题意

- 给定有向图
- 给定s，e，k
- 求s到e的第k短路

思路分析

- 使用A*寻路算法
- 设估值函数为f[x] = h[x]+g[x]
  h[x] 为 从s出发到x点走过的路程
  g[x]为当前点到终点的最短距离
- 每次优先选取估值函数最小的可扩展点进行BFS扩展（优先队列实现）

> 所谓A* 就是启发是搜索,说白了就是给BFS搜索一个顺序使得搜索更加合理减少无谓的搜索..如何来确定搜索的顺序？..也就是用一个值来表示这个值为f[x]..每次搜索取f[x]最小的拓展…那么这个f[x]=h[x]+g[x]其中这个h[x]就是当前搜索时的代价,如求K段路这个就是前一个点的h[x']+边的长度…而g[x]是一个估价函数..估价函数要小于是对当前点到目标的代价的估计..这个估计必须小于等于实际值~~否则会出错…A* 的关键也就是构造g[x],
>
> 而这里要说的求K短路一种方法,就是用BFS+A* 来搜索的过程.g[x]的设定为到这个点到目标点的最短路径,显然其实小于等于实际值的.h[x]就是搜索到这个点的代价.用一个优先队列来做..每次取出h[x]+g[x]最小的点来拓展…拓展也就是通过这点来更新其能直接经过一条边到达的点..这里做好一个新点就丢进优先队列里去..反正总会从对首弹出h[x]+g[x]最小的点..可以想一下…如果当前取出的优先队列头是一个点e并且是第一次取出h..那么就找到了一条从源点到h的最短路径..这里其实很djikstra的感觉差不多..如果第二次在对头取出了e..则是找到了一条从源点到h的第二短路径..依次类推..第几次从对头弹出e..则找到了从源点到e的第几短路径..
> 那要是本身就不存在K短路呢？？那就是e拓展不到K但是其他点很有可能一直打圈圈无限下去…这里就要用个条件来判断一下…首先在找某个点作为优先队列头出现了几次就用了一个计数器times[]..所求的点times[e]==k就代表得到了解..如果当前想拓展的点times[]>k就没必要拓展了..因为这个点已经是求到k+1短路了..从这个点继续往下搜肯定得到的是大于等于k+1短路的路径…就像1->2有3条路..2->3有2条路..那1->3有6条路的概念差不多..没必要对其进行拓展了..
> 还有一点要特别注意的就是题目要求必须要走..也就是s==e时..k++….

代码

```
#include<iostream>
#include<queue>
#include<vector>
using namespace std;
typedef long long LL;
struct fx{
    int h,g,p;
    bool operator < (fx a) const
    {
        return a.h+a.g<h+g;
    }
};
struct cre{
    int to;
    int val;
};
int n,m;
vector<cre>edge[1005];
vector<cre>_edge[1005];
int s,e,k;
int tim[1005];
priority_queue<fx> pq;
int dis[1005];
int book[1005];
int Astar()
{
    while(!pq.empty())
        pq.pop();
    fx now;
    now.p = s;
    now.g = 0;
    now.h = 0;
    pq.push(now);
    while(!pq.empty())
    {
        now = pq.top();
        pq.pop();
        tim[now.p]++;
        if(now.p == e&&tim[e] == k) return now.g+now.h;
        if(tim[now.p]>k) continue;
        fx t;
        for(int i=0;i<edge[now.p].size();i++)
        {
```

```
            cre v = edge[now.p][i];
            t.p = v.to;
            t.h = now.h +v.val;
            t.g = dis[v.to];
            pq.push(t);
        }
    }
    return -1;
}
void dij()
{
    for(int i=1;i<=n;i++)
    {
        dis[i] = 99999999;
    }
    dis[e] = 0;
    for(int p=1;p<=n;p++)
    {
        int now = 0;
        for(int i=1;i<=n;i++)
        {
            if( ( now==0||dis[i]<dis[now] ) &&book[i]==0)
            {
                now = i;
            }
        }
        book[now] = 1;
        for(int i=0;i<_edge[now].size();i++)
        {
            cre v = _edge[now][i];
            if(dis[now]+v.val<dis[v.to])
            {
                dis[v.to] = dis[now]+v.val;
            }
        }
    }
    return ;
}
int main()
{
    ios::sync_with_stdio(false);
    cin>>n>>m;
    int x,y,w;
    for(int i=1;i<=m;i++)
    {
        cin>>x>>y>>w;
```

```
        cre temp;
        temp.to =y;
        temp.val = w;
        edge[x].push_back(temp);
        temp.to = x;
        _edge[y].push_back(temp);
    }
    cin>>s>>e>>k;
    if(s==e)  k++;
    dij();
    int ans = Astar();
    cout<<ans<<endl;
}
```