# 信奥算法

# 最短路问题

## 复习Dijkstra算法

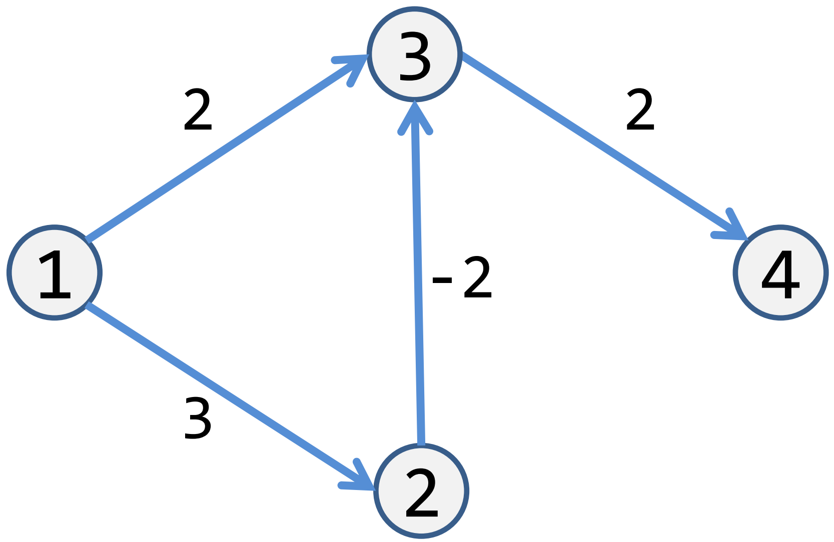
# 最短路问题

## 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

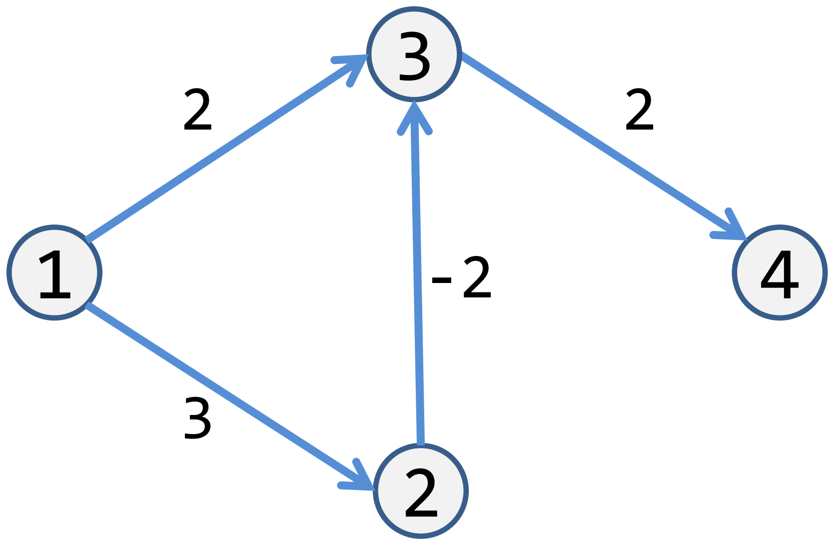



正确答案

| 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<br>d[3]获得了修改后续邻居的资格 | Dijkstra算法里<br>每条边最多只会<br>使用1次 |
| d[3]帮助修改d[4]一次后<br>d[4]再也没有机会改正了<br>虽然d[3]可以被d[2]改正 | 如何修改算法<br>使结果正确? |

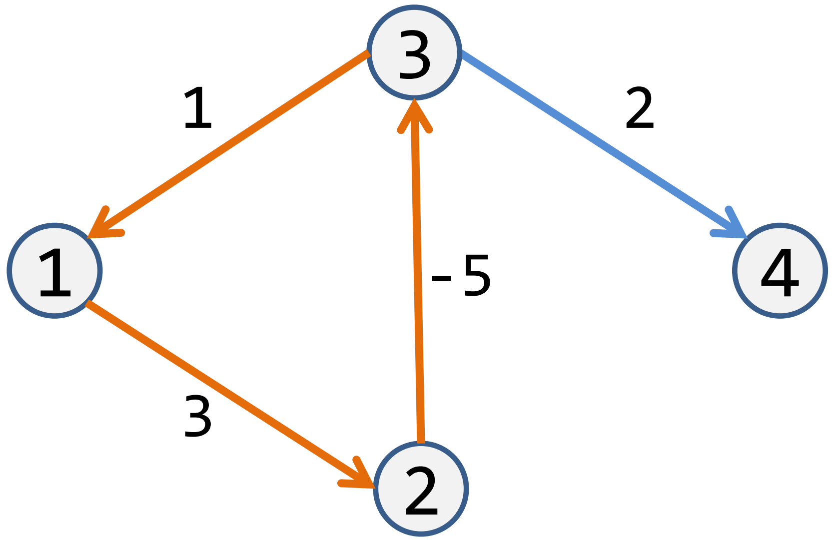
# 处理负边的算法

| | |
|---|---|
| 初始化 | d[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 |

最短路
的敌人

# 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

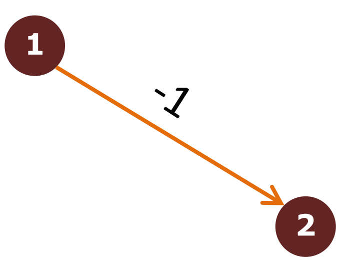
n个城市编号1到n。城市间共m条直飞单向航线，每条航线有起点，终点，和飞行时间。其中有些特殊航线的飞行时间为负数，代表时光倒流。求1号城市到各城市最少时间?

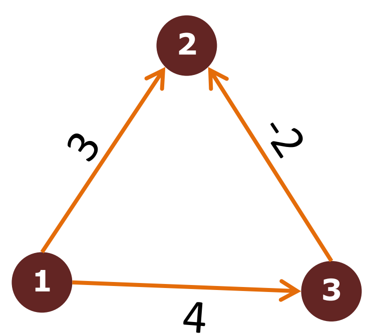
保证都能飞到，保证没有负环。n<=100,m<=200

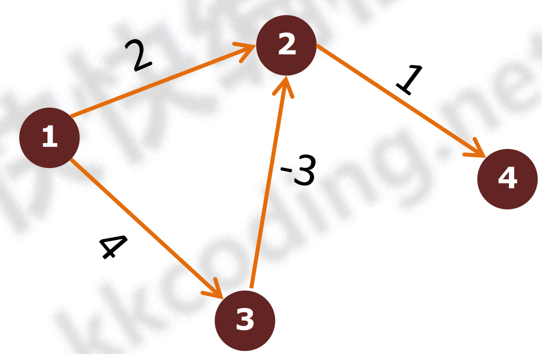
| 输入样例 | 输出样例 |
| --- | --- |
| 2 1<br>1 2 -1 | 0 -1 |

| 输入样例 | 输出样例 |
| --- | --- |
| 3 3<br>1 2 3<br>3 2 -2<br>1 3 4 | 0 2 4 |

| 输入样例 | 输出样例 |
| --- | --- |
| 4 5<br>1 2 2<br>1 3 4<br>3 2 -3<br>2 4 1 | 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

之后会揭秘

思考为什么n-1

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

复杂度是多少?

O(nm)

# Bellman-Ford

边集数组

代码1

```
typedef long long ll;
const ll N=109;
const ll M=209;
const ll INF=1e9;
struct edge{ll u,v,w;};
edge E[M];
ll n,m,d[N];
void BellmanFord(){
int main(){
    cin>>n>>m;
    for(ll i=0;i<m;++i)
        cin>>E[i].u>>E[i].v>>E[i].w;
    BellmanFord();
    for(ll u=1;u<=n;++u)
        cout<<d[u]<<" ";
    return 0;
}
```

| 易错点 | 无向边用双向边储存时<br>边数要翻倍M=409 |
|---|---|
| | u代表边的起点<br>v代表边的终点<br>w代表边的权值 |

# Bellman-Ford

边集数组

代码1

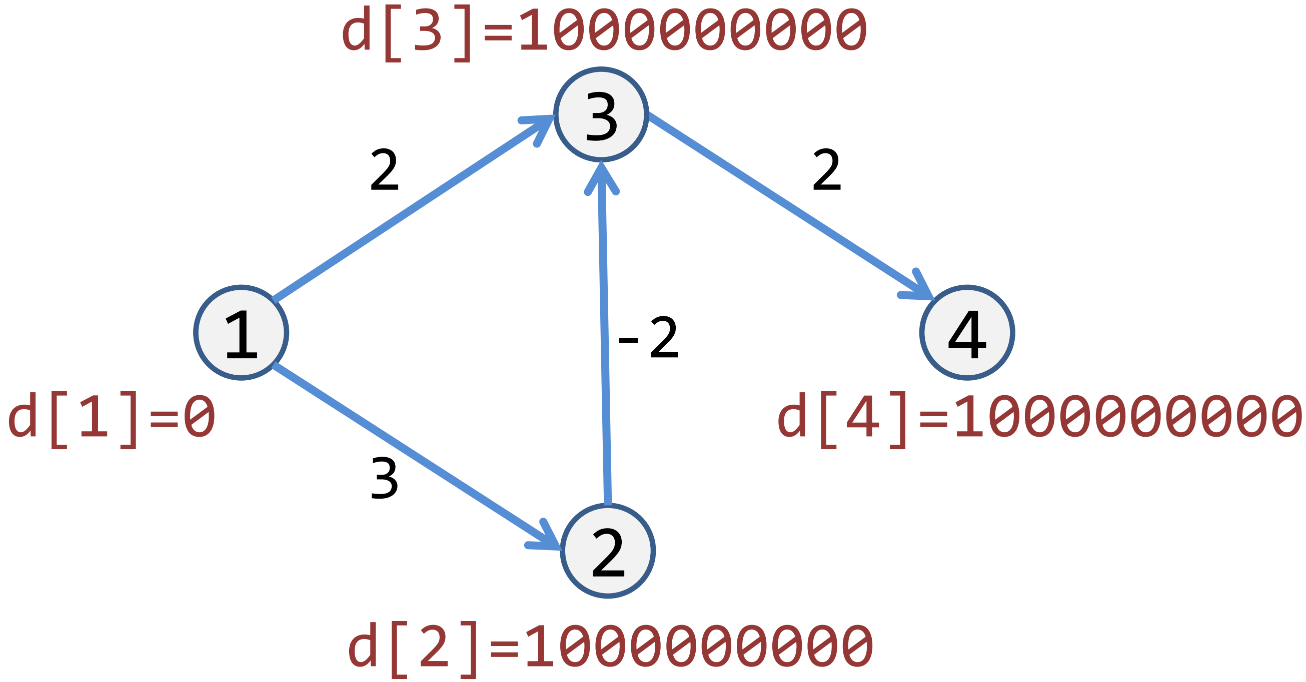
```
void BellmanFord(){
    fill(d,d+n+1,INF);
    d[1]=0;
    for(ll k=1;k<=n-1;++k)
        for(ll i=0;i<m;++i)
            d[E[i].v]=min(d[E[i].v],d[E[i].u]+E[i].w);
}
```

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

```
void BellmanFord(){
    fill(d,d+n+1,INF);

    for(ll k=1;k<=n-1;++k)
        for(ll i=0;i<m;++i)
            d[E[i].v]=
}
```

# 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

```
vector<ll> to[N],w[N];
ll n,m,d[N];
void BellmanFord(){
int main(){
    cin>>n>>m;
    for(ll i=0;i<m;++i){
        ll u,v,cost;
        cin>>u>>v>>cost;
        to[u].push_back(v);
        w[u].push_back(cost);
    }
    BellmanFord();
    for(ll u=1;u<=n;++u)
        cout<<d[u]<<" ";
    return 0;
}
```

# Bellman-Ford

邻接表

代码2

```
void BellmanFord(){
	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){
				ll v=to[u][i];
				ll cost=w[u][i];
				d[v]=min(d[v],d[u]+cost);
			}
}
```

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

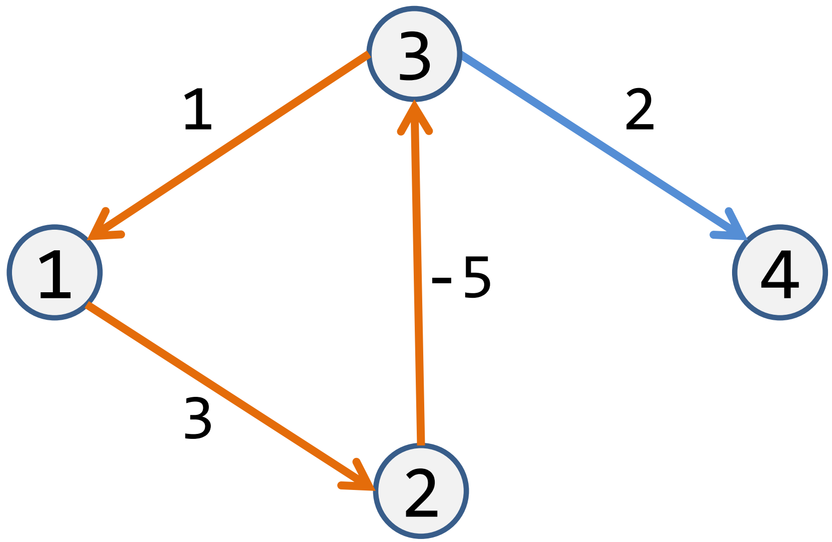
| |
|---|
| 邻接表 |
| 代码2 |
| 压缩版本 |

```
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] |

| 判断负环 |
| --- |
| **Bellman-Ford算法**<br>**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

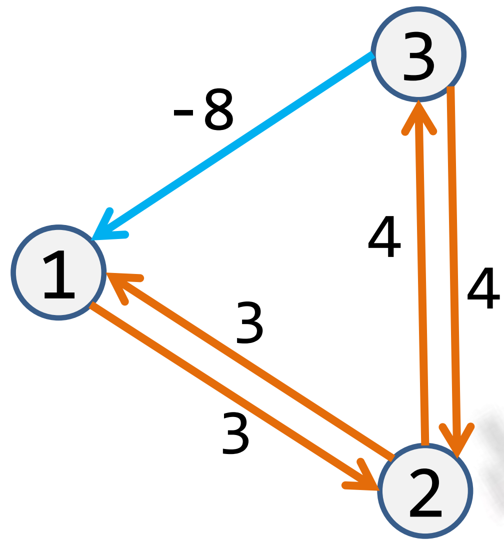
对于每条边：从u到v,权重w

若d[u]+w<d[v],则发现负环

```
bool BellmanFord(){
    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){
                ll v=to[u][i];
                ll cost=w[u][i];
                d[v]=min(d[v],d[u]+cost);
            }
    for(ll u=1;u<=n;++u)
        for(ll i=0;i<to[u].size();++i){
            ll v=to[u][i];
            ll cost=w[u][i];
            if(d[v]>d[u]+cost)return 1;
        }
    return 0;
}
```

快快编程590

| 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 |



图论建模

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

图论建模

```
/*姓名XXX
图论建模:
已知:
每个节点代表1个地点
每条边代表2个地点间可以到达,可能是穿越或者走马路
穿越是单向边,马路是双向边
共有n个节点
共有p+2*m条边

有权边,可能有负权
穿越对应负权
马路对应正权

求解:
判断有没有负环

方法:
用Bellman-Ford算法迭代n次
若还能更新说明有负环
*/
```

| |
|---|
| 请用电脑完成图论分析 |
| 2分钟后老师检查 |

共n个地点,p种穿越方式,m条正常双向马路

```
    cin>>n>>p>>m;
```

储存p种穿越方式

```
    for(ll i=0;i<p;++i){
        ll u,v,cost;
        cin>>u>>v>>cost;
        to[u].push_back(v);
        w[u].push_back(-cost);
    }
```

储存m条双向马路

```
    for(ll i=0;i<m;++i){
        ll u,v,cost;
        cin>>u>>v>>cost;
        to[u].push_back(v);
        w[u].push_back(cost);
        to[v].push_back(u);
        w[v].push_back(cost);
    }
    ll ans=BellmanFord();
    if(ans)cout<<"Yes"<<endl;
    else cout<<"No"<<endl;
```

**Bellman-Ford**

**优化加速**

# 如何加速

```
void BellmanFord(){
    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){
                ll v=to[u][i];
                ll cost=w[u][i];
                d[v]=min(d[v],d[u]+cost);
            }
}
```

请同学每人提出
1种加速方法

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**×**E**) running time. It also has an extremely simple pseudo-code.

# 快快编程作业

| | |
|---|---|
| 589 | 用Dijkstra解 |
| 589 | 用Bellman-Ford解 |
| 590 | |