Experiment topic: Finding the shortest path based on Dijsktra algorithm

Experiment purpose :

1. Master the adjacency matrix representation of graphs and the algorithms for creating graphs using adjacency matrix representation.
2. Master Dijstra's algorithm for finding the shortest path.

Experiment content:

Problem Description

A map includes n cities, assuming there are m paths (directed graph) between cities, and the length of each path is known. Given a starting city and an ending city on the map, use the Dijsktra algorithm to find the shortest path between the starting point and that in the map.

input requirements

Multiple sets of data, each set of m+3 rows. The first line contains two certificates n and m, respectively representing the number n of cities and the number m of routes. The second line has n characters representing the name of each city. From the third line to the m+2th line, each line has two characters a and b and an integer d, which means that there is a road with a distance of d from city a to city b. The last line is two characters, which represent the starting point and ending point of the city where the shortest path is to be found. When both n and m are equal to 0, the input ends.

output requirements

Output 2 lines for each set of data. The first line is an integer, which is the length of the shortest path from the start point to the end point. The second row is a string of strings representing the path. Separate every two characters with a space.

CODE:

Dijsktra.cpp

```
#include "stdafx.h"
#include "Graph.h"

int main()
{
	int n,m,a,b;
	char p,q;
	while(cin>>n>>m)
	{
		if(n==0&&m==0)
			break;
		AMGraph G;
		CreateUDN(G,n,m);
		cin>>p>>q;
		a=LocateVex(G,p);
		b=LocateVex(G,q);
funtion(G,a,b,n);
	}
	return 0;
}
```

Graph.cpp

```
#include "StdAfx.h"
#include "Graph.h"
```

```
int S[MVNum],D[MVNum],Path[MVNum];

int LocateVex(AMGraph G, char u)
{
	int i;
	for(i=0;i<G.vexnum;++i)
	{
		if(G.vexs[i]==u)
			return i;
	}
	return -1;
}

int CreateUDN(AMGraph &G,int n,int m)
{ //Using adjacency matrix notation, create an undirected network G
	int i,k,j,w;
	char v1, v2;
	G.vexnum=n;G.arcnum=m; //Enter the total number of vertices, the total number of edges
	for(i=0;i<G.vexnum;i++)
		cin>>G.vexs[i]; //Input the information of the points in sequence
	for(i=0;i<G.vexnum;++i)
	{
		for(j=0;j<G.vexnum;++j)
			G.arcs[i][j]=MaxInt; //Initialize the adjacency matrix, and set the weights of the edges
to the maximum value MaxInt
	}
	for(k=0;k<G.arcnum;++k) //construct adjacency matrix
	{
		cin>>v1>>v2>>w; //Enter the vertex and weight of an edge
		i=LocateVex(G,v1);
		j=LocateVex(G,v2); //Determine the positions of v1 and v2 in G, that is, the subscript of the
vertex array
		G.arcs[i][j]=w; //Set the weight of edge <v1,v2> to w
		G.arcs[j][i]=G.arcs[i][j]; //Set the weight of <v2,v1> of <v1,v2> to w
	}
	return OK;
}

void ShortestPath_DIJ(AMGraph G,int v0)
{ //Use the Dijkstra algorithm to find the shortest path from the v0 vertex of the directed graph G to the
rest of the vertices
	int i,w;
	int n=G.vexnum; //n is the number of vertices in G
	for(int v=0;v<n;++v) //n vertices are initialized sequentially
	{
		S[v]=false; //S is initially empty
		D[v]=G.arcs[v0][v]; //Initialize the shortest path length from v0 to each vertex as the weight
on the arc
		if(D[v]<MaxInt)
			Path[v]=v0; //If there is an arc between v0 and v, set the predecessor of v to v0
		else
			Path[v]=-1; //If there is no arc between v0 and v, set the predecessor of v to -1
	}
	S[v0]=true; //Add v0 to S
	D[v0]=0; //The distance between the source point and the source point is 0
//Initialization is complete, start the main loop, find the shortest path from v0 to a vertex v each time,
and calculate in turn
```

```
    for(i=1;i<n;i++) //calculate the remaining n-1 vertices sequentially
    {
        int min=MaxInt;
        for(w=0;w<n;++w)
        {
            if(!S[w]&&D[w]<min)
            {
                v=w;
                min=D[w]; //Select a current shortest path, the end point is v
            }
        }
            S[v]=true; //Add v to S
            for(w=0;w<n;w++) //Update the shortest path length from v0 to all vertices on the set
VS
            {
                if(!S[w]&&(D[v]+G.arcs[v][w]<D[w]))
                {
                    D[w]=D[v]+G.arcs[v][w]; //Update D[w]
                    Path[w]=v; //Update the predecessor of w to v
                }
            }
    }
}

void print(AMGraph G,int t)
{
    if(Path[t]==t)
    {
        cout<<G.vexs[t];
        return;
    }
    print(G,Path[t]);
    cout<<" "<<G.vexs[t];
    return;
}

void function(AMGraph G,int x,int y,int n)
{
    int j;
ShortestPath_DIJ(G,x);
    Path[0]=0;
    if(D[y]<MaxInt)
        cout<<D[y]<<endl;
    else
        cout<<"-1"<<endl;
    print(G,y);
    cout<<endl;
}
```

StdAfx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
//    Dijskta.pch will be the pre-compiled header
//    stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

```
// TODO: reference any additional headers you need in STDAFX.H
// and not in this file
```

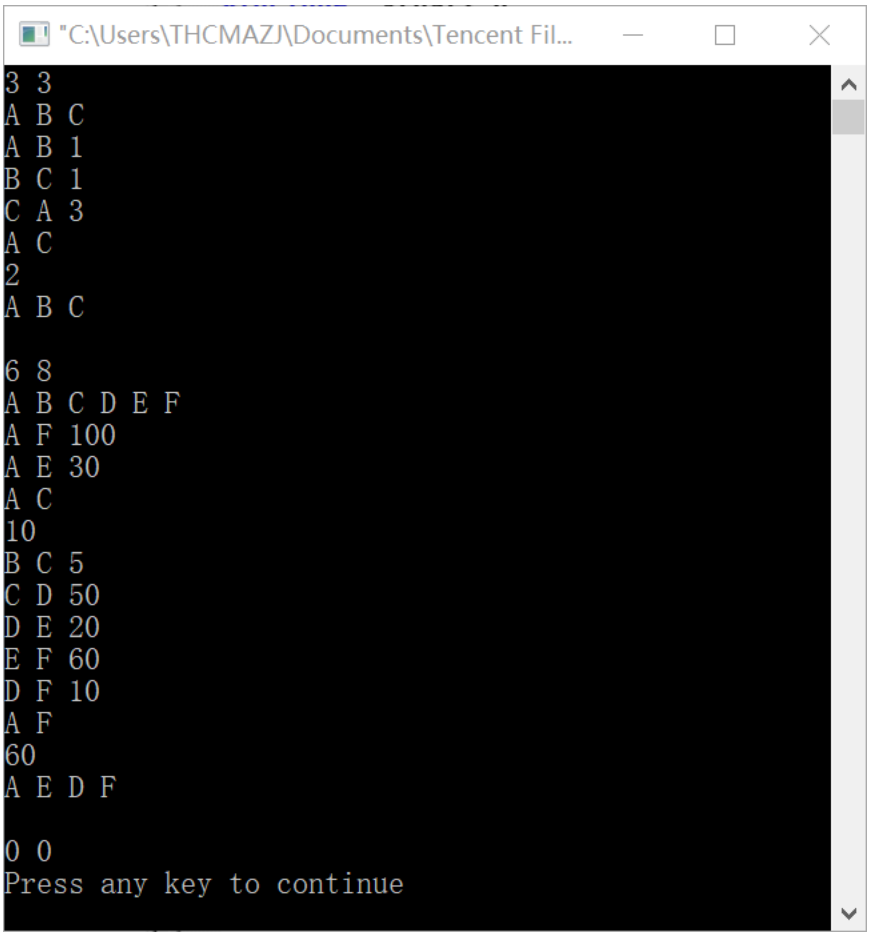
Graph.h

```
#define MaxInt 32767 //Indicates the maximum value
#define MVNum 100 //Indicates the number of vertices
typedef char VerTexType; //Assume the data type of the vertex is character
typedef int ArcType; //Assume the variable weight type is integer
typedef struct
{
    VerTexType vexs[MVNum]; //vertex table
    ArcType arcs[MVNum][MVNum]; //adjacency matrix
    int vexnum, arcnum; //The current number of points and edges of the graph
}AMGraph;

void ShortestPath_DIJ(AMGraph G,int v0); //Use Dijkstra algorithm to find the shortest path from v0
vertex of directed graph G to other vertices
int CreateUDN(AMGraph &G,int n,int m); //Using adjacency matrix notation, create an undirected
network G
int LocateVex(AMGraph G, char u);
void print(AMGraph G,int t);
void function(AMGraph G,int x,int y,int n);
```

StdAfx.h

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#if !defined(AFX_STDAFX_H__3672FEDB_35AE_4596_BEF5_7DD2EA9384A0__INCLUDED_)
#define AFX_STDAFX_H__3672FEDB_35AE_4596_BEF5_7DD2EA9384A0__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define WIN32_LEAN_AND_MEAN             // Exclude rarely-used stuff from Windows headers

#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#define OK 1
#define ERROR 0
using namespace std;

// TODO: reference additional headers your program requires here

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif
// !defined(AFX_STDAFX_H__3672FEDB_35AE_4596_BEF5_7DD2EA9384A0__INCLUDED_)
```

Test Results:


"C:\Users\THCMAZJ\Documents\Tencent Fil...
3 3
A B C
A B 1
B C 1
C A 3
A C
2
A B C
6 8
A B C D E F
A F 100
A E 30
A C
10
B C 5
C D 50
D E 20
E F 60
D F 10
A F
60
A E D F
0 0
Press any key to continue