高级数据结构
Bellman-ford 算法与SPFA

信智学院　陈琰宏

主要内容

01 Bellman-ford 算法

02 优先队列

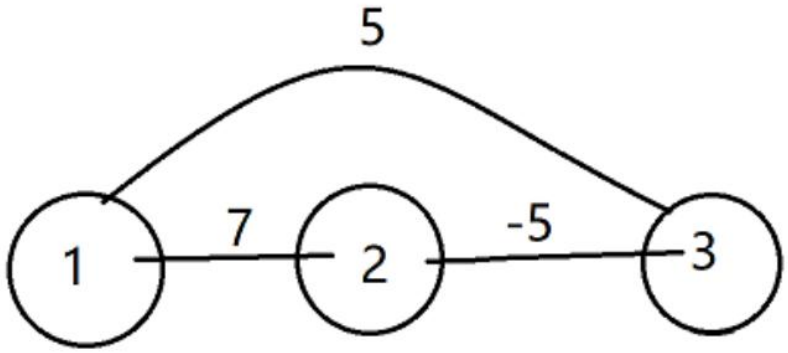
03 案例实现

1 最短路径

- 最短路径问题：如果从图中某一顶点(称为源点)到达另一顶点(称为终点)的路径可能不止一条，如何找到一条路径，使得沿此路径各边上的权值总和达到最小。
- 问题解法：
  1. 权值为非负的单源最短路径问题(固定源点)—Dijkstra算法(迪杰斯特拉算法)；
  2. 权值为任意值的单源最短路径问题(固定源点) —Bellman-Ford算法(贝尔曼—福特算法)；
  3. 所有顶点之间的最短路径问题—Floyd-Warshall算法(弗洛伊德算法)；

Bellman-ford 算法

Dijkstra算法虽然好，但是它不能解决带有负权边（边的权值为负数）的图。



如图，如果用dijkstra算法的话就会出错，因为如果从1开始，第一步dist[2] = 7, dist[3] = 5;在其中找出最小的边是dist[3] = 5;然后更新dist[2] = 0，最终得到dist[2] = 0,dist[3] = 5，而实际上dist[3] = 2；所以如果图中含有负权值，dijkstra失效。

Bellman-ford 算法

Bellman－Ford algorithm（贝尔曼-福特算法），是求解单源最短路径问题的一种算法，可以完美地解决带有负权边的图，它的原理是对图进行次松弛操作(松弛就是更新两点间的最短路径。)，得到所有可能的最短路径。

2.Bellman-Ford算法思想

适用前提：没有负环（或称为负权值回路），因为有负环的话距离为负无穷。

构造一个最短路径长度数组序列$dist^1[u]$ $dist^2[u]$...$dist^{n-1}[u]$，其中：

$dist^1[u]$为从源点v0出发到终点u的只经过一条边的最短路径长度，并有$dist^1[u]$ = Edge[v0][u]

$dist^2[u]$为从源点v0出发最多经过不构成负权值回路的两条边到终点u的最短路径长度

$dist^3[u]$为从源点v0出发最多经过不构成负权值回路的三条边到终点u的最短路径长度

.....

$dist^{n-1}[u]$为从源点v0出发最多经过不构成负权值回路的n-1条边到终点u的最短路径长度

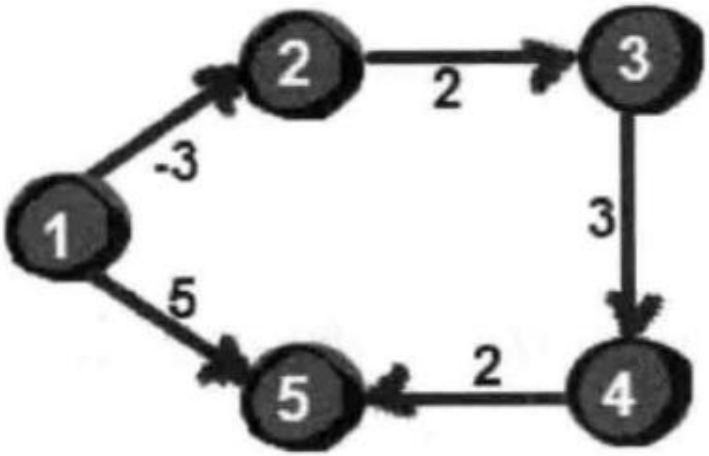
2.Bellman-Ford算法思想

算法最终目的是计算出$dist^{n-1}[u]$，即为源点到顶点u的最短路径长度

初始：$dist^{1}[u]$ = Edge[v0][u]

递推：$dist^{k}[u] = min(dist^{k-1}[u], min\{dist^{k-1}[j] + Edge[j][u]\})$

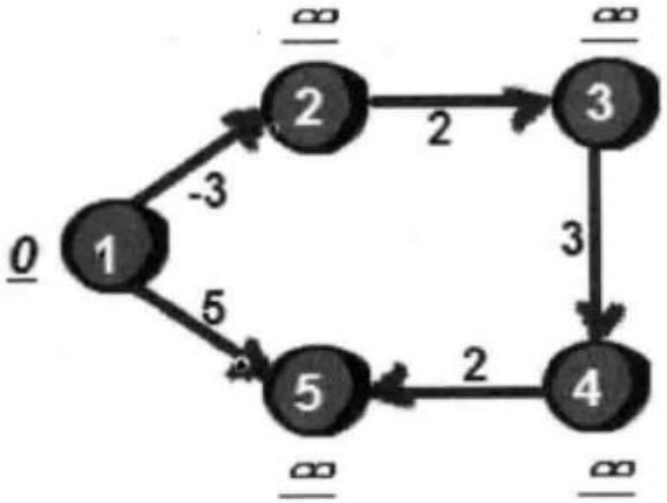
案例详解



边给出的顺序是

2 3 2
1 2 -3
1 5 5
4 5 2
3 4 3

初始化



边给出的顺序

2	3	2
1	2	-3
1	5	5
4	5	2
3	4	3

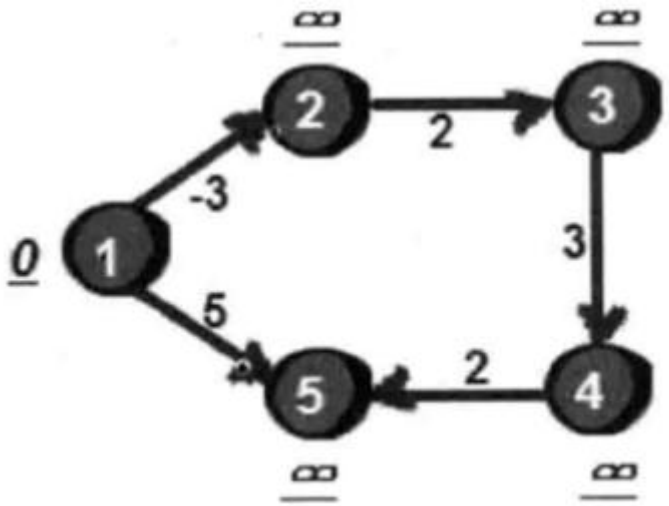
dis = {0, 0, 99999, 99999, 99999, 99999, 0 <repeats 44 times>}

u = {0, 2, 1, 1, 4, 3, 0 <repeats 9994 times>}

v = {0, 3, 2, 5, 5, 4, 0 <repeats 9994 times>}

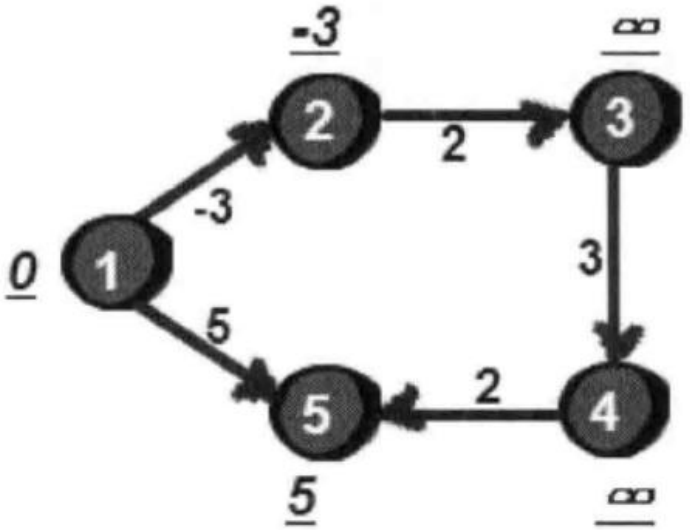
w = {0, 2, -3, 5, 2, 3, 0 <repeats 9994 times>}

对所有边进行第一轮松弛



边给出的顺序

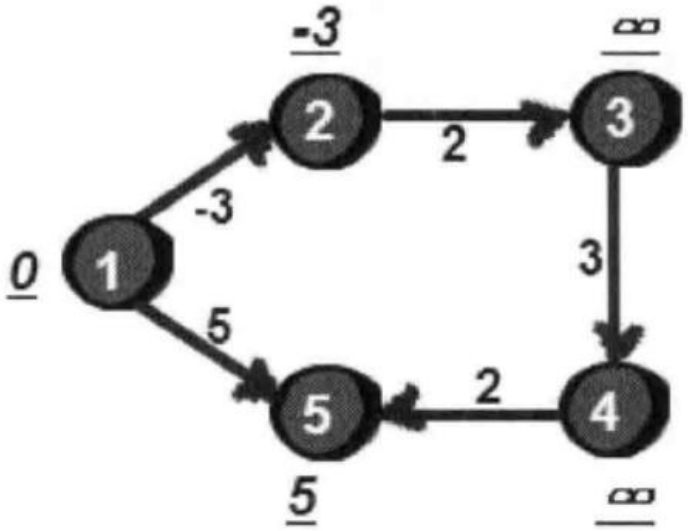
2	3	2
1	2	-3
1	5	5
4	5	2
3	4	3



第1轮：u[2]-v[3] 没有边更新
第1轮：u[1]-v[2] 边更新 dis[2]=-3
第1轮：u[1]-v[5] 边更新 dis[5]=5
第1轮：u[4]-v[5] 没有边更新
第1轮：u[3]-v[4] 没有边更新

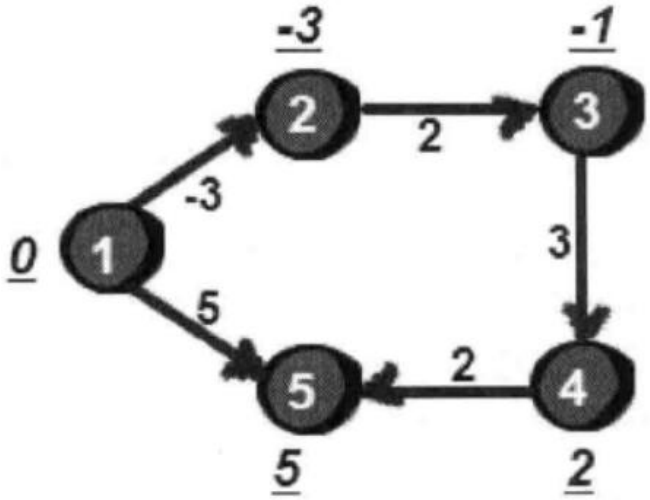
	1	2	3	4	5
dis	0	-3	∞	∞	5

对所有边进行第二轮松弛

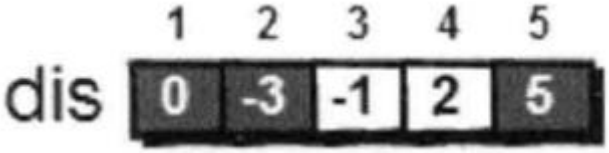


边给出的顺序

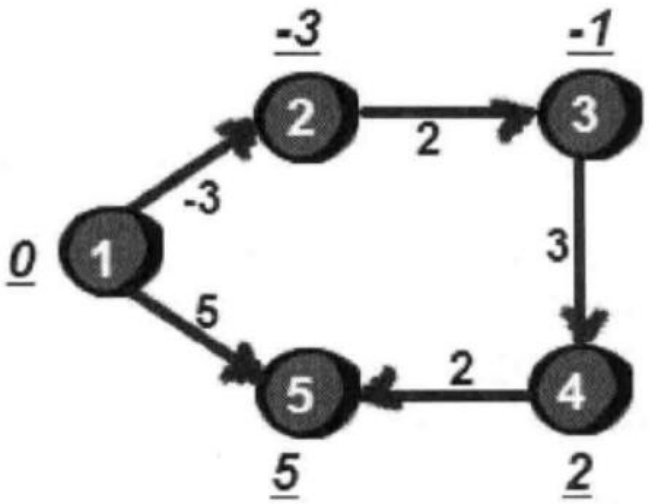
2 3 2
1 2 -3
1 5 5
4 5 2
3 4 3



第2轮：u[2]-v[3] 边更新 dis[3]=-1
第2轮：u[1]-v[2] 没有边更新
第2轮：u[1]-v[5] 没有边更新
第2轮：u[4]-v[5] 没有边更新
第2轮：u[3]-v[4] 边更新 dis[4]=2

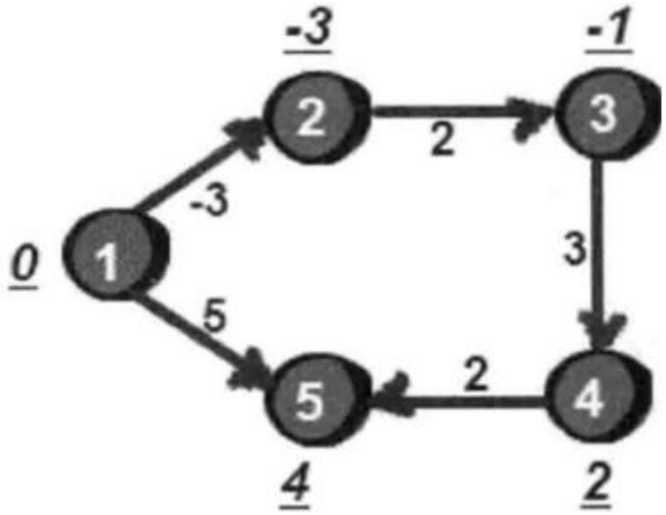


对所有边进行第三轮松弛

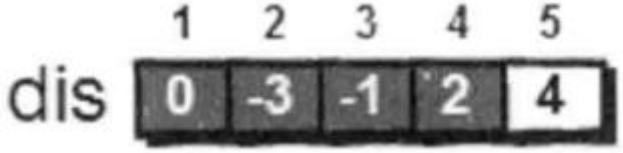


边给出的顺序

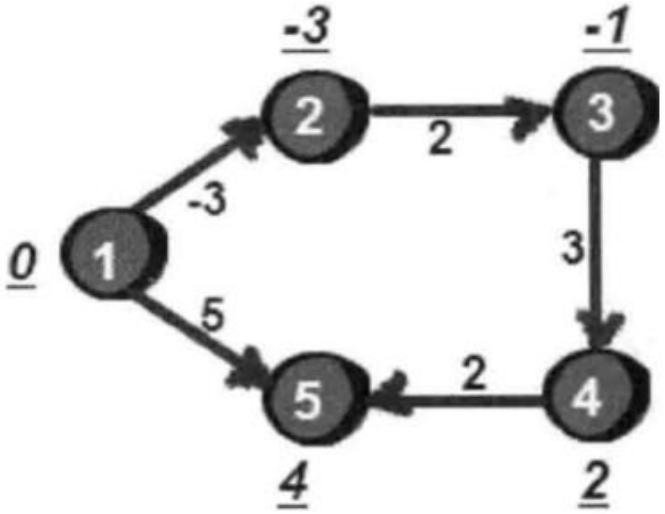
2 3 2
1 2 -3
1 5 5
4 5 2
3 4 3



第3轮：u[2]-v[3] 没有边更新
第3轮：u[1]-v[2] 没有边更新
第3轮：u[1]-v[5] 没有边更新
第3轮：u[4]-v[5] 边更新 dis[5]=4
第3轮：u[3]-v[4] 没有边更新



对所有边进行第四轮松弛



边给出的顺序

2 3 2
1 2 -3
1 5 5
4 5 2
3 4 3

第4轮：u[2]-v[3] 没有边更新
第4轮：u[1]-v[2] 没有边更新
第4轮：u[1]-v[5] 没有边更新
第4轮：u[4]-v[5] 没有边更新
第4轮：u[3]-v[4] 没有边更新

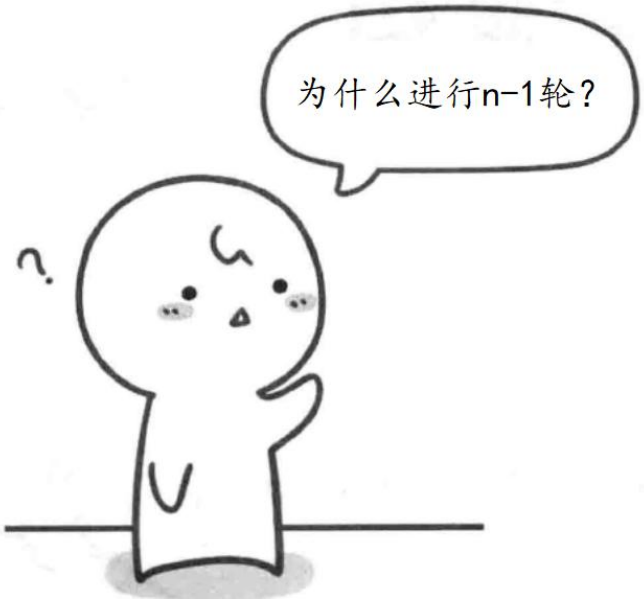
	1	2	3	4	5
dis	0	-3	-1	2	4

核心代码

```
23      //Bellman算法核心语句 4句
24      for (int k = 1; k < n; k++) {//n-1轮
25          for (int i = 1; i <= m; i++) {
26              //每轮对各边进行更新
27              if (dis[v[i]] > dis[u[i]] + w[i]) {
28                  dis[v[i]] = dis[u[i]] + w[i];
29              }
30          }
31      }
```

理解：第1轮在对所有的边进行松地之后，得到的是从1号顶点“只能经过一条边” 到达其余各顶点的最短路径长度。第2轮在对所有的边进行松弛之后，得到的是从1号顶点 “最多经过两条边”到达其余各顶点的最短路径长度。如果进行k轮的话，得到的就是1号 顶点“最多经过k条边”到达其余各顶点的最短路径长度。**进过k-1轮可以到达所有点**。

思考

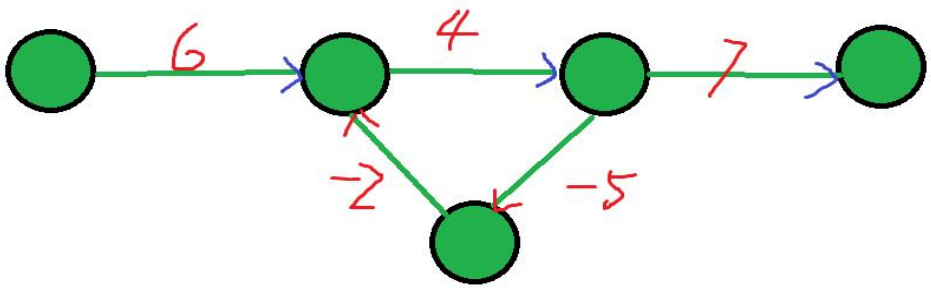


理解：第1轮在对所有的边进行松地之后，得到的是从1号顶点“只能经过一条边” 到达其余各顶点的最短路径长度。第2轮在对所有的边进行松弛之后，得到的是从1号顶点最多经过两条边到达其余各顶点的最短路径长度。如果进行k轮的话，得到的就是1号顶点最多经过k条边 到达其余各顶点的最短路径长度。**进过k-1轮可以到达所有点。**

检测一个图是否含有负权回路

如果一个图如果没有负权回路，那么最短路径所　包含的边最多为n-1条，即进行n-1轮松弛之后最短路不会再发生变化。如果在n-1轮松弛　之后最短路仍然会发生变化，则该图必然存在负权回路,用flag表示是否存在负环。

```
for(int i = 1; i <= m; i++){
    if (dis[v[i]] > dis[u[i]] + w[i]) {
        flag = 1;
    }
}
```



代码

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 50, MAXM = 10000, MAX = 99999;
int u[MAXM], v[MAXM], w[MAXM], dis[MAXN];
int n, m;
void init() {//输出数据；初始化
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        dis[i] = MAX;
    }
    dis[1] = 0;
    for (int i = 1; i <= m; i++) {
        cin >> u[i] >> v[i] >> w[i];
    }
}

int main() {
    init();
    //Bellman算法核心语句 4句
    for (int k = 1; k < n; k++) {//n-1轮
    int flag = 0;
    for(int i = 1; i <= m; i++){
    if(flag){//判断是否存在负环
}
```

代码

```
17  int main() {
18      init();
19      //Bellman算法核心语句 4句
20      for (int k = 1; k < n; k++) {//n-1轮
21          for (int i = 1; i <= m; i++) {
22              //每轮对各边进行更新
23              if (dis[v[i]] > dis[u[i]] + w[i]) {
24                  dis[v[i]] = dis[u[i]] + w[i];
25              }
26          }
27      }
28      int flag = 0;
29      for(int i = 1; i <= m; i++){
30          if (dis[v[i]] > dis[u[i]] + w[i]) {
31              flag = 1;
32          }
33      }
34      if(flag){//判断是否存在负环
35          cout << "此图含有负权回路!" << endl;
36      } else{
37          cout << "此图不含负权回路!" << endl;
38          for(int i = 1; i <= n; i++){
39              cout << dis[i] << " ";
40          }
41          cout << endl;
42      }
43  }
```

简单优化

Bellman-Ford算法的时间复杂度是O(NM)，这个时间复杂度似比Dijkstra算法还要高，我们还可以对其进行优化。在实际操作中， Bellman-Ford算法经常会在未达到n-1轮松弛前就已经计算出最短路，之前我们已经说过，n-1其实是最大值。因此可以添加一个标记标量标记是否有更新，如果在新一轮的松弛中数 组dis没有发生变化，则可以提前跳出循环，代码如下。

简单优化

```
18  int main() {
19      init();
20      //Bellman算法核心语句 4句
21      int check;
22      for (int k = 1; k < n; k++) {//n-1轮
23          check=0;//用来标记 本轮松弛中数组dis是否发生更新
24          for (int i = 1; i <= m; i++) {
25              //每轮对各边进行更新
26              if (dis[v[i]] > dis[u[i]] + w[i]) {
27                  dis[v[i]] = dis[u[i]] + w[i];
28                  check=1;//发生更新
29              }
30          }
31          if(check==0)break;//如果本轮没有更新，提前退出循环
32      }
33      int flag = 0;
34      for(int i = 1; i <= m; i++){
35          if (dis[v[i]] > dis[u[i]] + w[i]) {
36              flag = 1;
37          }
38      }
39      if(flag){//判断是否存在负环
48  }
```

队列优化

在每实施一次松弛操作后，就会有一些顶点已经求得其最短路，此后这些顶点的最短路的估计值就会一直保持不变，不再受后续松弛操作的影响，但是每次还要判断是否需要松弛，这里浪费了时间。这就启发我们： 每次仅对最短路估计值发生变化了的顶点的所有出边执行松弛操作。

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：

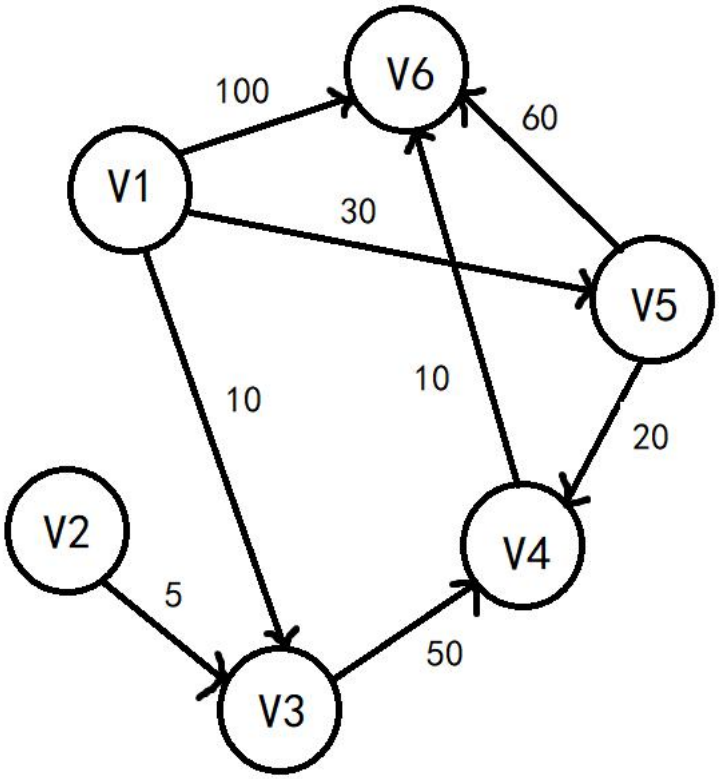
1：建立一个队列，初始时队列里只有起始点，

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

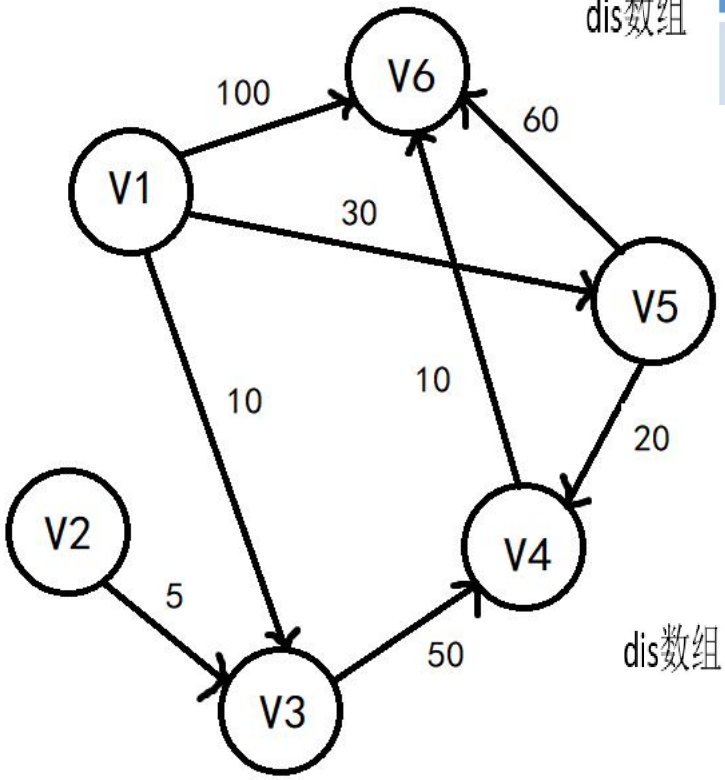


首先我们先初始化数组dis如下图所示：（除了起点赋值为0外，其他顶点的对应的dis的值都赋予无穷大，这样有利于后续的松弛）

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

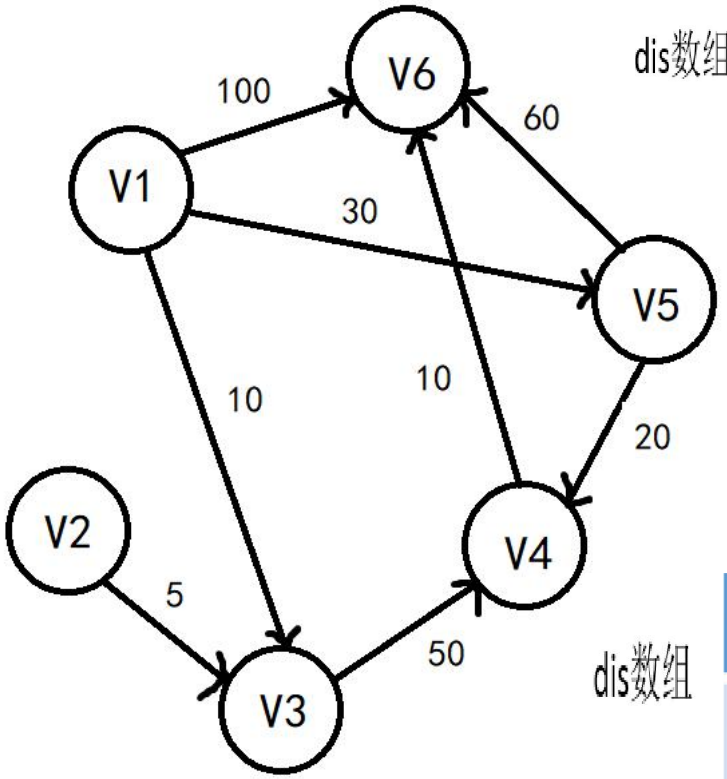
v1	v2	v3	v4	v5	v6
0	∞	∞	∞	∞	∞

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

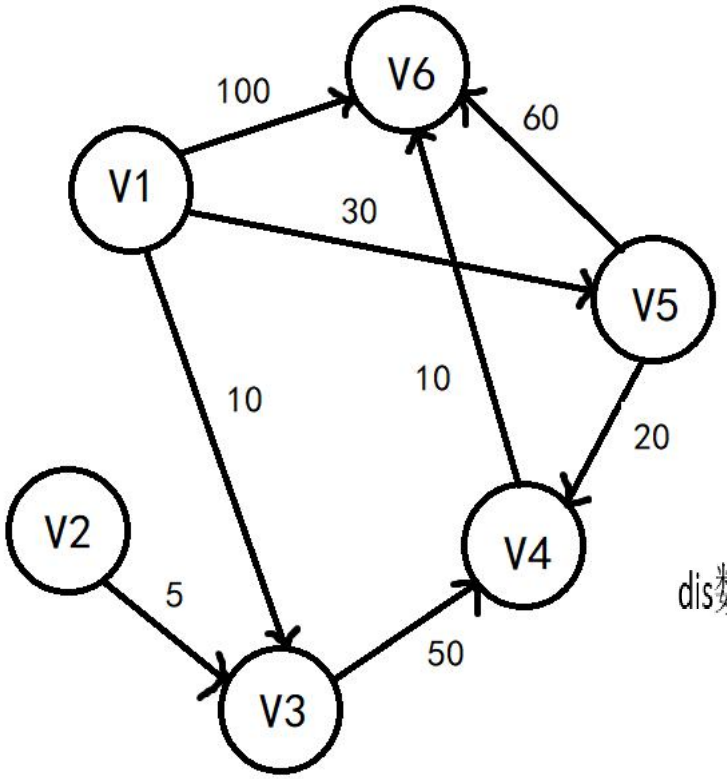
此时，队首元素为v3，v3出队列，然后，对以v3为弧尾的边对应的弧头顶点进行松弛操作，可以发现v1到v4的边，经过v3松弛变短了，所以更新dis数组，得到如下结果：

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

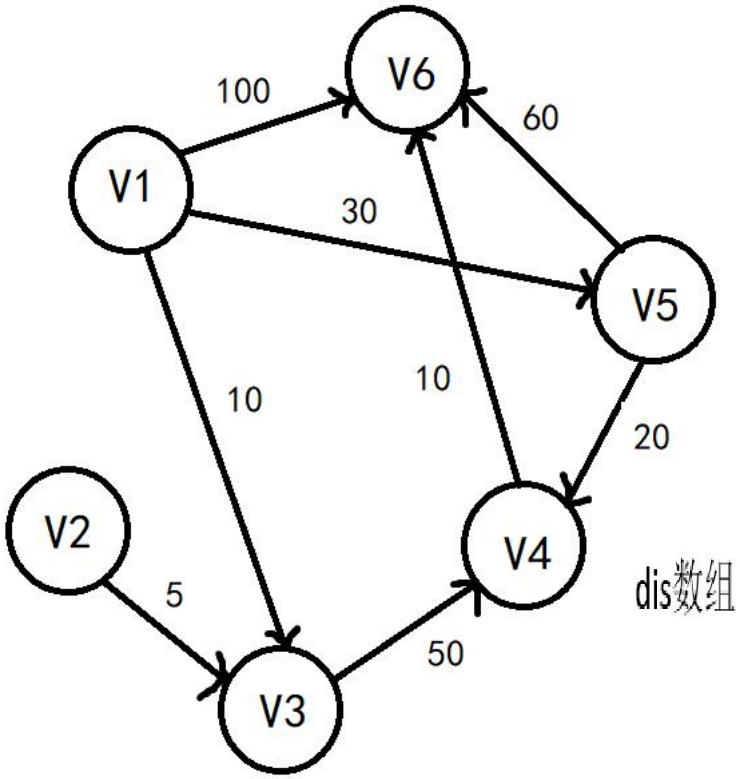
第三次循环

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}



dis数组

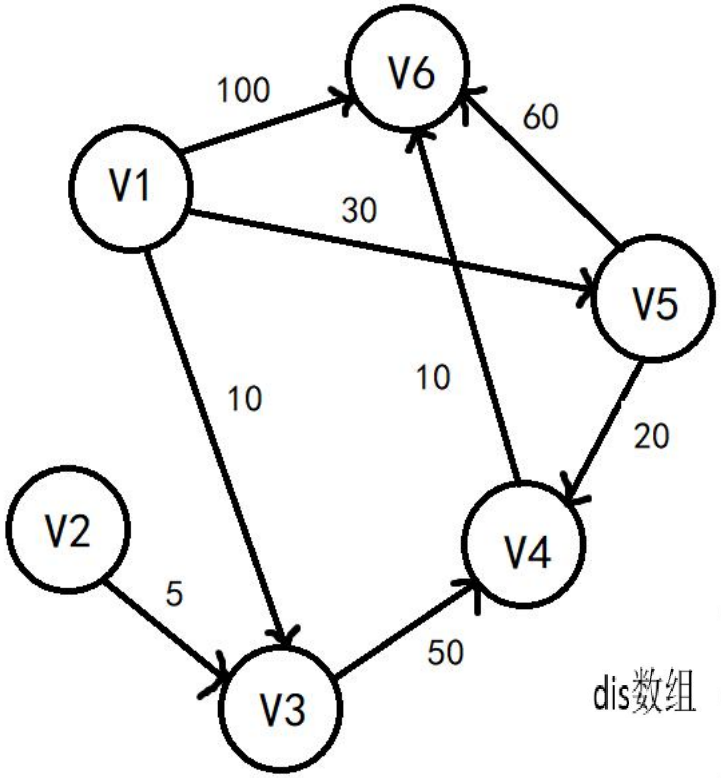
第四次循环

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}

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

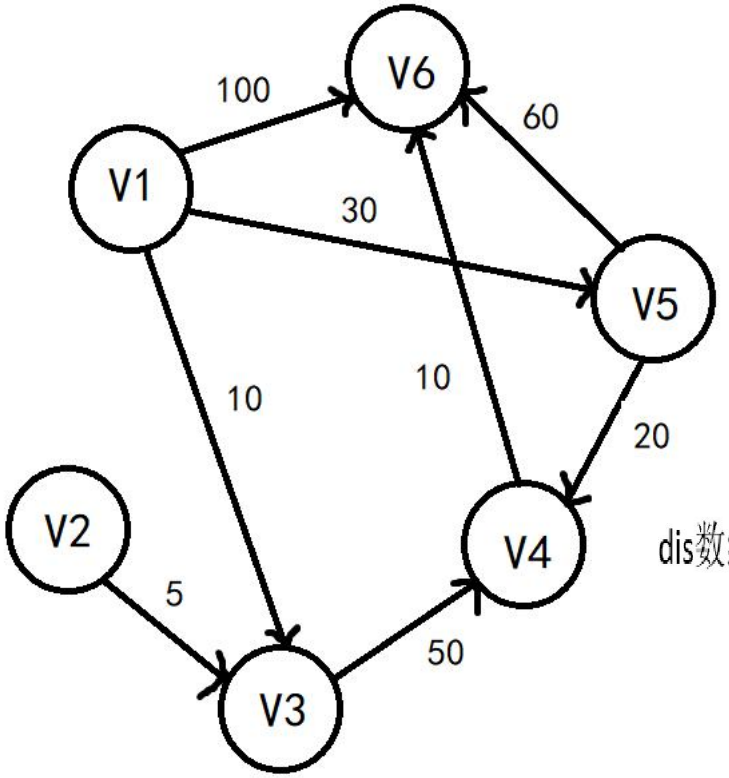
此时，队首元素为v4，v4出队列，然后，对以v4为弧尾的边对应的弧头顶点进行松弛操作，可以发现v1到v6的最短路径，经过v4松弛变短了，所以更新dis数组，得到如下结果：

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

[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 √

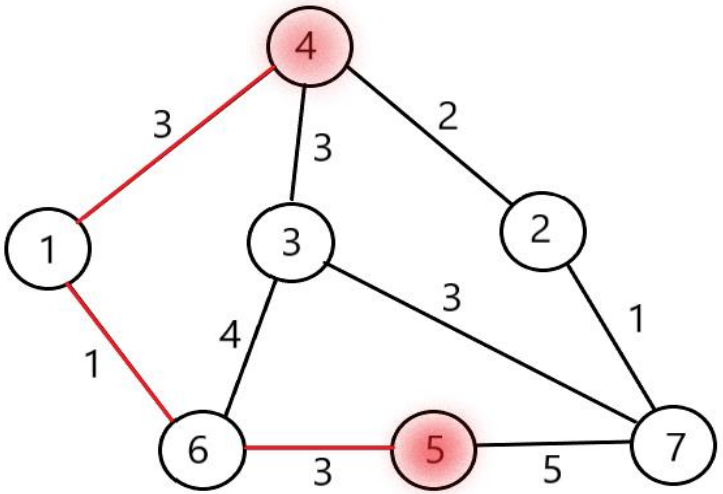
样例输出：

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  }
```

[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。

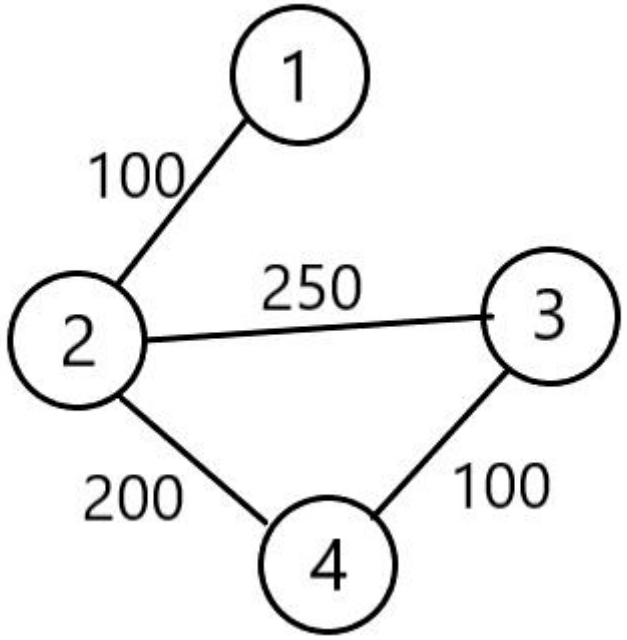
[3566] Roadblocks

样例输入：

```
4 4
1 2 100
2 4 200
2 3 250
3 4 100
```

R条路，N个路口

u、v、w



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)

[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。

[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

```
39 int main(){
40     int i,j,r,u,v,w;
41     cin>>n>>r;//r条路，n个路口
42     for(i=1;i<=r;i++){//输入数据，双向加边
43         scanf("%d%d%d",&u,&v,&w);
44         nod[u].push_back({v,w});
45         nod[v].push_back({u,w});
46     }
47     spfa(1,dis1);//求1到每一个节点的最短路
48     spfa(n,disN);//求n到每一个节点的最短路
49     int ans=inf;//ans：次短路长度
50     //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）
51     for(i=1;i<=n;i++){//枚举每一个点
52         for(j=0;j<nod[i].size();j++){//枚举该点连接的边
53             u=i;            //边的左端点
54             v=nod[i][j].v;//右端点
55             w=nod[i][j].w;//边权
56             int l=dis1[u]+w+disN[v];
57             if(ans>l&&dis1[n]!=l)ans=l;//更新次短路（dis1[n]是1到n的最短路）
58         }
59     }
60     cout<<ans;
61     return 0;
62 }
```

今天的课程结束啦……

下课了…
同学们再见！