

信奥算法



最短路问题

复习Dijkstra算法

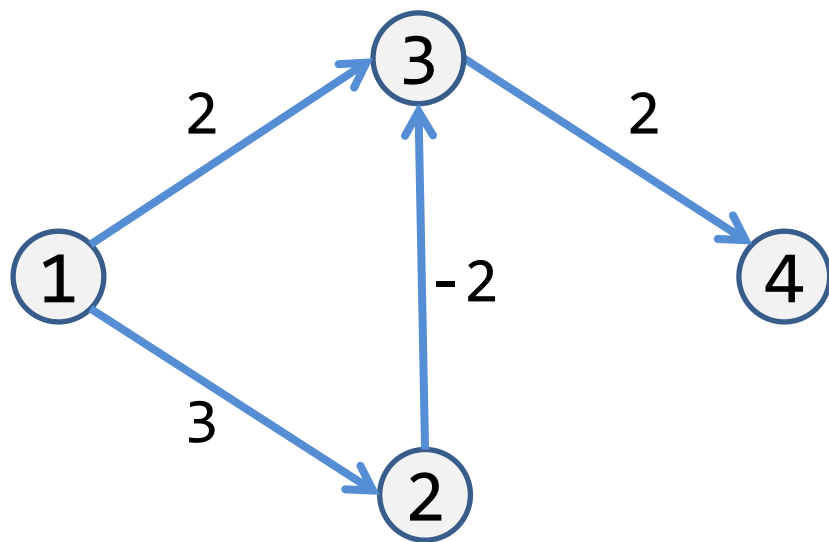
最短路问题

负权图

Dijkstra与负边

假设出现负权边，
Dijkstra算法求出的最短路还正确吗

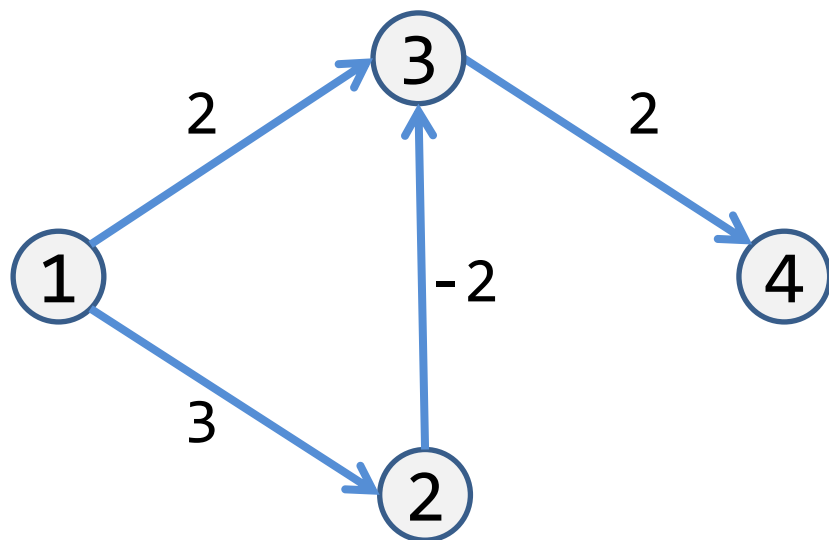
请画出一个简单的反例
包含负权边
使Dijkstra算法出错



正确答案

u	d[u]
1	0
2	3
3	1
4	3

跑一遍Dijkstra
得到的d[]会是哪4个数?



错误答案

u	d[u]	ok[u]
1	0	1
2	3	1
3	1	1
4	4	1

Dijkstra错误地确认 $d[3]=2$ 为最终值
 $d[3]$ 获得了修改后续邻居的资格

$d[3]$ 帮助修改 $d[4]$ 一次后
 $d[4]$ 再也没有机会改正了
 虽然 $d[3]$ 可以被 $d[2]$ 改正

Dijkstra算法里
 每条边最多只会
 使用1次

如何修改算法
 使结果正确?

处理负边的算法

初始化

$d[\text{起点}] = 0$, 其他 $d[]$ 值为 INF

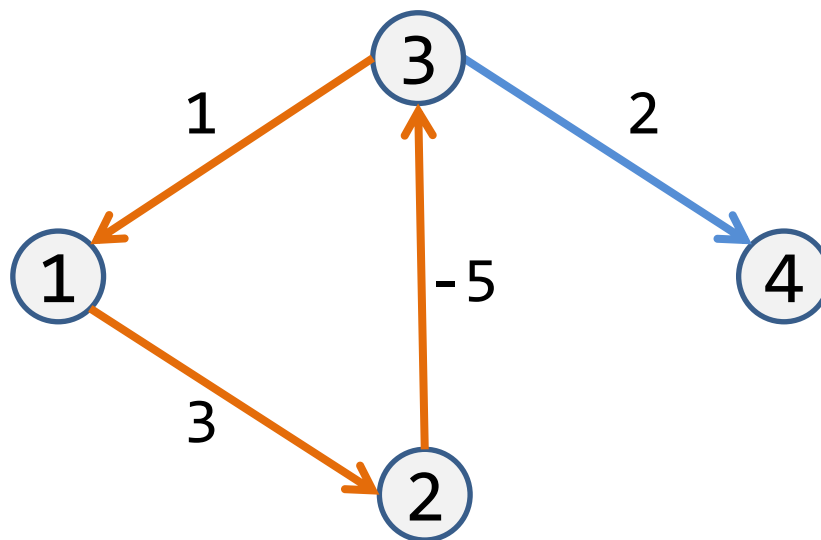
迭代
更新

对于每条边: u 指向 v , 边长 w

尝试用 $d[u] + w$ 去更新 $d[v]$

不断更新直到所有 $d[]$ 值不变

负权回路



若存在一个环路，权值总和为负数

沿负环不断行走，长度会被不断缩短

最短路数值无法确定

航班最短时间

n 个城市编号1到 n 。城市间共 m 条直飞单向航线，每条航线有起点，终点，和飞行时间。其中有些特殊航线的飞行时间为负数，代表时光倒流。求1号城市到各城市最少时间？

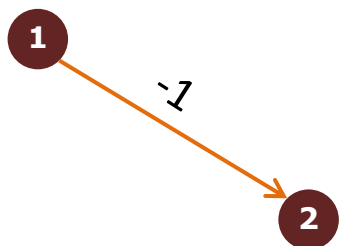
保证都能飞到，保证没有负环。 $n \leq 100, m \leq 200$

输入样例

```
2 1
1 2 -1
```

输出样例

```
0 -1
```

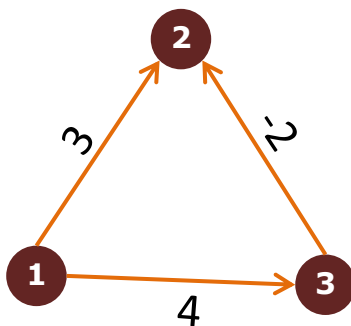


输入样例

```
3 3
1 2 3
3 2 -2
1 3 4
```

输出样例

```
0 2 4
```

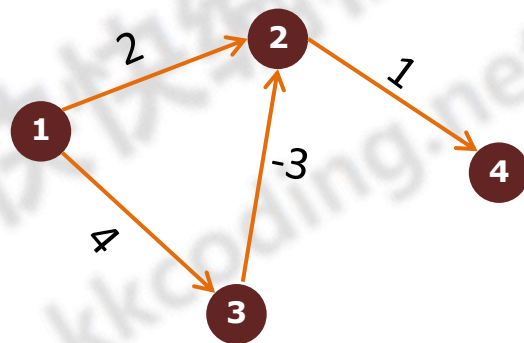


输入样例

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

输出样例

```
0 1 4 2
```



最短路问题

单源多汇SSSP

single source shortest path

Bellman-Ford算法

Bellman-Ford算法

$d[u]$ 记录从源点到 u 号点当前的最短路长度

初始化 $d[1]$ 为 0 ，其他 $d[u]$ 为 INF

主循环：以下操作重复共 $n-1$ 次

之后会揭秘

思考为什么 $n-1$

对于每条边：从 u 到 v ，权重 w

若 $d[u] + w < d[v]$ ，则更新
 $d[v] = d[u] + w$

复杂度是多少？

$O(nm)$

Bellman-Ford

边集
数组

代码1

```
3 typedef long long ll;
4 const ll N=109;
5 const ll M=209;
6 const ll INF=1e9;
7 struct edge{ll u,v,w};
8 edge E[M];
9 ll n,m,d[N];
10 void BellmanFord()
17 int main(){
18     cin>>n>>m;
19     for(ll i=0;i<m;++i)
20         cin>>E[i].u>>E[i].v>>E[i].w;
21     BellmanFord();
22     for(ll u=1;u<=n;++u)
23         cout<<d[u]<<" ";
24     return 0;
25 }
```

易错点

无向边用双向边储存时
边数要翻倍M=409

u代表边的起点
v代表边的终点
w代表边的权值

Bellman-Ford

边集
数组

代码1

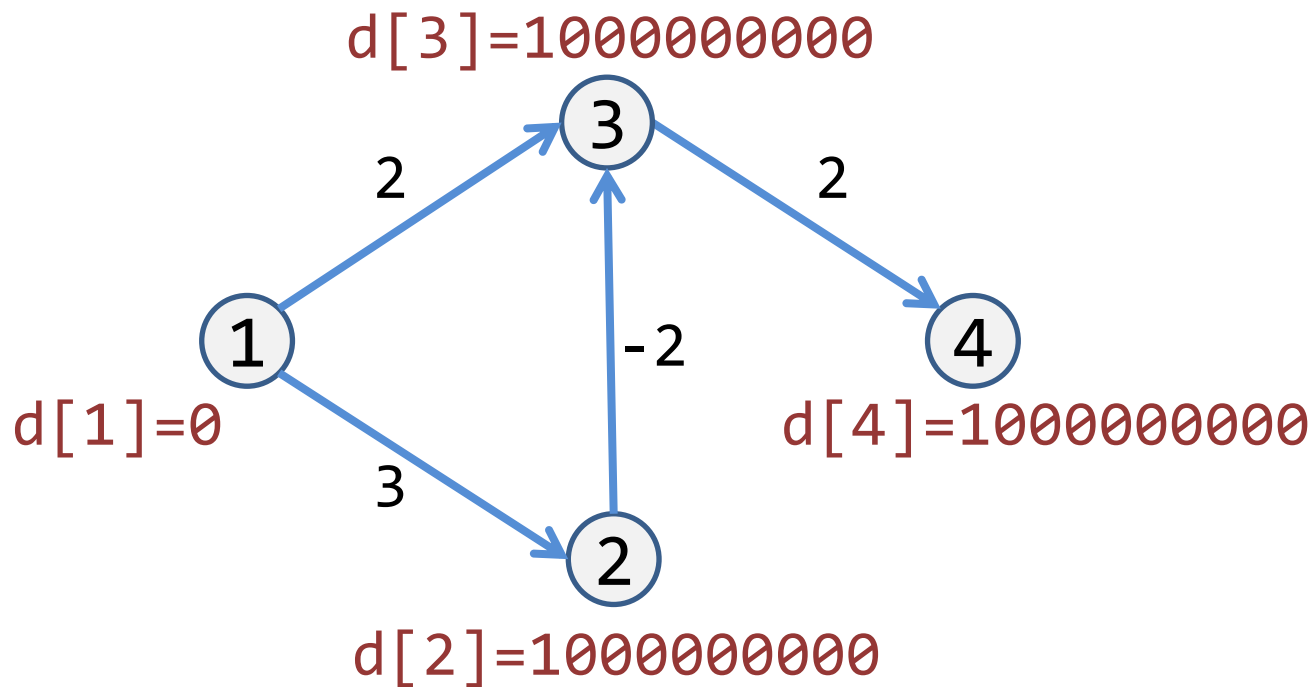
```
10 void BellmanFord(){
11     fill(d, d+n+1, INF); ←
12     d[1]=0; ←
13     for(ll k=1; k<=n-1; ++k)
14         for(ll i=0; i<m; ++i)
15             d[E[i].v] = min(d[E[i].v], d[E[i].u] + E[i].w);
16 }
```

共迭代n-1轮

枚举所有边

i号边E[i]的信息
起点E[i].u
终点E[i].v
边长E[i].w

用从源点到i号边起点E[i].u的
当前最短路d[E[i].u]
加上i号边长E[i].w
尝试更新
从源点到i号边终点E[i].v的
当前最短路d[E[i].v]



共4条边

从3号到4号边长2
从2号到3号边长-2
从1号到3号边长2
从1号到2号边长3

每1轮

用 所 有 边	尝 试 更 新
------------------	------------------

共
几
轮

Bellman-Ford

边集
数组

代码1

```
10 void BellmanFord(){
11     fill(d,d+n+1,INF);
12     
13     for(ll k=1;k<=n-1;++k)
14         for(ll i=0;i<m;++i)
15             d[E[i].v]=
16 }
```

快快编程
kkcoding.net

Bellman-Ford 正确性

$d[u]$ 记录从源点到 u 号点**当前**的最短路长度

初始化 $d[1]$ 为0, 其他 $d[u]$ 为INF

$d[i]$ 的
含义随着
循环变化

重复 $n-1$ 次: 查看所有边, 尝试更新 $d[]$

1次循环后 $d[u]$ 含义: 只借助1条边, 从源点到 u 的最短路长度

2次循环后 $d[u]$ 含义: 只借助2条边, 从源点到 u 的最短路长度

3次循环后 $d[u]$ 含义: 只借助3条边, 从源点到 u 的最短路长度

• • • • •

• • • • •

$n-1$ 次循环后 $d[u]$ 含义: 借助**所有**边, 从源点到 u 的最短路长度

所以迭代 $n-1$ 轮足够了!

Bellman-Ford

邻接表

代码2

```
6 vector<ll> to[N],w[N];
7 ll n,m,d[N];
8 void BellmanFord(){
19 int main(){
20     cin>>n>>m;
21     for(ll i=0;i<m;++i){
22         ll u,v,cost;
23         cin>>u>>v>>cost;
24         to[u].push_back(v);
25         w[u].push_back(cost);
26     }
27     BellmanFord();
28     for(ll u=1;u<=n;++u)
29         cout<<d[u]<<" ";
30     return 0;
31 }
```

快快编程
kkcoding.net

Bellman-Ford

邻接表

代码2

```
8 void BellmanFord(){
9     fill(d,d+n+1,INF);
10    d[1]=0;
11    for(ll k=1;k<=n-1;++k)
12        for(ll u=1;u<=n;++u)
13            for(ll i=0;i<to[u].size();++i){
14                ll v=to[u][i];
15                ll cost=w[u][i];
16                d[v]=min(d[v],d[u]+cost);
17            }
18 }
```

v是u的i号邻居
w是u出发的i号边长度

请用电脑完成此函数

2分钟后老师检查

Bellman-Ford

邻接表

代码2

压缩版本

```
fill(d,d+n+1,INF);  
d[1]=0;  
for(ll k=1;k<=n-1;++k)  
    for(ll u=1;u<=n;++u)  
        for(ll i=0;i<to[u].size();++i)  
            d[to[u][i]]=min()
```

易错点

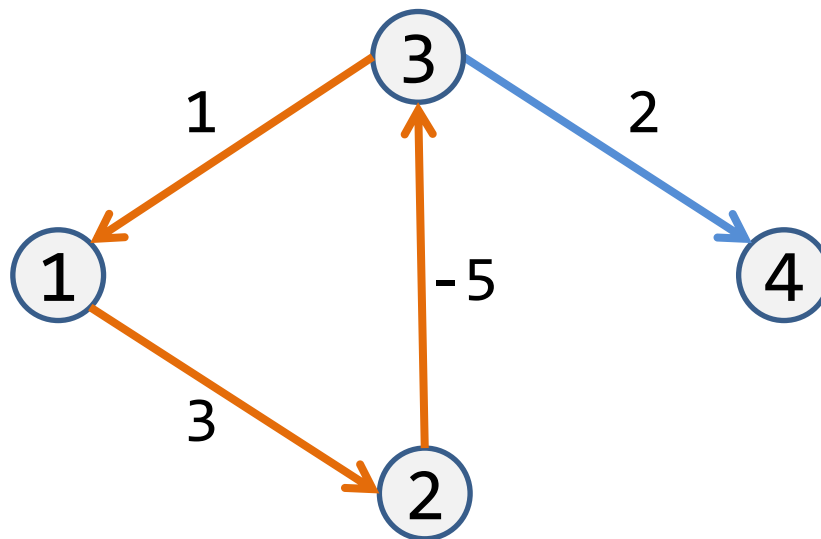
d[to[u][i]]写成d[i]

快如闪电
kkcoding.net

判断负环

Bellman-Ford算法
额外功能

如何判断负环是否存在？



若存在一个环路，权值总和为负数

沿负环不断行走，长度会被不断缩短

最短路数值无法确定

Bellman-Ford 判负环

$d[u]$ 记录从源点到 u 号点当前的最短路长度

初始化 $d[1]$ 为 0 ，其他 $d[u]$ 为 INF

主循环：以下操作重复共 $n-1$ 次

对于每条边：从 u 到 v ，权重 w

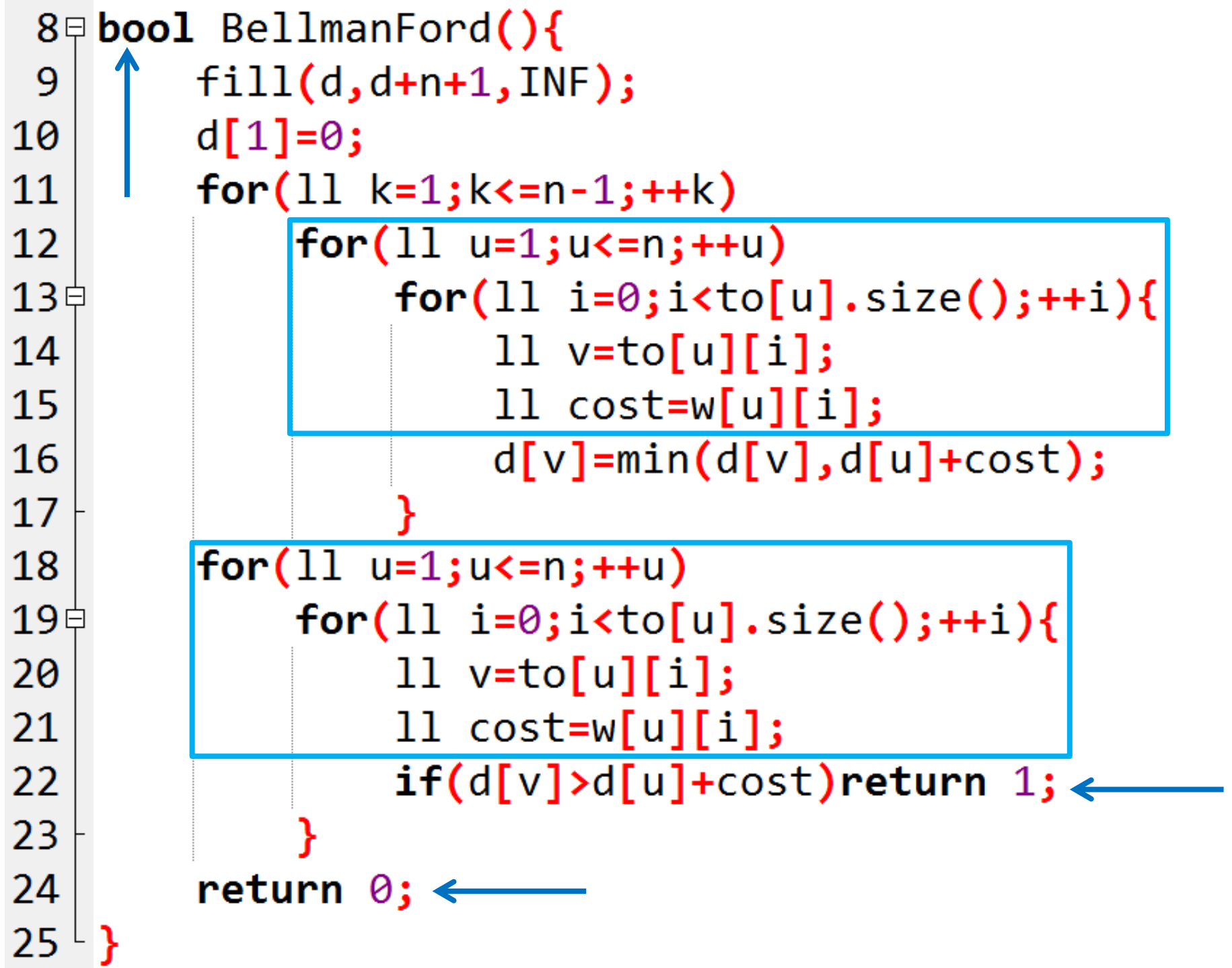
若 $d[u]+w < d[v]$ ，则更新
 $d[v] = d[u] + w$

额外再循环1次

对于每条边：从 u 到 v ，权重 w

若 $d[u]+w < d[v]$ ，则发现负环

```
8 bool BellmanFord(){
9     fill(d,d+n+1,INF);
10    d[1]=0;
11    for(ll k=1;k<=n-1;++k)
12        for(ll u=1;u<=n;++u)
13            for(ll i=0;i<to[u].size();++i){
14                ll v=to[u][i];
15                ll cost=w[u][i];
16                d[v]=min(d[v],d[u]+cost);
17            }
18    for(ll u=1;u<=n;++u)
19        for(ll i=0;i<to[u].size();++i){
20            ll v=to[u][i];
21            ll cost=w[u][i];
22            if(d[v]>d[u]+cost) return 1;
23        }
24    return 0;
25 }
```



The image shows a C++ implementation of the Bellman-Ford algorithm. The code is annotated with a line number column on the left (8 to 25) and blue arrows pointing to specific lines. A blue box highlights the inner loop of the first relaxation phase (lines 12-17). Another blue box highlights the second relaxation phase (lines 18-23). A blue arrow points to the return statement on line 22, and another points to the return statement on line 24.

快快编程590

快快编程
kkcoding.net

输入

3 1 2

3个地点,1种穿越方式,2条正常双向马路

3 1 8

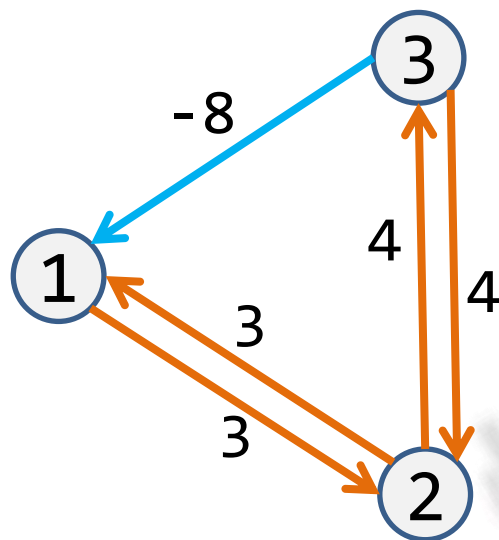
从3号地点穿越到1号地点,时间倒退8秒钟

1 2 3

1号地点和2号地点间有双向马路耗时3秒钟

2 3 4

2号地点和3号地点间有双向马路耗时4秒钟



把原问题已知条件转为图论信息

每个节点代表什么含义

每条边代表什么含义

边有没有方向?

共有几个节点?

n

共有几条边?

$p+m*2$

有权边还是无权边?

有没有负权边?

把求解的问题转为图论的问题描述

对有向有权图判断有没有负环

```
1  /*姓名XXX
2  图论建模:
3  已知:
4  每个节点代表1个地点
5  每条边代表2个地点间可以到达,可能是穿越或者走马路
6  穿越是单向边,马路是双向边
7  共有n个节点
8  共有 $p+2*m$ 条边
9
10  有权边,可能有负权
11  穿越对应负权
12  马路对应正权
13
14  求解:
15  判断有没有负环
16
17  方法:
18  用Bellman-Ford算法迭代n次
19  若还能更新说明有负环
20  */
```

请用电脑完成图论分析

2分钟后老师检查

```
49 cin>>n>>p>>m;
50 for(ll i=0;i<p;++i){
51     ll u,v,cost;
52     cin>>u>>v>>cost;
53     to[u].push_back(v);
54     w[u].push_back(-cost);
55 }
56 for(ll i=0;i<m;++i){
57     ll u,v,cost;
58     cin>>u>>v>>cost;
59     to[u].push_back(v);
60     w[u].push_back(cost);
61     to[v].push_back(u);
62     w[v].push_back(cost);
63 }
64 ll ans=BellmanFord();
65 if(ans)cout<<"Yes"<<endl;
66 else cout<<"No"<<endl;
```

共n个地点,p种穿越方式,m条正常双向马路

储存p种穿越方式

储存m条双向马路

快快编程
kkcoding.net

Bellman-Ford

优化加速

如何加速

请同学每人提出
1种加速方法

```
8 void BellmanFord(){
9     fill(d,d+n+1,INF);
10    d[1]=0;
11    for(ll k=1;k<=n-1;++k)
12        for(ll u=1;u<=n;++u)
13            for(ll i=0;i<to[u].size();++i){
14                ll v=to[u][i];
15                ll cost=w[u][i];
16                d[v]=min(d[v],d[u]+cost);
17            }
18 }
```

若某轮迭代中,所有 $d[]$ 都没更新过
那么 $d[]$ 再也不可能更新了

若 $d[u]$ 在上一轮迭代里没更新过
那么这一轮 $d[u]$ 不可能改进后续邻居

Bellman-Ford	Dijkstra	Floyd-Warshall
边核心	点核心	点核心
$O(VE)$	$O(V^2), O(E \log V)$	$O(V^3)$
SSSP	SSSP	任意两点最短路

稀疏图: $O(E) \approx O(V)$

不怕负边	怕负边	不怕负边
找负环	怕负环	找负环
边集数组	邻接矩阵	邻接矩阵
邻接表	邻接表	

可能提前结束

SSSP演示

算法可视化网址：
visualgo.net/en/sssp

The general purpose **Bellman Ford's algorithm** can solve **all kinds** of valid SSSP problem variants, with a **rather slow** $O(V \times E)$ running time. It also has an extremely simple pseudo-code.

快快编程
kkcoding.net

快快编程作业

589	用Dijkstra解
-----	------------

589	用Bellman-Ford解
-----	----------------

590
