高级数据结构-SPFA算法

信智学院　陈琰宏

主要内容

01 SPFA算法

02 SPFA应用

03
1 [1147] 德克萨斯长角牛（热浪）
2 [3566] Roadblocks

04
3 [3579] 农场派对
4 [3578] 通信线路 Telephone Lines
5 [3582] 新年好

05
6 [3585] 道路和航线
7 [3622]双调路径

SPFA算法

SPFA算法是求单源最短路径的一种算法，它是Bellman-ford的队列优化，它是一种十分高效的最短路算法。很多时候，给定的图存在负权边，这时类似Dijkstra等算法便没有了用武之地，而Bellman-Ford算法的复杂度又过高，这时SPFA算法便派上用场了。SPFA的复杂度大约是O(kE),k是每个点的平均进队次数(一般的，k是一个常数，在稀疏图中小于2)。

但是，SPFA算法稳定性较差，在稠密图中SPFA算法时间复杂度会退化。

SPFA算法

实现方法：

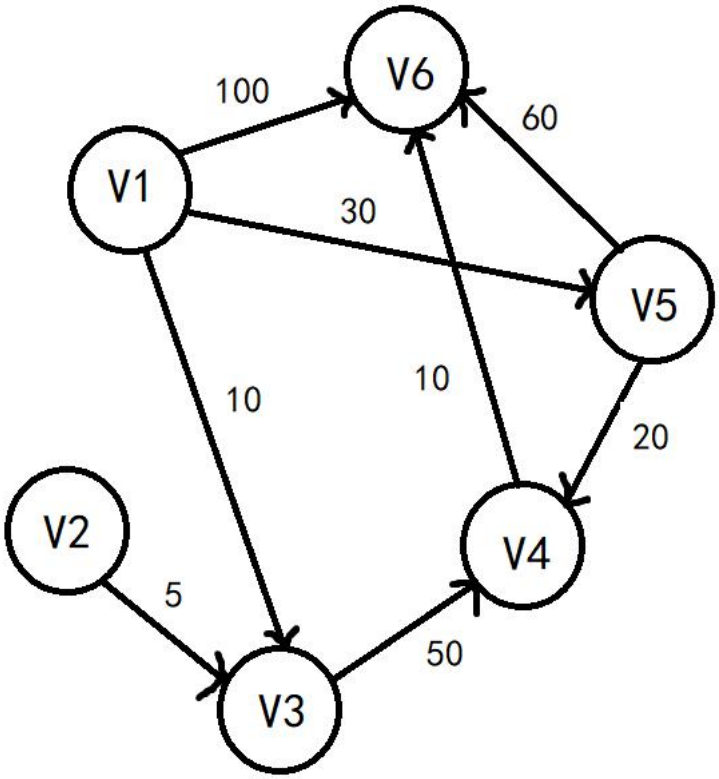
1：建立一个队列，初始时队列里只有起始点，

2：再建立一个表格记录起始点到所有点的最短路径（该表格的初始值要赋为极大值，该点到他本身的路径赋为0）。

3：然后执行松弛操作，用队列里有的点去刷新起始点到所有点的最短路，如果刷新成功且被刷新点不在队列中则把该点加入到队列最后。

4：重复执行3直到队列为空。

SPFA算法模拟

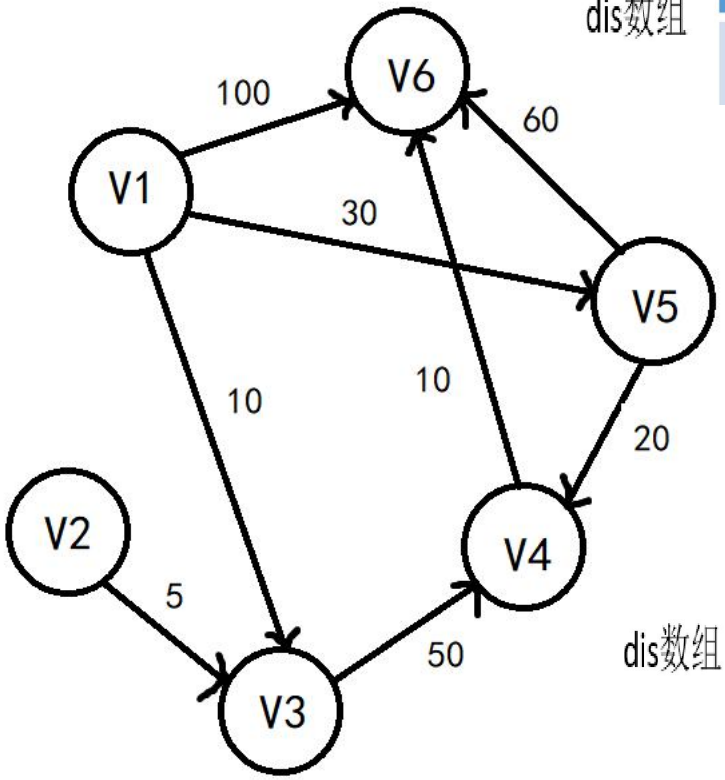


首先我们先初始化数组dis如下图所示：（除了起点赋值为0外，其他顶点的对应的dis的值都赋予无穷大，这样有利于后续的松弛）

dis数组

v1	v2	v3	v4	v5	v6
0	∞	∞	∞	∞	∞

此时，我们还要把v1入队列：{v1}
现在进入循环，直到队列为空才退出循环。



dis数组

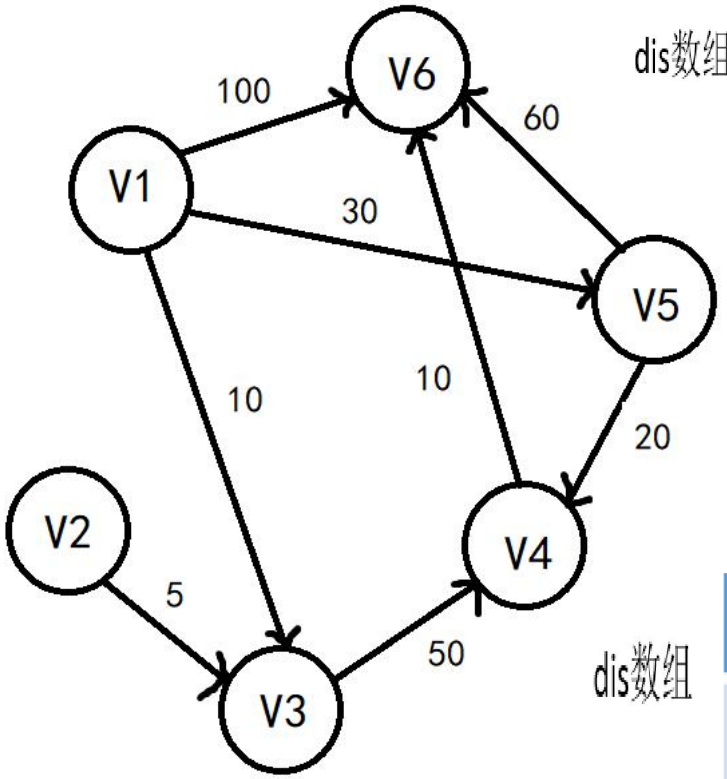
v1	v2	v3	v4	v5	v6
0	∞	∞	∞	∞	∞

第一次循环：
首先，队首元素出队列，即是v1出队列，然后，对以v1为弧尾的边对应的弧头顶点进行松弛操作，可以发现v1到v3，v5，v6三个顶点的最短路径变短了，更新dis数组的值，得到如下结果：

dis数组

v1	v2	v3	v4	v5	v6
0	∞	10	∞	30	100

我们发现v3，v5，v6都被松弛了，而且不在队列中，所以要它们都加入到队列中：{v3，v5，v6}



dis数组

v1	v2	v3	v4	v5	v6
0	∞	10	∞	30	100

第二次循环

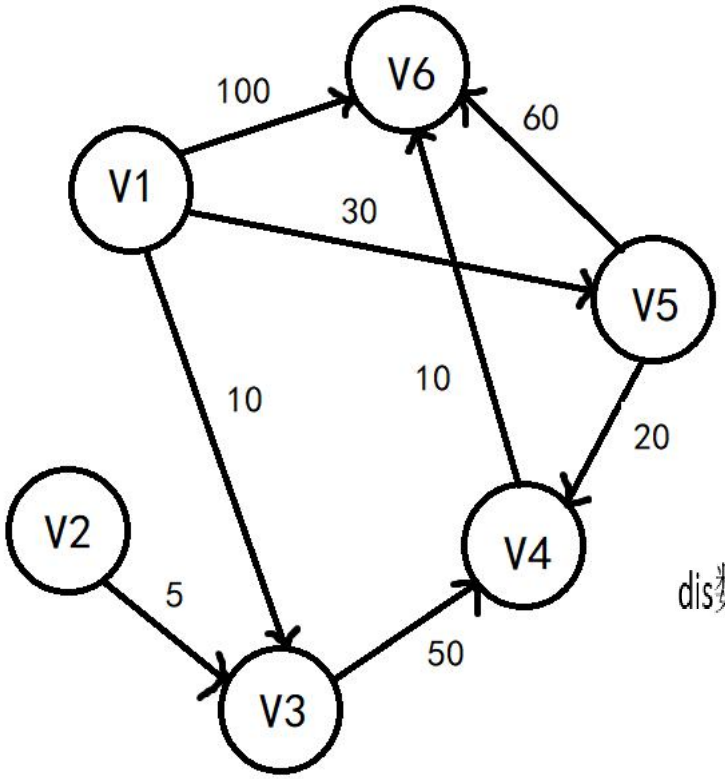
此时，队首元素为v3，v3出队列，然后，对以v3为弧尾的边对应的弧头顶点进行松弛操作，可以发现v1到v4的边，经过v3松弛变短了，所以更新dis数组，得到如下结果：

dis数组

v1	v2	v3	v4	v5	v6
0	∞	10	60	30	100

此时只有v4对应的值被更新了，而且v4不在队列中，则把它加入到队列中：

{v5, v6, v4}



dis数组

v1	v2	v3	v4	v5	v6
0	∞	10	60	30	100

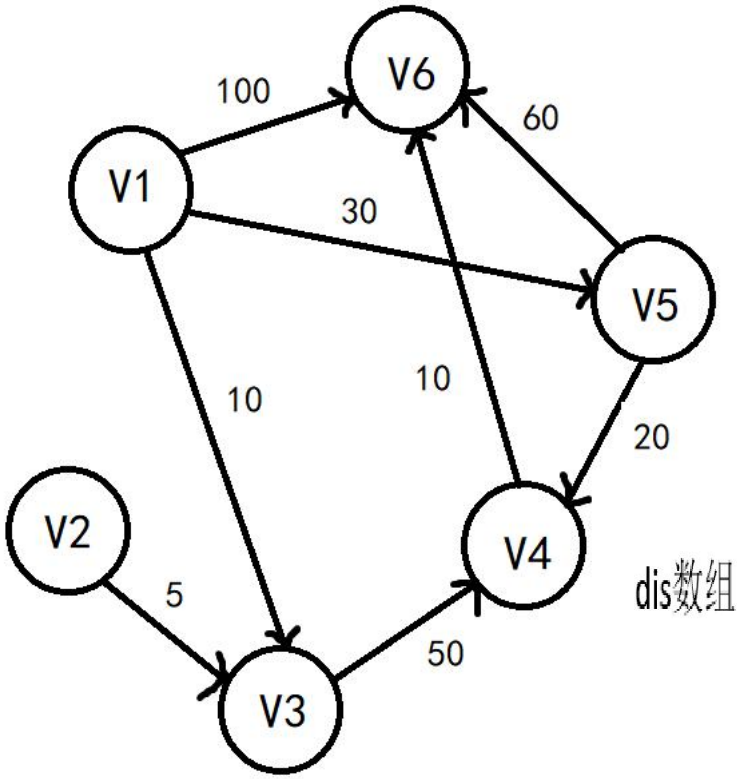
第三次循环

此时，队首元素为v5，v5出队列，然后，对以v5为弧尾的边对应的弧头顶点进行松弛操作，发现v1到v4和v6的最短路径，经过v5的松弛都变短了，更新dis的数组，得到如下结果：

dis数组

v1	v2	v3	v4	v5	v6
0	∞	10	50	30	90

我们发现v4、v6对应的值都被更新了，但是他们都在队列中了，所以不用对队列做任何操作。队列值为：{v6，v4}



第四次循环

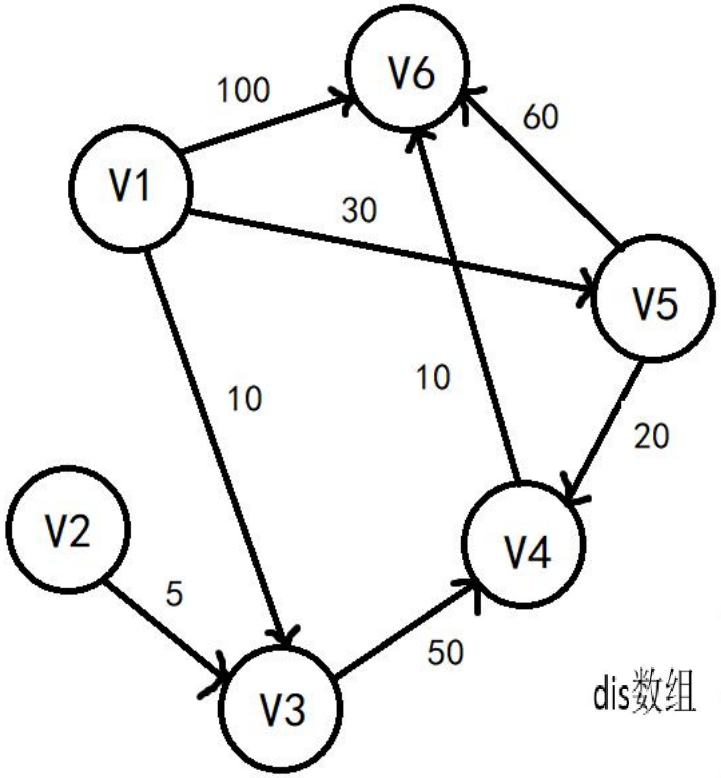
此时，队首元素为v6，v6出队列，然后，对以v6为弧尾的边对应的弧头顶点进行松弛操作，发现v6出度为0，所以我们不用对dis数组做任何操作，其结果和上图一样，队列同样不用做任何操作，它的值为：{v4}

dis数组

v1	v2	v3	v4	v5	v6
0	∞	10	50	30	90

dis数组

v1	v2	v3	v4	v5	v6
0	∞	10	50	30	90



第五次循环

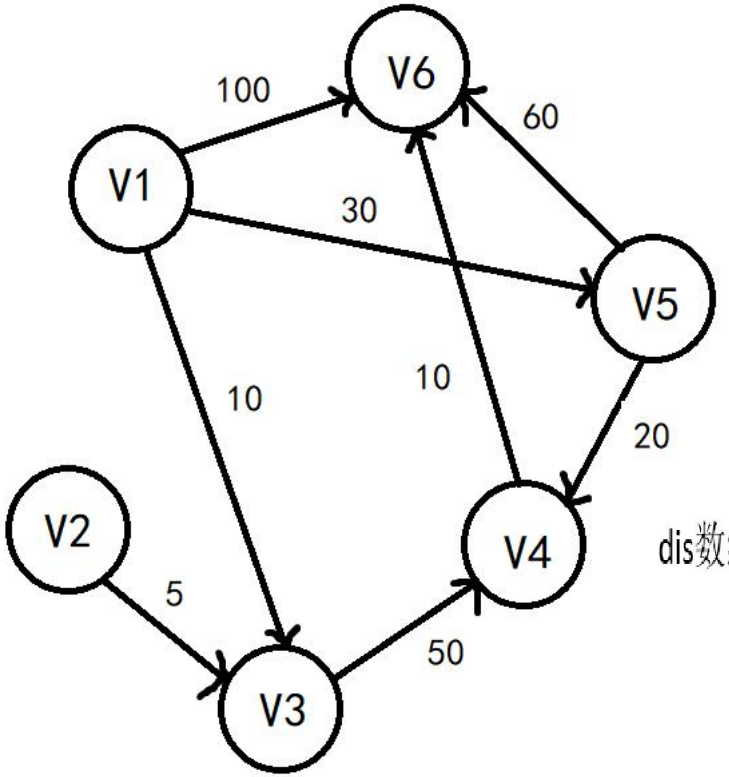
此时，队首元素为v4，v4出队列，然后，对以v4为弧尾的边对应的弧头顶点进行松弛操作，可以发现v1到v6的最短路径，经过v4松弛变短了，所以更新dis数组，得到如下结果：

dis数组

v1	v2	v3	v4	v5	v6
0	∞	10	50	30	60

dis数组

v1	v2	v3	v4	v5	v6
0	∞	10	50	30	60



第六次循环

此时，队首元素为v6，v6出队列，然后，对以v6为弧尾的边对应的弧头顶点进行松弛操作，发现v6出度为0，所以我们不用对dis数组做任何操作，其结果和上图一样，队列同样不用做任何操作。所以此时队列为空。

dis数组

v1	v2	v3	v4	v5	v6
0	∞	10	50	30	60

由于队列为空，队列循环结果，此时我们也得到了v1到各个顶点的最短路径的值了，它就是dis数组各个顶点对应的值

1 [1147] 德克萨斯长角牛（热浪）

题目描述：

德克萨斯纯朴的民众们这个夏天正在遭受巨大的热浪！Farmer John此时承担起向德克萨斯运送大量的营养冰凉的牛奶的重任，以减轻德克萨斯人忍受酷暑的痛苦。FJ已经研究过可以把牛奶从威斯康星运送到德克萨斯州的路线。这些路线包括起始点和终点先一共经过T(1 <= T <= 2,500)个城镇，方便地标号为1到T。除了起点和终点外地每个城镇由两条双向道路连向至少两个其它地城镇。每条道路有一个通过费用（包括油费，过路费等等）。

给定一个地图，包含C (1 <= C <= 6,200)条直接连接2个城镇的道路。每条双向道路由两个端点Rs和Re (1 <= Rs <= T; 1 <= Re <= T)，和花费Ci(1 <= Ci <= 1,000)组成。求从起始城镇Ts (1 <= Ts <= T)到终点城镇Te(1 <= Te <= T)最小的总费用。如样例这个有7个城镇的地图。城镇5是奶源，城镇4是终点（括号内的数字是道路的通过费用）。经过路线5-6-1-4总共需要花费3 (5-> 6) + 1 (6-> 1) + 3 (3-> 4) = 7的费用。

[1147] 德克萨斯长角牛（热浪）

样例输入：

7 11 5 4
2 4 2 √
1 4 3
7 2 2
3 4 3
5 7 5
7 3 3
6 1 1
6 3 4
2 4 3
5 6 3
7 2 1 √

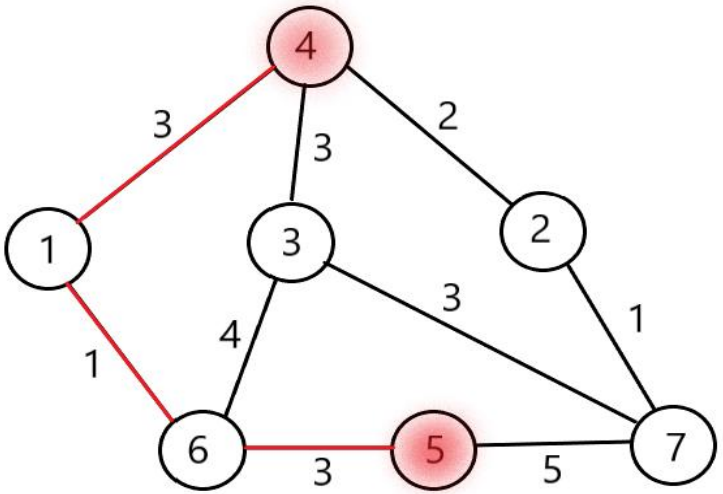
样例输出：

7 (3+1+3)

输入描述：

第一行：4个由空格隔开的整数：城镇数T，道路数C，起点Ts，终点Te。

第2到第C+1行：第i+1行描述第i条道路，每行有3个由空格隔开的整数：道路的两个端点Rs和Re，通过费用Ci。



输出描述：

第一行：一个单独的整数表示起点Ts到终点Te的最短路的长度（即花费的最少费用）。数据保证至少存在一条道路。

[1147] 德克萨斯长角牛（热浪）

这是一道很明显的最短路模板题。。

```
#include<bits/stdc++.h>
using namespace std;
#define MAXN 2510
struct node{
    int v;
    int w;
};
vector<node>nod[MAXN];
queue<int>q;
int dis[MAXN];//记录从起点到每个点的最短距离
bool vis[MAXN];//标记已在队列中的点
int main(){
    int i,n,m,st,ed,u,v,w;
    cin>>n>>m>>st>>ed;//n座城市，m条道路，st:起点，ed:终点
    for(i=1;i<=m;i++){
        scanf("%d%d%d",&u,&v,&w);
        nod[u].push_back({v,w});//双向道路
        nod[v].push_back({u,w});
    }
    spfa(st,ed);
    return 0;
}
```

[1147] 德克萨斯长角牛（热浪）

```
12  void spfa(int st,int ed){
13      memset(dis,127,sizeof(dis));
14      q.push(st);
15      vis[st]=1;
16      dis[st]=0;//起点到起点的距离为0
17      while(!q.empty()){
18          int u=q.front();
19          q.pop();
20          vis[u]=0;//出队，取消标记
21          for(int i=0;i<nod[u].size();i++){//从u点连向的每个点
22              int v=nod[u][i].v;
23              int w=nod[u][i].w;
24              if(dis[v]>dis[u]+w){//如果最短路径可以更新，则更新
25                  dis[v]=dis[u]+w;
26                  if(vis[v]==0){//如果被更新了的点不在队列中，则入队
27                      vis[v]=1;
28                      q.push(v);
29                  }
30              }
31          }
32      }
33      cout<<dis[ed];//输出答案，起点到终点的最短路
34  }
```

2 [3566] Roadblocks

题目描述：

某街区共有R条道路，N个路口。道路可以双向通行。问1号路口到N号路口的次短路是多少？次短路是比最短路长度长的次短的路径。同一条边可以经过多次。

输入描述：

第一行为N和R。

接下来R行为道路，输入u,v,w，意思是结点u和v的权值为w。

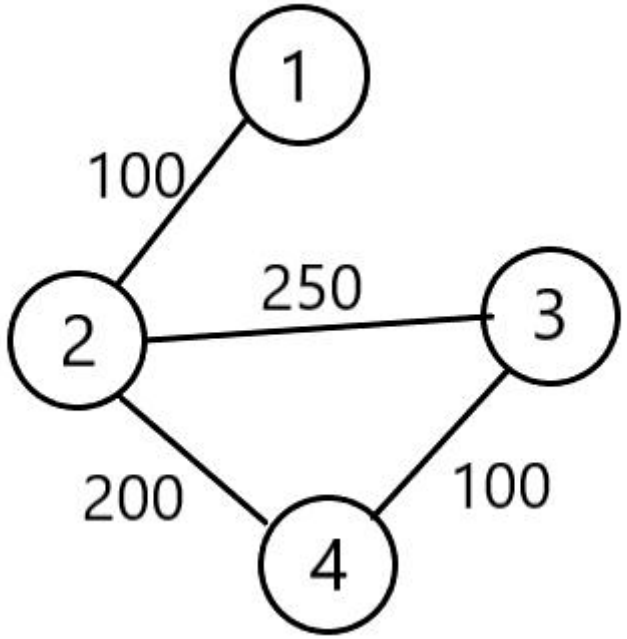
输出描述：

1到N的次短路长度。

2 [3566] Roadblocks

样例输入：

```
4 4          R条路，N个路口
1 2 100      u、v、w
2 4 200
2 3 250
3 4 100
```



样例输出：

```
450          1走到N的次短路长度
```

最短路：1 -> 2 -> 4 (长度为100+200=300)

次短路：1 -> 2 -> 3 -> 4 (长度为100+250+100=450)

2 [3566] Roadblocks

分析：

可以通过求最短路得到次短路长度。

1到n的次短路长度必然产生于：从1走到x的最短路 + edge[x][y] + y到n的最短路。

首先预处理好1到每一个节点的最短路，和n到每一个节点的最短路，然后枚举每一条边作为中间边（x，y）或者（y，x），如果加起来长度等于最短路长度则跳过，否则更新。

2 [3566] Roadblocks

```cpp
#include<bits/stdc++.h>
using namespace std;
#define MAXN 5010
struct node{
    int v;
    int w;
};
vector<node>nod[MAXN];
queue<int>q;
int n,vis[MAXN],dis1[MAXN],disN[MAXN];
//n个节点
//dis1[MAXN]: 1到每一个节点的最短路
//disN[MAXN]: n到每一个节点的最短路
```

[3566] Roadblocks

```
14  void spfa(int st,int (&dis)[MAXN]){//st: 出发点
15      memset(dis,127,sizeof(dis));
16      dis[st]=0;//到出发点的距离为0
17      q.push(st);
18      vis[st]=1;//标记出发点已入队
19      while(!q.empty()){
20          int u=q.front();
21          q.pop();
22          vis[u]=0;//点u已出队，标记为0
23          for(int i=0;i<nod[u].size();i++){
24              int v=nod[u][i].v;
25              int w=nod[u][i].w;
26              if(dis[u]+w<dis[v]){//如果新的路径比原来的路径短，更新最短路
27                  dis[v]=dis[u]+w;
28                  if(vis[v]==0){//如果更新后的点不在队列中，则入队
29                      vis[v]=1;
30                      q.push(v);
31                  }
32              }
33          }
34      }
35  }
```

[3566] Roadblocks

```
int main(){
    int i,j,r,u,v,w;
    cin>>n>>r;//r条路，n个路口
    for(i=1;i<=r;i++){//输入数据，双向加边
        scanf("%d%d%d",&u,&v,&w);
        nod[u].push_back({v,w});
        nod[v].push_back({u,w});
    }
    spfa(1,dis1);//求1到每一个节点的最短路
    spfa(n,disN);//求n到每一个节点的最短路
    int ans=inf;//ans：次短路长度
    //枚举每一条边作为中间边（可能会有边被重复枚举，如x->y和y->x是同一条双向边）
    for(i=1;i<=n;i++){//枚举每一个点
        for(j=0;j<nod[i].size();j++){//枚举该点连接的边
            u=i;            //边的左端点
            v=nod[i][j].v;//右端点
            w=nod[i][j].w;//边权
            int l=dis1[u]+w+disN[v];
            if(ans>l&&dis1[n]!=l)ans=l;//更新次短路（dis1[n]是1到n的最短路）
        }
    }
    cout<<ans;
    return 0;
}
```

3 [3579] 农场派对

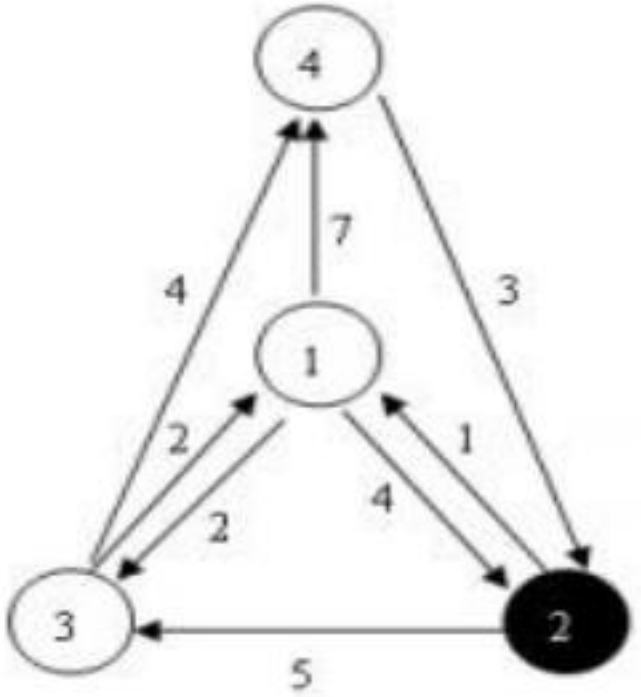
题目描述：寒假到了，N头牛都要去参加一场在编号为X（1≤X≤N）的牛的农场举行的派对（1≤N≤1000），农场之间有M（1≤M≤100000）条有向路，每条路长Ti（1≤Ti≤100）。每头牛参加完派对后都必须回家，无论是去参加派对还是回家，每头牛都会选择最短路径，求这N头牛的最短路径（一个来回）中最长的一条路径长度。

【输入格式】
第一行三个整数N, M, X;
第二行到第M+1行：每行有三个整数Ai，Bi，Ti ，表示有一条从Ai农场到Bi农场的道路，长度为Ti。
【输出格式】
一个整数，表示最长的最短路得长度。

[3579] 农场派对

样例输入
4 8 2
1 2 4
1 3 2
1 4 7
2 1 1
2 3 5
3 1 2
3 4 4
4 2 3

样例输出
10



分析

分析：

注意：同一条边来去可能路程不同（如：1->2：4；　　2->1:1）
可以通过建图，正反跑spfa，最后遍历相加。

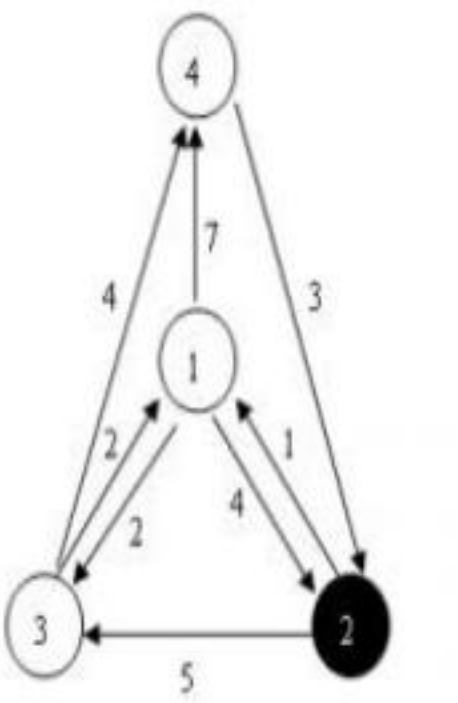
3579 农场派对

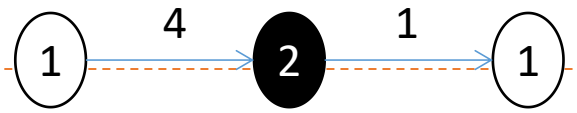
样例输入

4 8 2
1 2 4
1 3 2
1 4 7
2 1 1
2 3 5
3 1 2
3 4 4
4 2 3

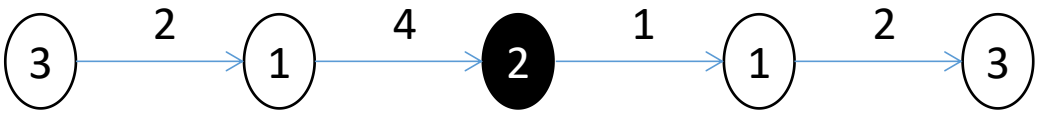
样例输出

10



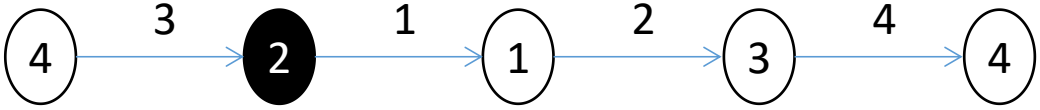


5



1

0



9

3579 农场派对

分析

题目要找其他点到终点X的最短路径和终点X到其他点的最短路径。
通过SPFA算法可以求出X到各个点之间的最短距离，也就是牛回去的最短距离。
而SPFA跑一遍反向图就可以求出各个点到X的最短距离，即可求出答案。
因此我们需要分别存储一遍正向图和反向图。

[3579] 农场派对

```
11  void spfa(int k){//k=1时正向图，k=2时反向图
35  int main()
36  {
37      cin>>n>>m>>l;
38      memset(f,0x3f3f3f,sizeof(f));
39      for(int a=1;a<=m;a++)
40      {
41          int x,y,z;
42          cin>>x>>y>>z;
43          edge cmp;
44          cmp.v=y;cmp.u=z;
45          G[1][x].push_back(cmp);//正向建图
46          cmp.v=x;
47          G[2][y].push_back(cmp);//反向建图
48      }
49      spfa(1);//跑正向图
50      spfa(2);//跑反向图
51      for(int a=1;a<=n;a++){
52          if(a==l)continue;
53          ans=max(f[1][a]+f[2][a],ans);//找最长的
54      }
55      cout<<ans<<endl;
56      return 0;
57  }
```

```
3   struct edge
4   {
5       int u,v;//到达点、距离
6   };
7   vector<edge> G[3][1001];//动态数组存边
8   queue<int> Q;
9   int n,m,f[3][1001],l,ans;
10  bool vis[1001];
```

[3579] 农场派对

```
11 void spfa(int k){//k=1时正向图，k=2时反向图
12     memset(vis,0,sizeof(vis));//初始化false
13     vis[l]=true;
14     f[k][l]=0;
15     Q.push(l);
16     while(!Q.empty())
17     {
18         int news=Q.front();
19         Q.pop();
20         vis[news]=false;
21         for(int i=0;i<G[k][news].size();i++)
22         {
23             int v=G[k][news][i].v,u=G[k][news][i].u;
24             if(f[k][v]>f[k][news]+u)
25             {
26                 f[k][v]=f[k][news]+u;//计算最短
27                 if(!vis[v]){//不在，入队
28                     vis[v]=true;
29                     Q.push(v);
30                 }
31             }
32         }
33     }
34 }
```

4 [3578] 通信线路 Telephone Lines

FJ的农场周围分布着N(1<=N<=1,000)根按1..N顺次编号的废弃的电话线杆。一共P(1<=P<=10,000)对电话线杆间可以拉电话线。FJ的任务仅仅是找一条将1号和N号电话线杆连起来的路径。电信公司同意免费为FJ连结K(0<=K<N)对由FJ指定的电话线杆。对于此外的那些电话线，FJ需要为它们付的费用，等于其中最长的电话线的长度。

输入：

第1行：3个用空格隔开的整数：N，P，以及K

* 第2..P+1行：第i+1行为3个用空格隔开的整数：A_i，B_i，L_i

输出：

第1行：输出1个整数，为FJ在这项工程上的最小支出。如果任务不可能完成，输出-1

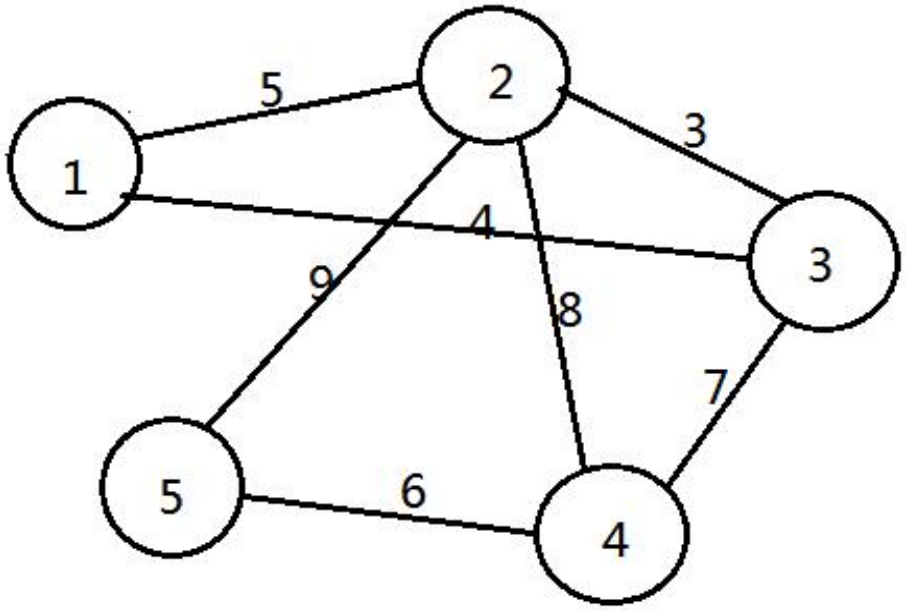
4 [3578] 通信线路 Telephone Lines

输入样例:

5 7 1
1 2 5
3 1 4
2 4 8
3 2 3
5 2 9
3 4 7
4 5 6

输出样例:

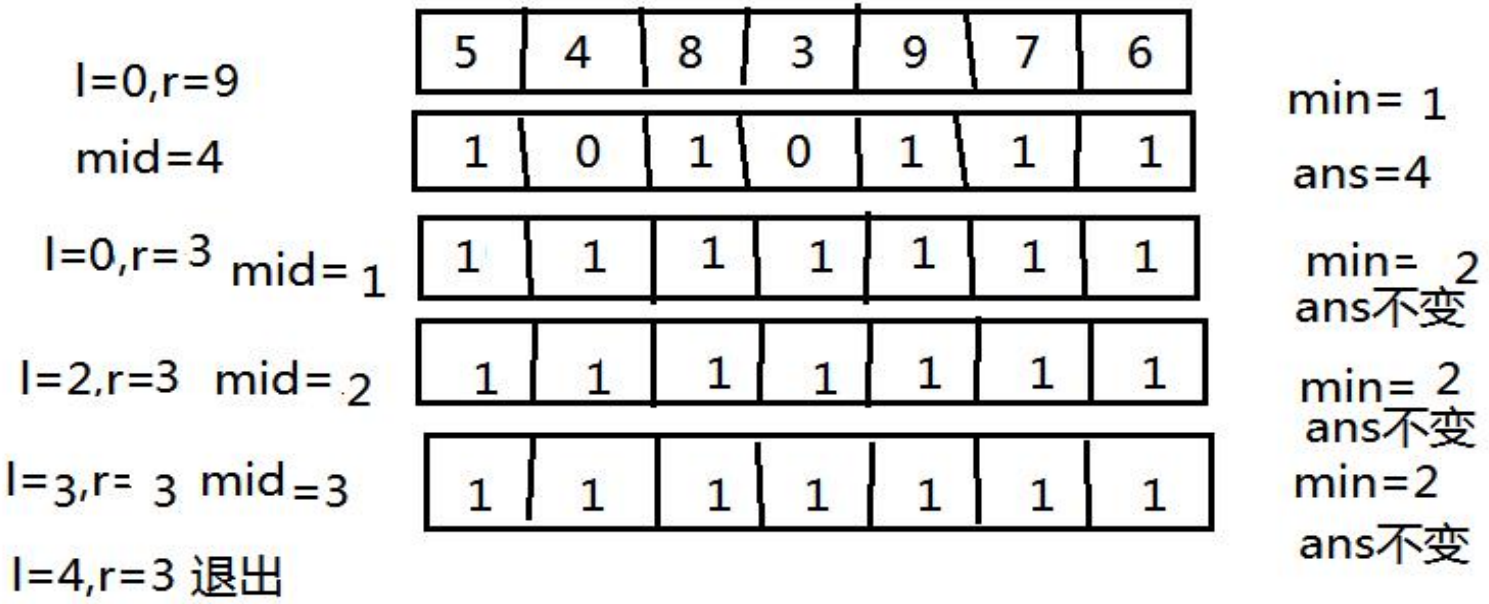
4



4 [3578] 通信线路 Telephone Lines

分析：

二分出mid，然后把长度大于mid的边看成1，长度小于等于mid的边看成0，跑最短路（Spfa）。判断距离是否小于等于k。



[3578] 通信线路 Telephone Lines

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int M=10005;
using namespace std;
int r[M],inq[M],n,p,k;
int head[M],to[M],val[M],next[M],cnt;
queue<int>q;
bool judge(int mid)
{
    int u,i,d;
    for(i=1;i<=n;i++)
        r[i]=0x3fffffff;
    r[1]=0;//到点i的最短路径中边权值大于mid的边数
    inq[1]=1;//标记点i是否入队
    q.push(1);//点1入队
    while(!q.empty())
    {
        u=q.front();//取队首数据
        q.pop();//删除队首数据
        inq[u]=0;
        for(i=head[u];i;i=next[i])//遍历与点u相连的各条边
        {
            d=(val[i]>mid);//判断点u到第i条边的终点之间的边权值是否大于mid
```

[3578] 通信线路 Telephone Lines

```
25                  if(r[to[i]]>r[u]+d)
26                  {
27                      r[to[i]]=r[u]+d;
28  //点1到点u的权值大于mid的边数算上当前边，如果小于点1到终点的权值大于mid的边数，更新数据
29                      if(!inq[to[i]])
30                      {
31                          inq[to[i]]=1;
32                          q.push(to[i]);//未入队的点入队
33                      }
34                  }
35              }
36      }
37      return r[n]<=k;//最短路中权值大于mid的边的数量小于等于k，返回1，反之返回0
38  }
39  void add(int x, int y,ll z)
40  {
41      to[++cnt]=y;//边的终点
42      val[cnt]=z;//边的权值
43      next[cnt]=head[x];//当前连接点x的上一条边的编号
44      head[x]=cnt;//当前连接点x的最后一条边的编号
45  }
46  int main()
47  {
48      int i,x,y,l=0,r=0,mid,ans=-1,z;
```

[3578] 通信线路Telephone Lines

```
48      int i,x,y,l=0,r=0,mid,ans=-1,z;
49      scanf("%d %d %d",&n,&p,&k);
50      for(i=0;i<p;i++)
51      {
52          scanf("%d %d %d",&x,&y,&z);
53          add(x,y,z);
54          add(y,x,z);//建立无向邻接表
55          r=max(r,z);//找到最大权值
56      }
57      while(l<=r)//二分法
58      {
59          mid=(l+r)>>1;//即（l+r）/2
60          if(judge(mid))
61          //假设mid为所求答案，判断最短路径中大于mid的边是否小于等于k（k为可免费的边数）
62          {
63              ans=mid;
64              r=mid-1;//k还有余量，mid前还有可实现条件的答案（往前找最小的答案）
65          }
66          else l=mid+1;//k不够了，mid不符合题目条件，从mid后寻找答案
67      }
68      printf("%d\n",ans);
69      return 0;
70  }
```

5 [3582] 新年好

重庆城里有n个车站,m条双向公路连接其中的某些车站.每两个车站最多用一条公路连接,从任何一个车站出发都可以经过一条或者多条公路到达其他车站,但不同的路径需要花费的时间可能不同.在一条路径上花费的时间等于路径上所有公路需要的时间之和.
佳佳的家在车站1,他有五个亲戚,分别住在车站a,b,c,d,e.过年了,他需要从自己的家出发,拜访每个亲戚(顺序任意),给他们送去节日的祝福.怎样走,才需要最少的时间?
输入
第一行:n,m为车站数目和公路的数目.
第二行:a,b,c,d,e， 为五个亲戚所在车站编号。
以下m行,每行三个整数x,y,t,为公路连接的两个车站编号和时间.
输出
仅一行,包含一个整数T,为最少的总时间.

[3582] 新年好

样例输入
6 5
2 3 4 5 6
6 1 6
2 1 2
3 1 3
1 4 4
1 5 5
样例输出
34

详细分析

由于她是依次不断拜访，即到达一家后立即前往下一家，所以我们首先可以以这6个点为起点依次跑最短路，得到以这6个点为起点的最短路径。换句话说，我们知道了每两个地方间的最短距离。接着要定的就是拜访顺序了。
由于只有六处，且出发点固定，我们就可以考虑深搜，将所有的可能的拜访顺序求出，同时将每个顺序的时间求出，取出所有中的最小值即可。

大体思路如上，下面分析一下代码细节：
1、有五处亲戚，为了便于编写，用一个一维数组来存储。同时，因为起点佳佳在1号车站，所以我们可以在这个数组的第0个位置存入佳佳家。
2、求最短路的过程可以使用SPFA或者堆优化dijkstra。
3、由于需要求解以六个车站为起点的最短路径，所以存最短路径时用一般的一维数组是肯定不行的。所以我们考虑用二维数组，其中第一维可以存储出发亲戚家的编号。
4、由于求最小值，所以深搜时可以剪枝：当前所用时间已经大于过去最小值时，就可以return。

[3582]新年好——代码

```
#include<bits/stdc++.h>
using namespace std;
const int M=200010,INF=0x3f3f3f3f;
const int N=50010;
vector<int>g[N],w[N];//存图
struct node{//dis为该点到起点的时间，num是该点的编号
    int dis,num;
};
priority_queue<node>que;//小顶堆
bool operator<(const node &a,const node &b)
{
    return a.dis>b.dis;
}
int vis[6][N],ra[6],ans=INF;
//vis存储以6个亲戚为出发点的最短路径 ，ra存储每个亲戚的车站号
bool st[N];//搜索时记录已经拜访过了的亲戚
```

[3582]新年好——代码

```
16  void dijkstra(int x)//x为亲戚编号
17  {
18      int start=ra[x];//start为起点车站
19      vis[x][start]=0;//起点车站到自己的时间为0
20      que.push(node{0,start});     //起点入队
21      while(!que.empty())
22      {
23          node now=que.top();
24          que.pop();
25          if(now.dis!=vis[x][now.num])//如果该点被取过了就continue
26          continue;
27          for(int i=g[now.num].size()-1;i>=0;i--)
28          {//遍历邻接表
29              int to=g[now.num][i];
30              if(vis[x][to]>vis[x][now.num]+w[now.num][i])
31              {
32                  vis[x][to]=vis[x][now.num]+w[now.num][i];//更新时间
33                  que.push(node{vis[x][to],to});
34              }
35          }
36      }
37  }
38  void dfs(int cur,int cost ,int pos)
39  //cur为当前拜访的亲戚数目 ， cost为当前所耗费的时间，pos为出发点
40  {
41      if(cost>ans) return;//如果当前时间大于过去最小值，返回
```

```
{
    if(cost>ans) return;//如果当前时间大于过去最小值，返回
    if(cur==5)
    {
        ans=min(ans,cost);//取最小值
        return ;
    }
    for(int i=1;i<=5;i++)
    {
        if(!st[i])//当这个亲戚没有拜访过时
        {
            st[i]=true;
            dfs(cur+1,cost+vis[pos][ra[i]],i);
            st[i]=false;
        }
    }
}
int main()
{
    int n,m;//n为车站数目 ， m公路条数
    cin>>n>>m;
    memset(vis,0x3f,sizeof(vis));
    for(int i=1;i<=5;i++)
        cin>>ra[i];//输入5个亲戚所在的车站
    ra[0]=1;
    for(int i=1;i<=m;i++)
    {//存图
        int x,y,k;
        cin>>x>>y>>k;
        g[x].push_back(y);
        w[x].push_back(k);
        g[y].push_back(x);
        w[y].push_back(k);
    }
    for(int i=0;i<=5;i++)
        dijkstra(i);//遍历求得以每个亲戚为起点的最短路径
    dfs(0,0,0);
    cout<<ans<<endl;
    return 0;
}
```

6 [3585] 道路和航线

Farmer John正在一个新的销售区域对他的牛奶销售方案进行调查。他想把牛奶送到T个城镇（1 <= T <= 25,000），编号为1T。这些城镇之间通过R条道路（1 <= R <= 50,000，编号为1到R）和P条航线（1 <= P <= 50,000，编号为1到P）连接。每条道路i或者航线i连接城镇A_i（1 <= A_i <= T)到B_i（1 <= B_i <= T)，花费为C_i。对于道路，0 <= C_i <= 10,000;然而航线的花费很神奇，花费C_i可能是负数(-10,000 <= C_i <= 10,000)。道路是双向的，可以从A_i到B_i，也可以从B_i到A_i，花费都是C_i。然而航线与之不同，只可以从A_i到B_i。事实上，由于最近恐怖主义太嚣张，为了社会和谐，出台 了一些政策保证：如果有一条航线可以从A_i到B_i，那么保证不可能通过一些道路和航线从B_i回到A_i。由于FJ的奶牛世界公认十分给力，他需要运送奶牛到每一个城镇。他想找到从发送中心城镇S(1 <= S <= T) 把奶牛送到每个城镇的最便宜的方案，或者知道这是不可能的。

6 [3585] 道路和航线

输入
第1行：四个空格隔开的整数: T, R, P, and S * 第2到R+1行：三个空格隔开的整数（表示一条道路）：A_i, B_i 和 C_i * 第R+2到R+P+1行：三个空格隔开的整数（表示一条航线）：A_i, B_i 和 C_i

输出
第1到T行：从S到达城镇i的最小花费，如果不存在输出”NO PATH’’。

样例输入

6 3 3 4
1 2 5
3 4 5
5 6 10
3 5 -100
4 6 -100
1 3 -10

样例输出

NO PATH
NO PATH
5
0
-95
-100

分析：经典最小路问题
Spfa+SLF优化

也可以用SPFA加SLF优化：
SLF优化就是small label first 优化
利用一个双端队列，比较即将插入队首的和队首的dis值，如果比队首小，就加到队首，否则直接放到队尾可以减少一些无用的状态

思路

因为有负数，我们用spfa求最短路，但单纯这样会超时，我们要加一个slf优化 slf算法就是如果新的点的路程小于spfa中的队列的对首的点路程时把它放到对首，反之放到队尾。不断用最小的点，能较快跑完。

分析
求起点到各个点的最短距离，尝试了用SPFA，但是这题数据卡SPFA

注意到无向边边权是非负的，这提示我们可以在无向边上跑最短路。并且我们可以知道，如果将无向边连接的点看作一个整体，最后图中只剩下有向边的话，这个图就是一个有向无环图。
所以我们就可以分开考虑，首先求出若干个由无向边组成的连通块，对于块内的点，通过堆优化的dijkstra算法更新最短路；然后在块与块之间，类似于拓扑排序，一层一层地进行更新，最后就可以求出源点s到所有点的最短路了。

拓扑排序+dijkstra

分析

求起点到各个点的最短距离，尝试了用SPFA，但是这题数据卡SPFA

注意到无向边边权是非负的，这提示我们可以在无向边上跑最短路。并且我们可以知道，如果将无向边连接的点看作一个整体，最后图中只剩下有向边的话，这个图就是一个有向无环图。
所以我们就可以分开考虑，首先求出若干个由无向边组成的连通块，对于块内的点，通过堆优化的dijkstra算法更新最短路；然后在块与块之间，类似于拓扑排序，一层一层地进行更新，最后就可以求出源点s到所有点的最短路了。

拓扑排序+dijkstra

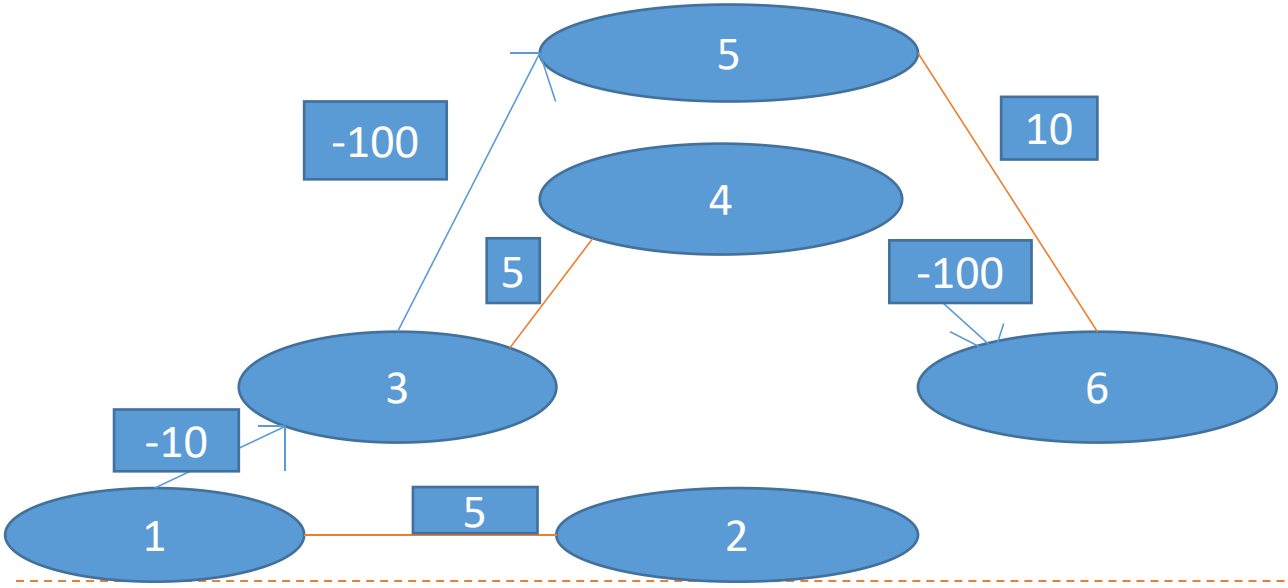
样例输入

```
6 3 3 4
1 2 5
3 4 5
5 6 10
3 5 -100
4 6 -100
1 3 -10
```

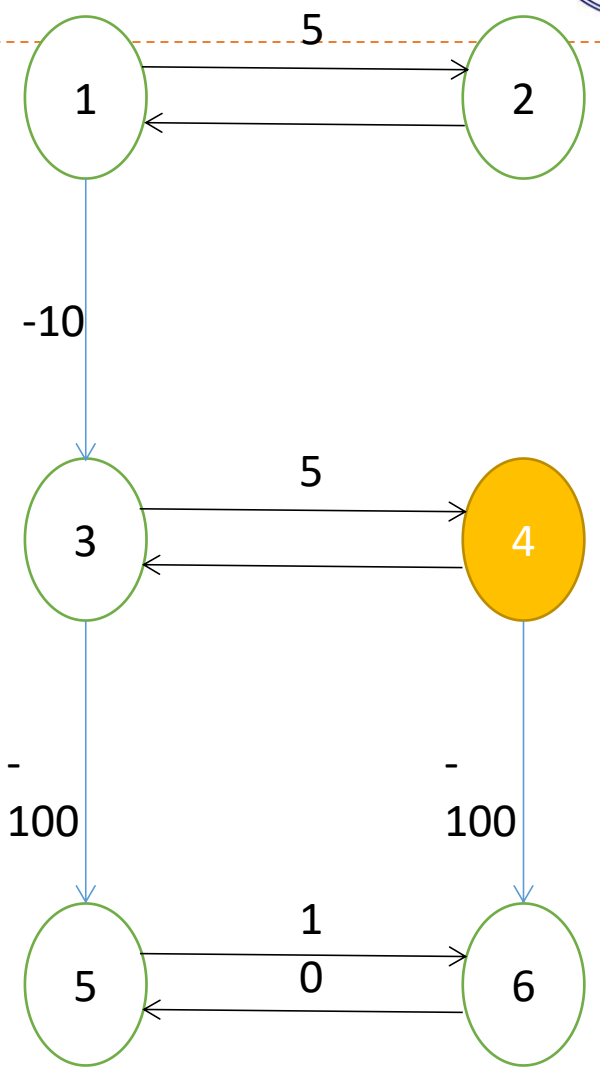
样例输出

```
NO PATH
NO PATH
5
0
-95
-100
```

橙线为道路
蓝线为航线



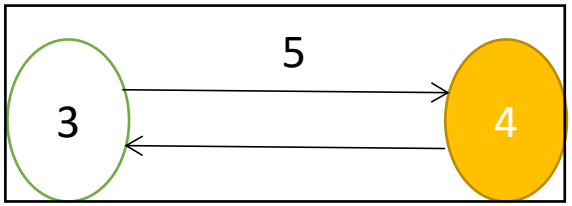
样例输入	样例输出
6 3 3 4	NO PATH
1 2 5	NO PATH
3 4 5	5
5 6 10	0
3 5 -100	-95
4 6 -100	-100
1 3 -10	



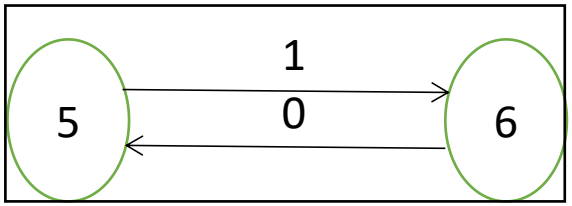
block
[1]

如何划分连通块

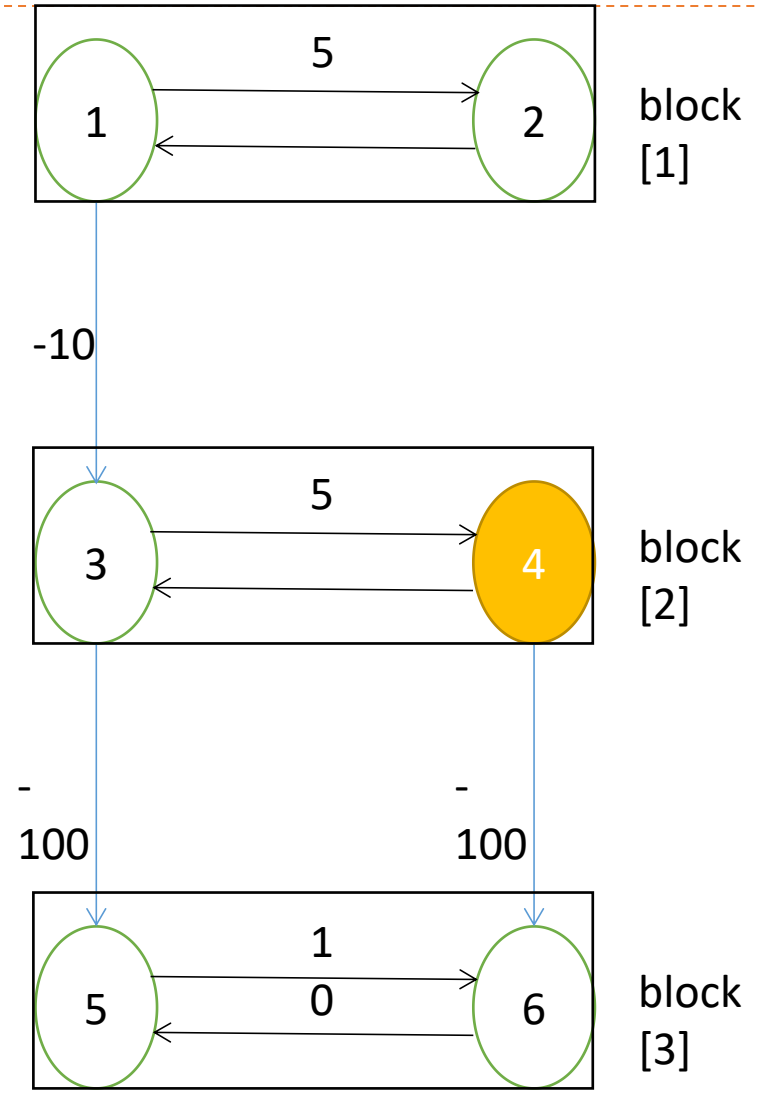
可以用在输入了道路之后深度搜索每一个点，因为此时还未读入航线，所以连通块是分开的。



block
[2]



block
[3]



可以将此图划分为三个连通块（block[1] block[2] block [3]）,建立一个预处理队列，将s存在的连通块以及入度为0的连通块放入预处理队列中，逐个取出进行处理，先在连通块内部进行dijkstra，当涉及到跨越连通块的边（也就是航线）时，若处理完此边并删去后另一个连通块的入度为0，则将此连通块放入预处理队列

3585 道路和

```cpp
#include<bits/stdc++.h>
using namespace std;
#define INF 0x3f3f3f3f
typedef long long ll;
const int N=50005,M=100005;
int n,r,p,s;
struct edge{//edge存边
    int v,next,w;
}e[M<<1];
struct node{
    int cost,u;
    bool operator < (const node &A)const{//用于优先队列排序，cost小的优先
        return cost>A.cost;
    }
};
int head[N],tot;
void adde(int u,int v,int w){//加边
    e[tot].v=v,e[tot].w=w,e[tot].next=head[u],head[u]=tot++;
}
bool vis[N];
int cnt;
int bl[N],in[N];//bl存每个点所在的连通块，in存每个连通块的入度
int cost[N];//到每个点的最小花费
vector<int>block[N];//存每个连通块的元素

void dfs(int u)//dfs划分连通块
{
    vis[u]=1;
    bl[u]=cnt;
    block[cnt].push_back(u);
    for(int i=head[u];i!=-1;i=e[i].next)
    {
        int v=e[i].v;
        if(vis[v])continue;
        dfs(v);
    }
}

void solve()//拓扑排序+Dijkstra
{
    memset(vis,0,sizeof(vis));
    memset(cost,0x7f,sizeof(cost));
    cost[s]=0;
    queue<int>Q;
    Q.push(bl[s]);
    for(int i=1;i<=cnt;i++)if(!in[i])Q.push(i);
    priority_queue<node>q;
    while(!Q.empty())//拓扑排序循环
    {
        int blo=Q.front();Q.pop();
        int blosize=block[blo].size();//将连通块内所有点放入q队列用于Dijks
        for(int i=0;i<blosize;i++)q.push(node{cost[block[blo][i]],block[bl
        while(!q.empty())//Dijkstra
        {
            node now=q.top();q.pop();
            int u=now.u;
            if(vis[u])continue;
            vis[u]=1;
            for(int i=head[u];i!=-1;i=e[i].next)
            {
                int v=e[i].v;
                if(cost[v]>cost[u]+e[i].w)
                {
                    cost[v]=cost[u]+e[i].w;
                    if(bl[v]==bl[u])q.push(node{cost[v],v});
                }
                if(bl[v]!=bl[u]&&(--in[bl[v]])==0)Q.push(bl[v]);//拓扑排序
            }
        }
    }
}

```

```
72
73  int main()
74  {
75      scanf("%d %d %d %d",&n,&r,&p,&s);
76      memset(head,-1,sizeof(head));
77      for(int i=1;i<=r;i++)//输入道路
78      {
79          int a,b,c;
80          scanf("%d %d %d",&a,&b,&c);
81          adde(a,b,c);
82          adde(b,a,c);
83      }
84      for(int i=1;i<=n;i++)//划分连通块
85      {
86          if(!vis[i])
87          {
88              cnt++;
89              dfs(i);
90          }
91      }
92      for(int i=1;i<=p;i++)//输入航线
93      {
94          int a,b,c;
95          scanf("%d %d %d",&a,&b,&c);
96          adde(a,b,c);
97          in[bl[b]]++;
98      }
99      solve();//拓扑排序+Dijkstra
100     for(int i=1;i<=n;i++)
101     {
102         if(cost[i]>INF)printf("NO PATH\n");
103         else printf("%d\n",cost[i]);
104     }
105     return 0;
106 }
```

7 [3622]双调路径

题目描述：

选择最佳路径是很现实的问题。城市的道路是双向的，每条道路有固定的旅行时间以及需要支付的费用。路径由连续的道路组成。总时间是各条道路旅行时间的和，总费用是各条道路所支付费用的总和。同样的出发地和目的地，如果路径A比路径B所需时间少且费用低，那么我们说路径A比路径B好。对于某条路径，如果没有其他路径比它好，那么该路径被称为最优双调路径。这样的路径可能不止一条，或者说根本不存在。给出城市交通网的描述信息，起始点和终点城市，求最优双条路径的条数。城市不超过100个，边数不超过300，每条边上的费用和时间都不超过100。

读入网络，计算最小路径的总数。费用时间都相同的两条最小路径只算作一条。你只要输出不同种类的最小路径数即可。

[3622]双调路径

输入：

第一行给出有多少个点,多少条边,开始点及结束点. 下面的数据用于描述这个地图

接下来每行描述了一条道路的信息：两个端点 p,r，费用 c，以及时间 t；

两个城市之间可能有多条路径连接。

输出：

有多少条最优双调路径。

样例输入：

```
4 5 1 4
2 1 2 1
3 4 3 1
2 3 1 2
3 1 1 4
2 4 2 4
```

样例输入：

```
2
```

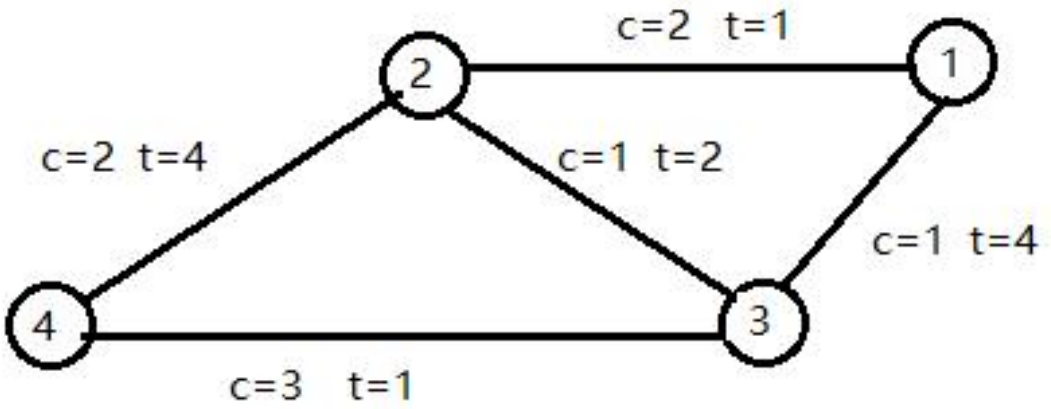
[3622]双调路径

样例输入：

```
4 5 1 4
2 1 2 1
3 4 3 1
2 3 1 2
3 1 1 4
2 4 2 4
```

样例输入：

```
2
```



1——>4的路线：

路线一	1->2->4:	费用：4	时间：5
路线二	1->3->4:	费用：4	时间：5
路线三	1->2->3->4:	费用：6	时间：4
路线四	1->3->2->4:	费用：4	时间：10

路线一和路线二优于路线四，并且路线一和二费用和时间均相同，所以看作一条路线，而路线一和路线三无法比较优劣，所以一共有两条最优路径。

[3622]双调路径

分析：

运用背包的思想，dis[i][j]表示起点到i点费用为j所用的时间，用spfa算法求dis，再遍历dis[终点][0-费用最大值]，费用最小且时间不是无穷大的路径一定为最优路径，之后虽然费用不是最小，但是时间小的也是最优路径。

费用最大值：起点到终点最差的情况是每个城市都需经过，则边数为n-1，费用最大值为（n-1）*100

本题中的图为无向图，因此起点终点正反需要存两次，边<=600,同时存每条边的费用和时间

[3622]双调路径 spfa

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=110,M=610; //城市数<=100,道路数<=300,双向道路，所以边<=600
int h[N],e[M],c[M],t[M],idx,ne[M];
int dis[N][N*N],vis[N][N*N],inf;
//dis[i][j]表示起点到i点费用为j所用的时间,vis记录状态，inf为时间和费用的最大值
struct node{
    int v,ci;  // v：点 ci：费用
};
void add(int a,int b,int cc,int tt){  //建立无向图
    e[idx]=b,t[idx]=tt,c[idx]=cc,ne[idx]=h[a],h[a]=idx++;
}

int main(){
    int n,m,q1,q2;
    cin>>n>>m>>q1>>q2;  //n：城市数  m：道路数  q1：起点  q2：终点
    memset(h,-1,sizeof(h)); //h数组初始化为-1
    for(int i=1;i<=m;i++)
    {
        int a,b,cc,tt;
        cin>>a>>b>>cc>>tt;  //输入图的信息
        add(a,b,cc,tt);  //双向
        add(b,a,cc,tt);
    }
    inf=(n-1)*110; //加入起点到终点每个城市都要经过，即为最大的费用和时间
    spfa(q1);
    int ans=0,minn=inf; //ans：最优路径条数  minn：时间最小值
    for(int i=0;i<=inf;i++)  //费用从小到大取
    {
        if(dis[q2][i]<minn) //最开始费用最小，即为最优路径，此后时间最小，也为最优路径
        {
            minn=dis[q2][i];
            ans++;
        }
    }
    cout<<ans;  //输出
    return 0;
}
```

[3622]双调路径 spfa

```
void spfa(int s){
    queue<node> q;  //队列初始化
    memset(dis,0x3f,sizeof(dis));  //时间初始化为无穷大
    dis[s][0]=0; //起点到起点费用为0，时间为0
    vis[s][0]=1;  //记录状态
    q.push({s,0});  //将状态压入队列
    while(!q.empty())
    {
        node g=q.front();  //出队
        q.pop();
        vis[g.v][g.ci]=0;  //记录状态
        for(int i=h[g.v];i!=-1;i=ne[i])
        {
            int r=e[i];  //与当前点相连的点
            if(g.ci+c[i]<=inf&&dis[g.v][g.ci]+t[i]<dis[r][g.ci+c[i]])
            {
            //若当前点的费用加上到r点的费用小于费用最大值
            //并且当前点到r点的时间小于原有的dis[r][当前点的费用加上到r点的费用]
                dis[r][g.ci+c[i]]=dis[g.v][g.ci]+t[i];//更新
                if(!vis[r][g.ci+c[i]]) //若没有记录过
                {
                    vis[r][g.ci+c[i]]=1;  //记录
                    q.push({r,g.ci+c[i]});  //将状态压入队列
                }
            }
        }
    }
}
```

[3622]双调路径 dijistra

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=110,M=610;
int h[N],e[M],c[M],t[M],idx,ne[M];
int dis[N][N*N],vis[N][N*N],inf;
struct node{
    int v,ci,tim;
};
priority_queue<node> que;
bool operator<(const node &a,const node &b)
{
    return a.tim>b.tim;
}
void add(int a,int b,int cc,int tt){
    e[idx]=b,t[idx]=tt,c[idx]=cc,ne[idx]=h[a],h[a]=idx++;
}
```

```cpp
int main(){
    int n,m,q1,q2;
    cin>>n>>m>>q1>>q2;
    memset(h,-1,sizeof(h));
    for(int i=1;i<=m;i++)
    {
        int a,b,cc,tt;
        cin>>a>>b>>cc>>tt;
        add(a,b,cc,tt);
        add(b,a,cc,tt);
    }
    inf=(n-1)*110;
    dijistra(q1);
    int ans=0,minn=inf;
    for(int i=0;i<=inf;i++)
    {
        if(dis[q2][i]<minn)
        {
            minn=dis[q2][i];
            ans++;
        }
    }
    cout<<ans;
    return 0;
}
```

[3622]双调路径 dijistra

```
17  void dijistra(int s){
18      memset(dis,0x3f,sizeof(dis));
19      dis[s][0]=0;
20      que.push({s,0,0});
21      while(!que.empty())
22      {
23          node tmp=que.top();
24          que.pop();
25          if(tmp.tim!=dis[tmp.v][tmp.ci])
26          {
27              continue;
28          }
29          for(int i=h[tmp.v];i!=-1;i=ne[i])
30          {
31              int r=e[i];
32              if(c[i]+tmp.ci<inf&&tmp.tim+t[i]<dis[r][c[i]+tmp.ci])
33              {
34
35                  dis[r][c[i]+tmp.ci]=tmp.tim+t[i];
36                  que.push({r,c[i]+tmp.ci,dis[r][c[i]+tmp.ci]});
37              }
38          }
39      }
40  }
```

[3622]双调路径

- 优化：

如果dis[i][j]与dis[i][k]有：k<j和dis[i][j]>dis[i][k]这样的关系，则dis[i][j]就是无用的状态

所以可以用树状数组来维护最小值，只有dis[i][j]小于最小值，才更新。

[3622]双调路径

树状数组：

dis为初始数组
de为树状数组

例如 1-8

de[x][1]=dis[x][1]

de[x][2]=de[x][1]、dis[x][2]

de[x][3]=dis[x][3]

de[x][4]=de[x][2]、de[x][3]、dis[x][4]

de[x][5]=dis[x][5]

de[x][6]=de[x][5]、dis[x][6]

de[x][7]=dis[x][7]

de[x][8]=de[x][4]、de[x][6]、de[x][7]、dis[x][8]

de下标转化为二进制，如8=1000，末尾有3个0，则管辖2的3次方，即8个数；如5=101，末尾没有0，则只管辖2的0次方，即1个数

[3622]双调路径

树状数组：

如果已知y，要求遍历1-y的所有dis[x][1-y]

将y转化为二进制，如7=111（则可知de[x][7]），再将末尾的1变为0，即为110=6（则可知de[x][6]），重复操作，100=4（则可知de[x][4]）,直到为0结束。 所以 1-y的所有dis[x][1-y]=de[x][7]、de[x][6]、de[x][4];

如果已知y,w，要求将所有管辖dis[x][y]的de[x][1-max]的值更新为w

与上述操作相反，将y转化为二进制后，重复将末尾0变为1或直接在末尾加上1

[3622]双调路径

树状数组：

lowbit（int x）函数可以获取x的二进制末尾1的位置

x&（-x）

-x是指将x的二进制取反再加1

例如13=1011，-13=0100+1=0101，则13&（-13）=1011&0101=0001=1

[3622]双调路径 spfa+树状数组

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=110,M=610; //城市数<=100,道路数<=300,双向道路，所以边<=600
int h[N],e[M],c[M],t[M],idx,ne[M];
int dis[N][N*N],vis[N][N*N],inf,de[N][N*N];
//dis[i][j]表示起点到i点费用为j所用的时间,vis记录状态，inf为时间和费用的最大值
struct node{
    int v,ci;  // v：点 ci：费用
};
void add(int a,int b,int cc,int tt){  //建立无向图
    e[idx]=b,t[idx]=tt,c[idx]=cc,ne[idx]=h[a],h[a]=idx++;
}
int lowbit(int x){  //取得x的二进制的末尾1的位置
    return x&(-x);
}
int get(int x,int y){  //获取1-y的最小值
    y++;  //树状数组下标不能为0
    int minn=inf;
    while(y>=1)
    {
        minn=min(de[x][y],minn);
        y-=lowbit(y);
    }
    return minn;
}
void update(int x,int y,int w){  //更新
    y++;
    while(y<inf)  //所有管辖y的状态
    {
        de[x][y]=min(de[x][y],w);
        y+=lowbit(y);
    }
}
```

[3622]双调路径 spfa+树状数组

```
void spfa(int s){
    queue<node> q;  //队列初始化
    memset(dis,0x3f,sizeof(dis));  //时间初始化为无穷大
    memset(de,0x3f,sizeof(de));
    dis[s][0]=0; //起点到起点费用为0，时间为0
    vis[s][0]=1;  //记录状态
    update(s,0,0);  //树状数组初始化
    q.push({s,0});  //将状态压入队列
    while(!q.empty())
    {
        node g=q.front();  //出队
        q.pop();
        vis[g.v][g.ci]=0;  //记录状态
        for(int i=h[g.v];i!=-1;i=ne[i])
        {
            int r=e[i];  //与当前点相连的点
            if(g.ci+c[i]<=inf&&dis[g.v][g.ci]+t[i]<get(r,g.ci+c[i]))
            {
            //若当前点的费用加上到r点的费用小于费用最大值
            //（0-当前点的费用加上到r点的费用）所有状态的时间都小于此时的状态
                dis[r][g.ci+c[i]]=dis[g.v][g.ci]+t[i];//更新
                update(r,g.ci+c[i],dis[g.v][g.ci]+t[i]); //树状数组更新
                if(!vis[r][g.ci+c[i]]) //若没有记录过
                {
                    vis[r][g.ci+c[i]]=1;  //记录
                    q.push({r,g.ci+c[i]});  //将状态压入队列
                }
            }
        }
    }
}
```

```
int lowbit(int x){  //取得x的二进制的末尾1的位置
    return x&(-x);
}
int get(int x,int y){  //获取1-y的最小值
    y++;  //树状数组下标不能为0
    int minn=inf;
    while(y>=1)
    {
        minn=min(de[x][y],minn);
        y-=lowbit(y);
    }
    return minn;
}
void update(int x,int y,int w){  //更新
    y++;
    while(y<inf)  //所有管辖y的状态
    {
        de[x][y]=min(de[x][y],w);
        y+=lowbit(y);
    }
}
```

[3622]双调路径 spfa+树状数组

```
int main(){
    int n,m,q1,q2;
    cin>>n>>m>>q1>>q2;  //n：城市数  m：道路数  q1：起点  q2：终点
    memset(h,-1,sizeof(h)); //h数组初始化为-1
    for(int i=1;i<=m;i++)
    {
        int a,b,cc,tt;
        cin>>a>>b>>cc>>tt;  //输入图的信息
        add(a,b,cc,tt);  //双向
        add(b,a,cc,tt);
    }
    inf=(n-1)*110; //加入起点到终点每个城市都要经过，即为最大的费用和时间
    spfa(q1);
    int ans=0,minn=inf; //ans：最优路径条数  minn：时间最小值
    for(int i=0;i<=inf;i++)  //费用从小到大取
    {
        if(dis[q2][i]<minn) //最开始费用最小，即为最优路径，此后时间最小，也为最优路径
        {
            minn=dis[q2][i];
            ans++;
        }
    }
    cout<<ans;  //输出
    return 0;
}
```

[3622]双调路径 dijistra+树状数组

```
void dijistra(int s){
    memset(dis,0x3f,sizeof(dis));
    memset(de,0x3f,sizeof(de));
    dis[s][0]=0;
    update(s,0,0);
    que.push({s,0,0});
    while(!que.empty())
    {
        node tmp=que.top();
        que.pop();
        if(tmp.tim!=dis[tmp.v][tmp.ci])
        {
            continue;
        }
        for(int i=h[tmp.v];i!=-1;i=ne[i])
        {
            int r=e[i];
            if(c[i]+tmp.ci<inf&&tmp.tim+t[i]<get(r,tmp.ci+c[i]))
            {

                dis[r][c[i]+tmp.ci]=tmp.tim+t[i];
                update(r,c[i]+tmp.ci,tmp.tim+t[i]);
                que.push({r,c[i]+tmp.ci,dis[r][c[i]+tmp.ci]});
            }
        }
    }
}
```

```
int lowbit(int x){
    return x&(-x);
}
int get(int x,int y){
    y++;
    int minn=inf;
    while(y>=1)
    {
        minn=min(minn,de[x][y]);
        y-=lowbit(y);
    }
    return minn;
}
void update(int x,int y,int w){
    y++;
    while(y<=inf)
    {
        de[x][y]=min(de[x][y],w);
        y+=lowbit(y);
    }
}
```