# 补充

## 1086 最短路径

### 题意理解

这一题给定一个无向图，要求求出结点1到结点n的最短路径，如果可达，还要输出最短路径个数与一条最短路径的线路。

### 源代码

```
#include <bits/stdc++.h>

using namespace std;

const int INF = 0x3f3f3f3f;
const int maxn = 102;
typedef pair<int, int> P;

struct edge                                  //定义边的结构体
{
    int to, cost;
    edge(int k1, int k2): to{k1}, cost{k2} {}
};

int d[maxn], pre[maxn], state[maxn];
bool used[maxn];
vector<edge> G[maxn];
int V, E;
vector<int> path;
priority_queue< P, vector<P>, greater<P> > que;

void Dijkstra(int s)                                //进行搜索
{
    memset(d, INF, sizeof(d));
    d[s] = 0;
    state[s] = 1;
    que.push(P(0, s));

    while(!que.empty())
    {
        P p = que.top(); que.pop();
        int v = p.second;
        if(d[v] < p.first) continue;
        for(unsigned int i = 0; i < G[v].size(); i++)
        {
            edge e = G[v][i];
            if(d[e.to] == d[v] + e.cost)             //如果有相同长度
                state[e.to] += state[v];
            else if(d[e.to] > d[v] + e.cost)
            {
                state[e.to] = state[v];
                pre[e.to] = v;
                d[e.to] = d[v] + e.cost;
```

```
                que.push(P(d[e.to], e.to));
            }
        }
    }
}
int main()
{
    int x, y, z;
    scanf("%d %d", &V, &E);
    for(int i = 1; i <= E; i++)
    {
        scanf("%d %d %d", &x, &y, &z);
        G[x].push_back(edge(y, z));
        G[y].push_back(edge(x, z));
    }

    Dijkstra(1);

    if(d[V] == INF)
        printf("%d\n%d\n%d\n", -1, 0, -1);
    else
    {
        printf("%d\n%d\n", d[V], state[V]);
        int t = V;
        for(; t; t = pre[t]) path.push_back(t);
        for(unsigned i = path.size() - 1; i > 0; i--)
            printf("%d ", path[i]);
        printf("%d\n", path[0]);
    }
}
```

## 程序调试经验总结

这题要求的是两个结点之间的最短路径，所以采用Dijkstra算法来计算最短路径，同时在求解过程中要维护最短路径个数和前驱结点(用来记录路径)。

为了节约空间，使代码更加方便，我用了邻接表来存储边的信息，每一个结点的边存在一个vector里，如果用邻接矩阵的话，首先是100*100的数组造成内存浪费，在查找的时候时间效率也不高，并且按照题意说明，相同的边可能有很多条，所以还要开一个mark数组来记录重边的个数，较为繁琐。

在进行Dijkstra算法时，相等情况需要特别维护最短路径的条数(用state数组记录)：

```
if(d[e.to] == d[v] + e.cost)              //如果有相同长度
       state[e.to] += state[v];
```

若是小于时，要同时维护最短路径条数和前驱结点：

```
else if(d[e.to] > d[v] + e.cost)
{
    state[e.to] = state[v];
    pre[e.to] = v;
    d[e.to] = d[v] + e.cost;
    que.push(P(d[e.to], e.to));
 }
```

最后在输出答案时，要将记录的前驱结点倒着输出即可。