"============================================"<
<endl;
	cout<<"Enter landmarks for following vartices:"<<endl;
	for(int i=0;i<vn;i++)
	{
		cout<<"Vertex "<<i<<":";
		cin>>str[i];
	}
	cout<<endl<<"You entered::"<<endl;
	for(int i=0;i<vn;i++){
		cout<<i<<":"<<str[i]<<endl;
	}
	cout<<endl<<"============================================"<
<endl;
	cout<<"Enter edges of graph"<<endl;
	if(directed==0)
	{
        for(int i=0;i<en;i++){
            cout<<"Enter vertex 1 vertex 2 and weight:";
            cin>>a>>b>>w;
            weight[a][b]=w;
            weight[b][a]=w;
        }
    }
    else
    {
        for(int i=0;i<en;i++){
            cout<<"Enter vertices and weight:";
            cin>>a>>b>>w;
            weight[a][b]=w;
        }
    }

}
//=========display graph==========================
void Graph::display(){
    cout<<"Graph matix is:";
    for(int i=0;i<vn;i++){
        cout<<endl;
        for(int j=0;j<vn;j++){
            cout<<weight[i][j]<<" ";
        }
    }
}
//===============dijkstra algorithm==============
void Graph::dijkstra(){
    int visited[vn];
    int src,current;
    cout<<endl<<"Enter source vertex:";
    cin>>src;

    //find initial distance from source to all vertices
    for(int i=0;i<vn;i++){
        if(weight[src][i]!=0)
            dist[i]=weight[src][i];
        else
            dist[i]=32767;
        path[i]=src;
        visited[i]=0;
    }
    //display initial distances from source
    cout<<"Vertex\tPath\tDistance"<<endl;
    for(int i=0;i<vn;i++){
        cout<<str[i]<<"\t"<<str[path[i]]<<"\t"<<dist[i]<<endl;
    }
    //take source as current vertex and make it visited
    current=src;
    visited[current]=1;


    for(int j=0;j<vn-2;j++){
        int mindist=32767;
        //find minimum distance from current to all other vertices
        for(int i=0;i<vn;i++){
            if(visited[i]==0 && dist[i]<mindist){
                mindist=dist[i];
                current=i;
            }
        }
        //display selected vertex i.e. in current
        cout<<endl<<"Selected vertex:"<<current;
        cout<<endl<<"Cost:"<<mindist<<endl;
        //make current as visited
        visited[current]=1;


        //find shortest path from current
        for(int i=0;i<vn;i++){
            if(visited[i]==0 && (dist[current]+weight[current][i]) <
dist[i]){
                if(weight[current][i]!=0){
                    dist[i]=dist[current]+weight[current][i];
                    path[i]=current;
                }

            }
        }
    }
    //display shortest path
    cout<<endl<<"Shortest path from source to all verices:"<<endl;
    for(int i=0;i<vn;i++){
        if(i!=src){
            cout<<endl<<str[i]<<":: Distance:"<<dist[i]<<" Path: "<<str[i];
            int j=i;
                    do
                    {
                        j=path[j];
                        cout<<" <- "<<str[j];
                    }while(j!=src); //j!=Source
        }
    }

}


Graph::~Graph() {
    // TODO Auto-generated destructor stub
}
```

- **Assignemt8.cpp**

```
//==========================================================
==================
// Name        : Assignement8.cpp
// Author      : Megha Sonavane
// Description :Dijkstra's algorithm
//==========================================================
==================

#include <iostream>
#include"Graph.h"
using namespace std;

int main() {
	cout<<"***Shortest Path Finding***"<<endl;
	Graph g;
	//creation of graph
	g.createGraph();
	//display graph
	g.display();
	//find shortest path
	g.dijkstra();
	return 0;
}
```

- **Output:**

```
***Shortest Path Finding***
Enter 1:Directed graph
0:Undirected graph::0

Enter number of vertices (Max 20):5
Enter number of edges (Max 20):7

====================================
Enter landmarks for following vartices:
Vertex 0:hospital
Vertex 1:temple
Vertex 2:school
Vertex 3:bank
Vertex 4:library

You entered::
0:hospital
1:temple
2:school
3:bank
4:library

====================================
Enter edges of graph
Enter vertex 1 vertex 2 and weight:0
1
7
Enter vertex 1 vertex 2 and weight:1
2
5
Enter vertex 1 vertex 2 and weight:2
3
2
Enter vertex 1 vertex 2 and weight:3
4
12
Enter vertex 1 vertex 2 and weight:4
0
9
```

```
Enter vertex 1 vertex 2 and weight:1
4
3
Enter vertex 1 vertex 2 and weight:2
4
1
Graph matix is:
0 7 0 0 9
7 0 5 0 3
0 5 0 2 1
0 0 2 0 12
9 3 1 12 0
Enter source vertex:0
Vertex      Path        Distance
hospital    hospital    32767
temple      hospital    7
school      hospital    32767
bank        hospital    32767
library     hospital    9

Selected vertex:1
Cost:7

Selected vertex:4
Cost:9

Selected vertex:2
Cost:10

Shortest path from source to all verices:

temple:: Distance:7 Path: temple <- hospital
school::  Distance:10 Path: school <- library <- hospital
bank::  Distance:12 Path: bank <- school <- library <- hospital
library::  Distance:9 Path: library <- hospital
```