```cpp
//日期: 2018/ 时间:
#include <stdio.h>
#include <stdlib.h>
#include <algorithm>
#include <vector>
using namespace std;
const int maxv = 1000;          //最大顶点数
const int INF = 1000000000;     //很大的数

//**************************新增边权**************************************
#define p4
#ifdef p4
int n,G[maxv][maxv];            //n为顶点数，maxv为最大顶点数
int d[maxv];                    //起点到达各点的最短路径长度
bool vis[maxv] = {false};
int pre[maxv];              //pre[v]表示从起点到顶点v的最短路径上v的前一个顶点

int cost[maxv][maxv];
int c[maxv];

void Dijkstra(int s){           //s为起点
    fill(d,d+maxv,INF);         //将整个数组赋值为INF
    fill(c,c+maxv,INF);
    for(int i=0;i<n;i++) pre[i]=i;
    d[s] = 0;                   //起点s到达自身的距离为0

    for(int i=0;i<n;i++){       //循环n次
        int u = -1,MIN = INF;
        for(int j=0;j<n;j++){   //找到未访问的顶点中d[]最小的
            if(vis[j] == false && d[j] < MIN){
                u = j;
                MIN = d[j];
            }
        }

        //找不到小于INF的d[u]，说明剩下的顶点和起点s不连通
        if(u == -1) return;

        vis[u] = true;          //标记u为已访问
        for(int v=0;v<n;v++){   //优化d[u]
            //如果v未访问 && u能到达v && 以u为中介点可以使d[v]更优
            if(vis[v] == false && G[u][v] != INF){
                if(d[u]+G[u][v] < d[v]){
                    d[v] = d[u] + G[u][v];
                    c[v] = c[v] + cost[u][v];
                    pre[v] = u;
                } else if(d[u]+G[u][v] == d[v] && c[u] + cost[u][v] < c[v]){
                    c[v] = c[v] + cost[u][v];
                    pre[v] = u;
                }
            }
        }

    }
}

```

```
57  void DFS(int s,int v){          //s为起点编号，v为当前访问的顶点编号
58      if(v == s){             //如果当前已经到达起点s，则输出起点并返回
59          printf("%d\n",s);
60          return;
61      }
62      DFS(s,pre[v]);
63      printf("%d\n",v);
64  }
65
66  #endif
67  //*************************************************************************
    *
68
69
70  //**************************输出最短路径
    ***************************************
71  //#define p3
72  #ifdef p3
73  int n,G[maxv][maxv];            //n为顶点数，maxv为最大顶点数
74  int d[maxv];                    //起点到达各点的最短路径长度
75  bool vis[maxv] = {false};
76  int pre[maxv];              //pre[v]表示从起点到顶点v的最短路径上v的前一个顶点
77
78  void Dijkstra(int s){           //s为起点
79      fill(d,d+maxv,INF);         //将整个数组赋值为INF
80      for(int i=0;i<n;i++) pre[i]=i;
81      d[s] = 0;                   //起点s到达自身的距离为0
82
83      for(int i=0;i<n;i++){       //循环n次
84          int u = -1,MIN = INF;
85          for(int j=0;j<n;j++){   //找到未访问的顶点中d[]最小的
86              if(vis[j] == false && d[j] < MIN){
87                  u = j;
88                  MIN = d[j];
89              }
90          }
91
92          //找不到小于INF的d[u]，说明剩下的顶点和起点s不连通
93          if(u == -1) return;
94
95          vis[u] = true;          //标记u为已访问
96          for(int v=0;v<n;v++){   //优化d[u]
97              //如果v未访问 && u能到达v && 以u为中介点可以使d[v]更优
98              if(vis[v] == false && G[u][v] != INF && d[u]+G[u][v] < d[v]){
99                  d[v] = d[u] + G[u][v];
100                 pre[v] = u;
101             }
102         }
103
104     }
105 }
106
107 void DFS(int s,int v){          //s为起点编号，v为当前访问的顶点编号
108     if(v == s){             //如果当前已经到达起点s，则输出起点并返回
109         printf("%d\n",s);
```

```
110         return;
111     }
112     DFS(s,pre[v]);
113     printf("%d\n",v);
114 }
115
116 #endif
117 //*****************************************************************************
    *
118
119 //**************************邻接矩阵版*************************************
120 //#define p1
121 #ifdef p1
122 int n,G[maxv][maxv];            //n为顶点数，maxv为最大顶点数
123 int d[maxv];                    //起点到达各点的最短路径长度
124 bool vis[maxv] = {false};
125
126 void Dijkstra(int s){           //s为起点
127     fill(d,d+maxv,INF);         //将整个数组赋值为INF
128     d[s] = 0;                   //起点s到达自身的距离为0
129
130     for(int i=0;i<n;i++){       //循环n次
131         int u = -1,MIN = INF;
132         for(int j=0;j<n;j++){   //找到未访问的顶点中d[]最小的
133             if(vis[j] == false && d[j] < MIN){
134                 u = j;
135                 MIN = d[j];
136             }
137         }
138
139         //找不到小于INF的d[u]，说明剩下的顶点和起点s不连通
140         if(u == -1) return;
141
142         vis[u] = true;          //标记u为已访问
143         for(int v=0;v<n;v++){   //优化d[u]
144             //如果v未访问 && u能到达v && 以u为中介点可以使d[v]更优
145             if(vis[v] == false && G[u][v] != INF && d[u]+G[u][v] < d[v]){
146                 d[v] = d[u] + G[u][v];
147             }
148         }
149
150     }
151 }
152 #endif
153 //*****************************************************************************
    *
154 //**************************邻接表版*****************************************
155 //#define p2
156 #ifdef p2
157
158 struct Node{
159     int v,dis;                  //v为边的目标顶点，dis为边权
160 };
```

```
161 vector<Node> adj[maxv];
162 int n;                          //n为顶点数，图G使用邻接表实现
163 int d[maxv];                     //起点到达各点的最短路径长度
164 bool vis[maxv] = {false};
165
166 void Dijkstra(int s){
167     fill(d,d+maxv,INF);
168     d[s] = 0;
169
170     for(int i=0;i<n;i++){
171         int u = -1,MIN = INF;
172         for(int j=0;j<n;j++){
173             if(vis[j] == false && d[j] < MIN){
174                 u=j;
175                 MIN = d[j];
176             }
177         }
178
179         //找不到小于INF的节点，说明剩下的顶点和起点s 不连通
180         if(u == -1) return;
181
182         vis[u] = true;
183         for(int j=0;j<adj[u].size();j++){
184             int v = adj[u][j].v;
185             if(vis[v] == false && d[u]+adj[u][j].dis < d[v]){
186                 d[v] = d[u]+adj[u][j].dis;
187             }
188         }
189     }
190 }
191
192 #endif
193 //*************************************************************************
    *
194
195 int main(){
196
197
198     return 0;
199 }
200
201
```