图算法篇：单源最短路径问题之 **Bellman-Ford算法**

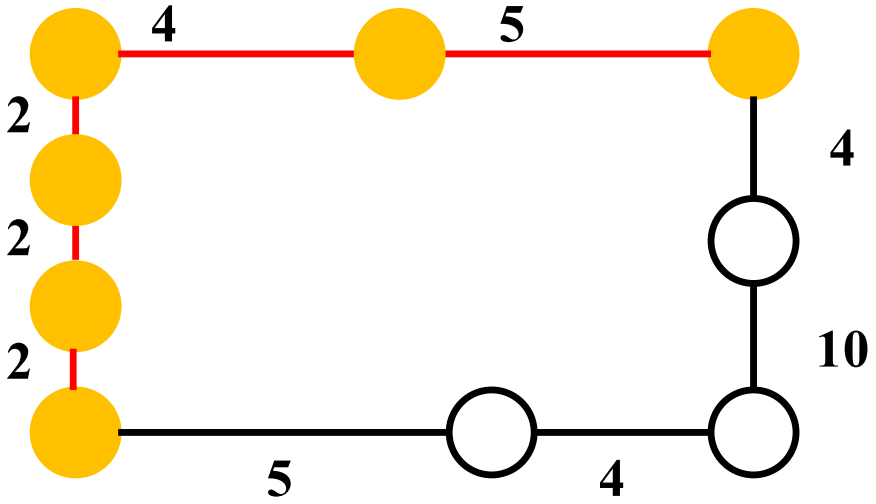
北京航空航天大学
计算机学院

提纲

- 问题背景
- 算法思想
- 算法实例
- 算法分析
- 算法性质

问题背景

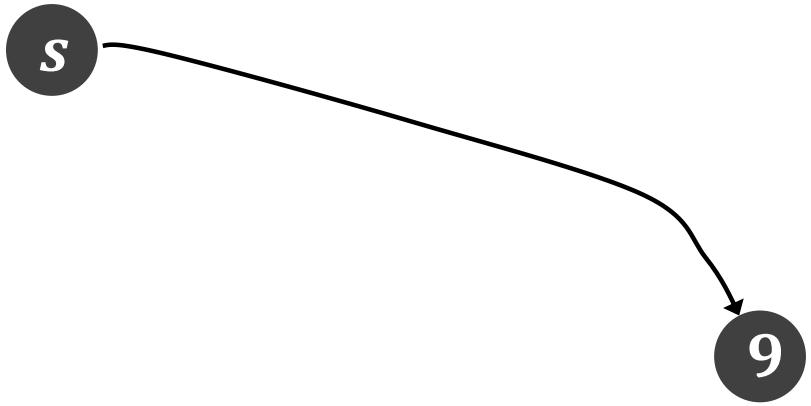
- 从知春路到其他站点，如何安排路线？



Dijkstra算法可以求解单源最短路径

回顾：Dijkstra 算法

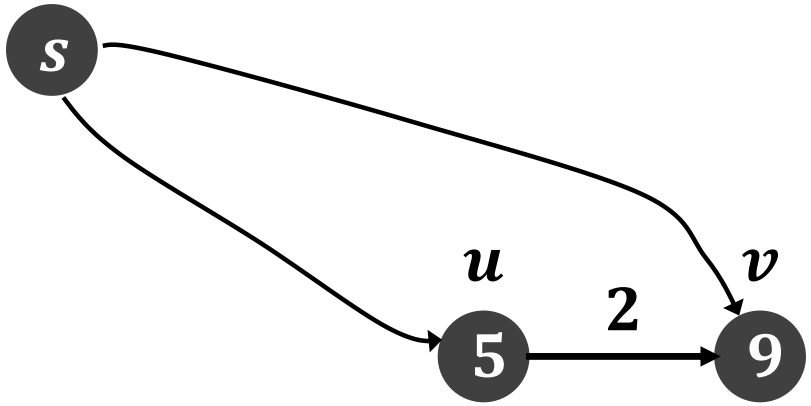
- 通过松弛操作迭代更新最短距离



Dijkstra算法适用范围：边权为正的图

回顾：Dijkstra 算法

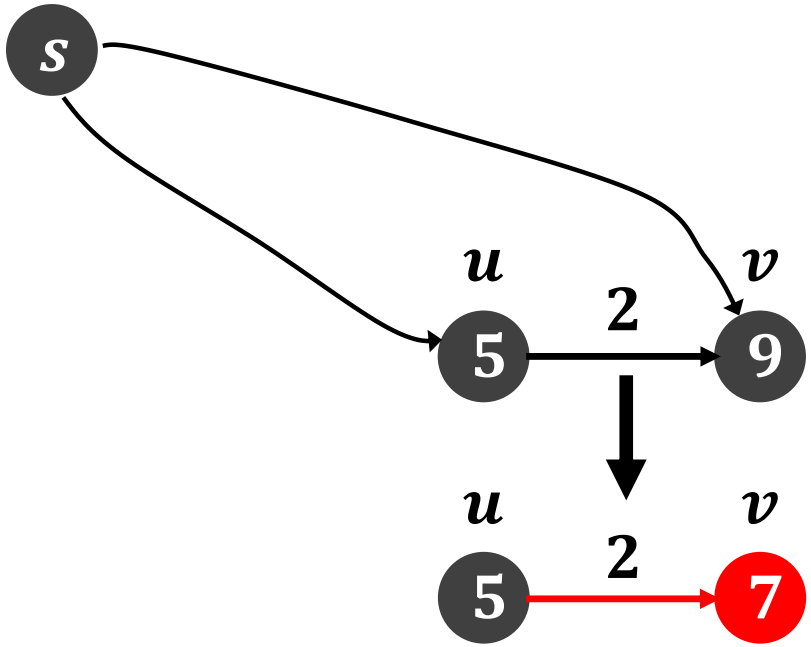
- 通过松弛操作迭代更新最短距离



Dijkstra算法适用范围：边权为正的图

回顾：Dijkstra 算法

- 通过松弛操作迭代更新最短距离

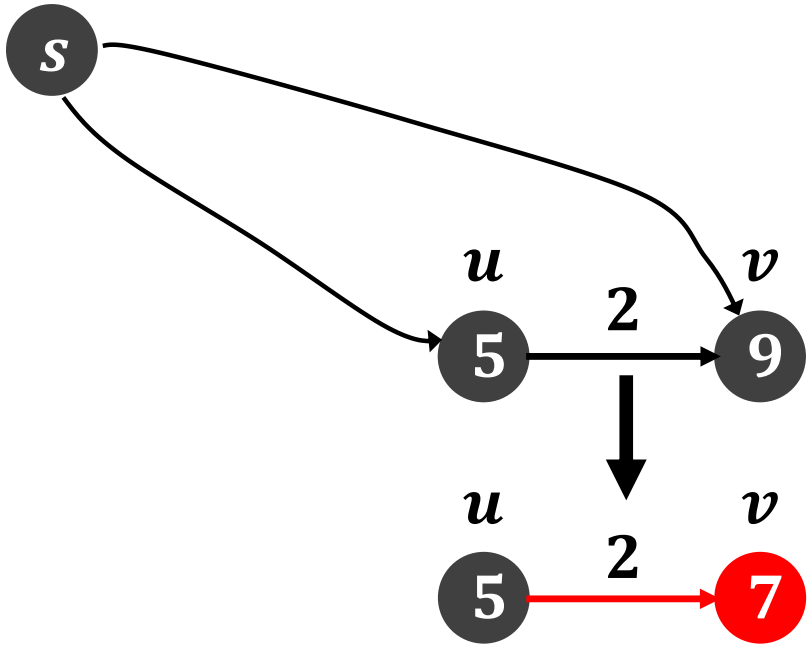


松弛成功

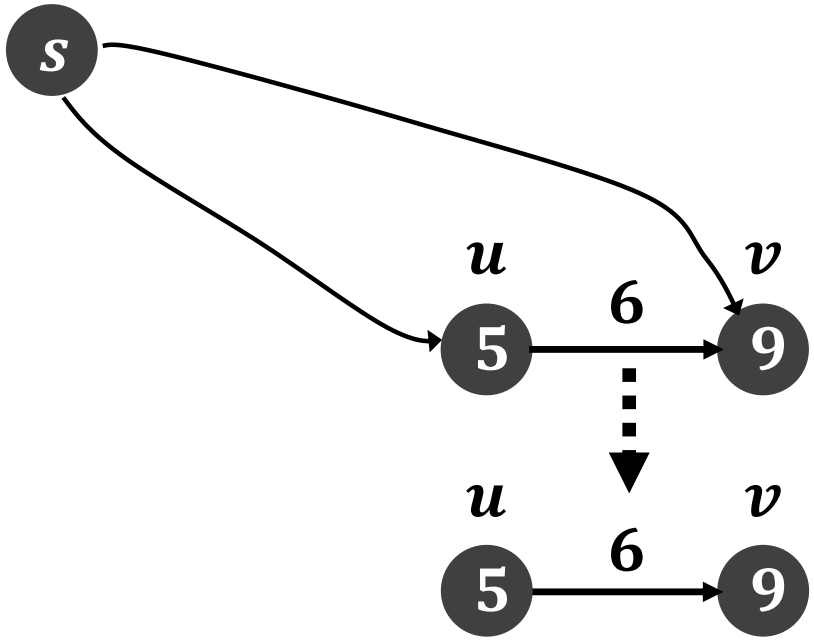
Dijkstra算法适用范围：边权为正的图

回顾：Dijkstra 算法

- 通过松弛操作迭代更新最短距离



松弛成功



松弛失败

Dijkstra算法适用范围：边权为正的图

回顾：Dijkstra 算法

输入: 图$G =< V, E, W >$, 源点s　时间复杂度：$O(|V| + |E|) \cdot \log|V|)$

输出: 单源最短路径P

新建一维数组$color[1..|V|]$, $dist[1..|V|]$, $pred[1..|V|]$

新建空优先队列Q

//初始化

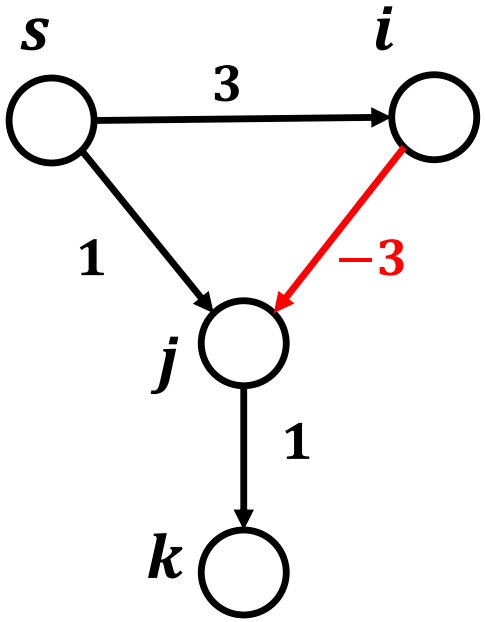
```
for u ∈ V do
    color[u] ← WHITE
    dist[u] ← ∞
    pred[u] ← NULL
end
dist[s] ← 0
Q.Insert(V, dist)
```

//执行单源最短路径算法

```
while 优先队列Q非空 do
    v ← Q.ExtractMin()
    for u ∈ G.adj[v] do
        if dist[v] + w(v, u) < dist[u] then
            dist[u] ← dist[v] + w(v, u)
            pred[u] ← v
            Q.DecreaseKey((u, dist[u]))
        end
    end
    color[v] ← BLACK
end
```

问题背景

- 图中存在负权边，Dijkstra算法不再适用



V_A中顶点

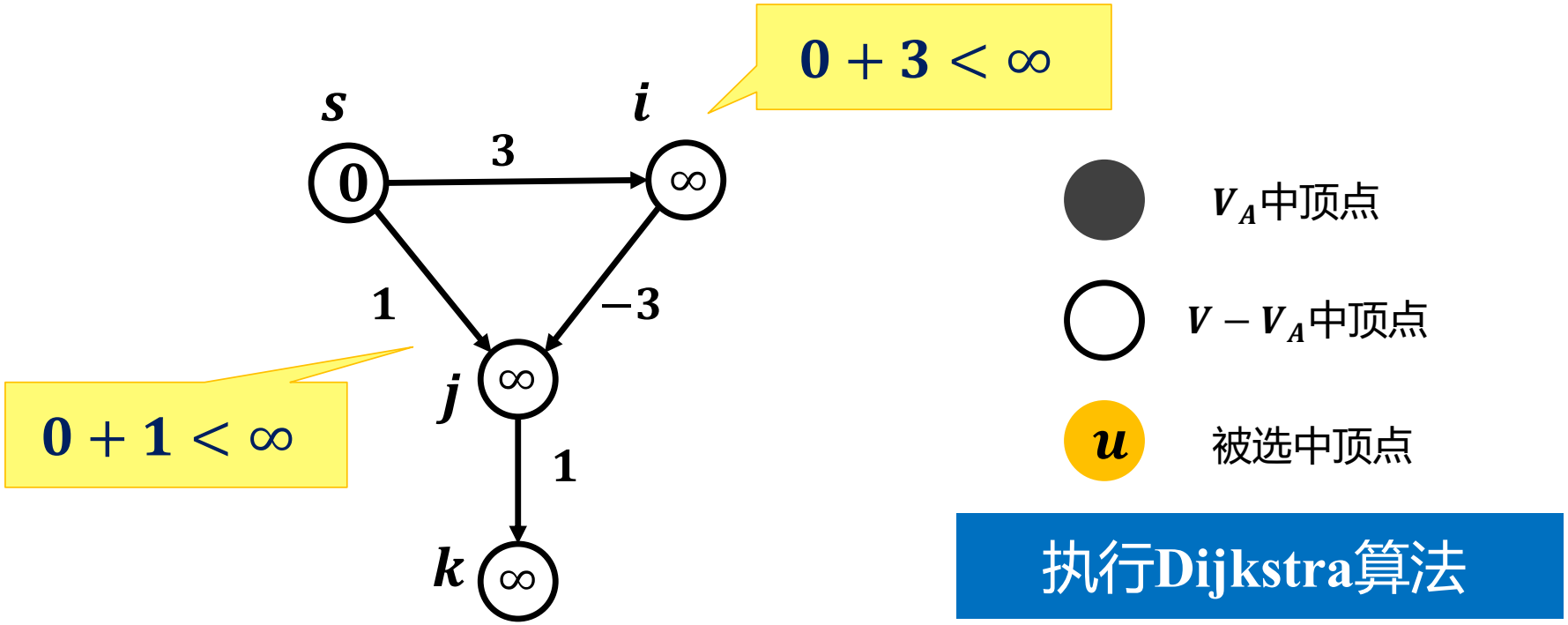
$V - V_A$中顶点

u 被选中顶点

执行Dijkstra算法

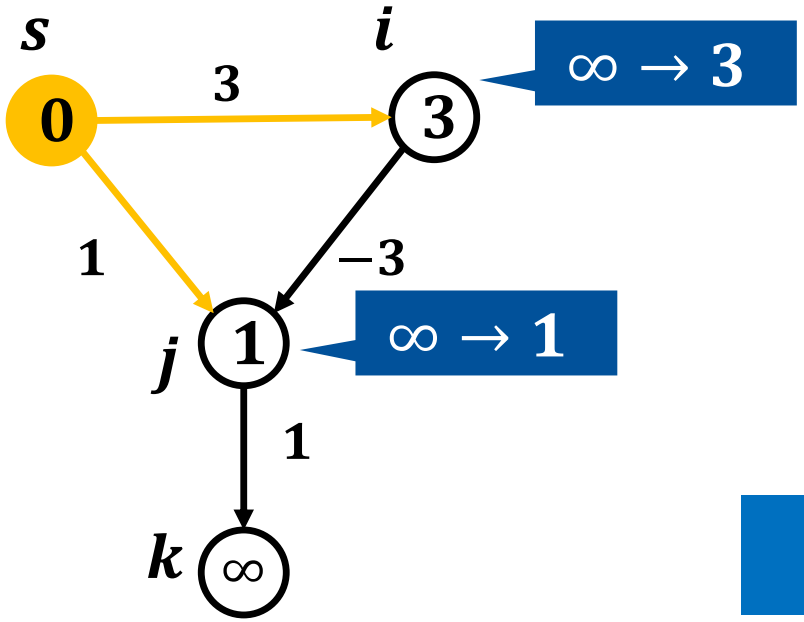
问题背景

- 图中存在负权边，Dijkstra算法不再适用



问题背景

- 图中存在负权边，Dijkstra算法不再适用



V_A中顶点

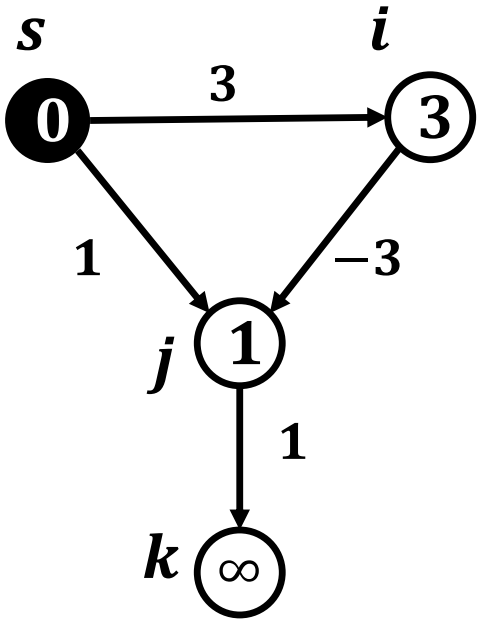
$V - V_A$中顶点

u 被选中顶点

执行Dijkstra算法

问题背景

- 图中存在负权边，Dijkstra算法不再适用



V_A中顶点

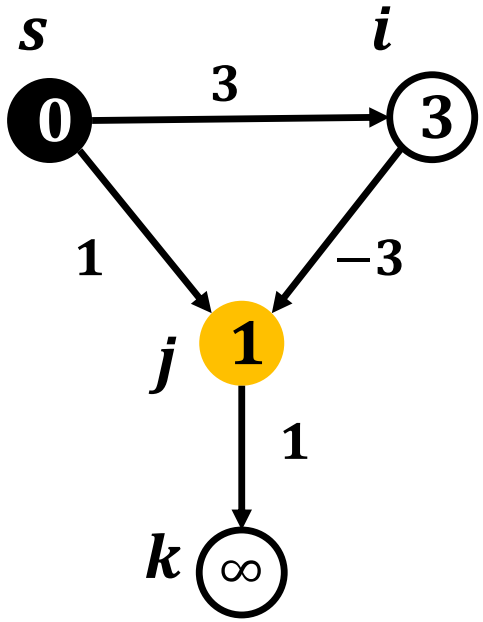
$V - V_A$中顶点

u 被选中顶点

执行Dijkstra算法

问题背景

- 图中存在负权边，Dijkstra算法不再适用



V_A中顶点

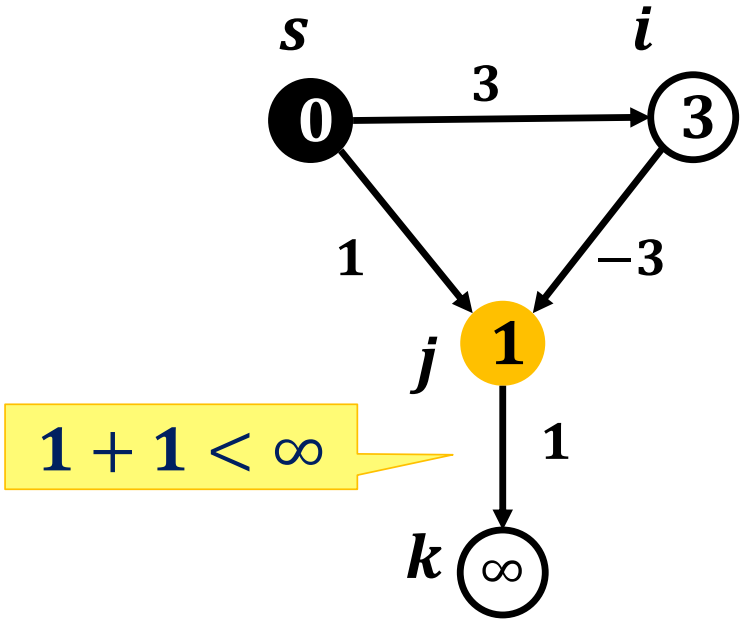
$V - V_A$中顶点

u 被选中顶点

执行Dijkstra算法

问题背景

- 图中存在负权边，Dijkstra算法不再适用



V_A中顶点

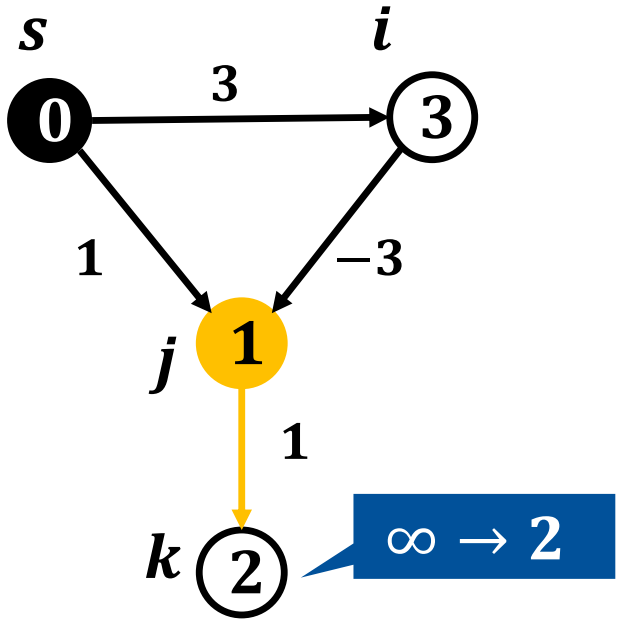
$V - V_A$中顶点

u 被选中顶点

执行Dijkstra算法

问题背景

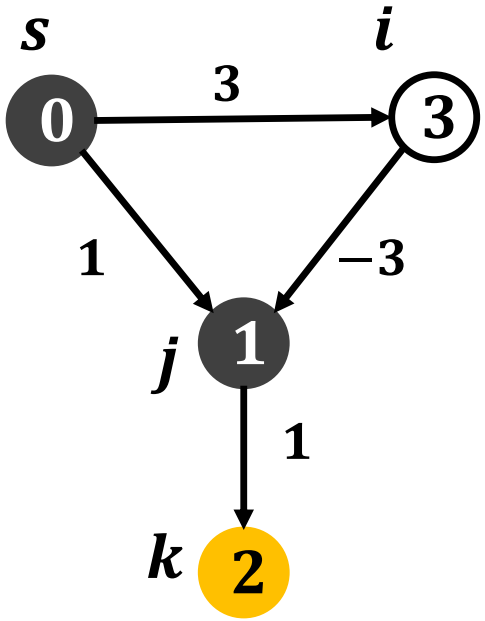
- 图中存在负权边，Dijkstra算法不再适用



执行Dijkstra算法

问题背景

- 图中存在负权边，Dijkstra算法不再适用



V_A中顶点

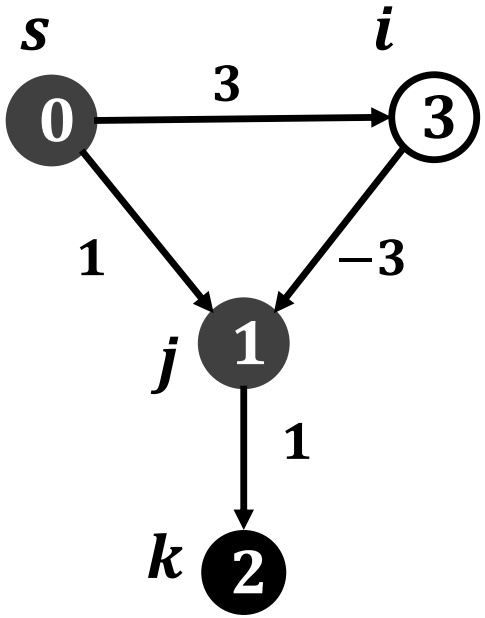
$V - V_A$中顶点

u 被选中顶点

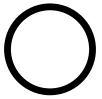
执行Dijkstra算法

问题背景

- 图中存在负权边，Dijkstra算法不再适用




V_A中顶点

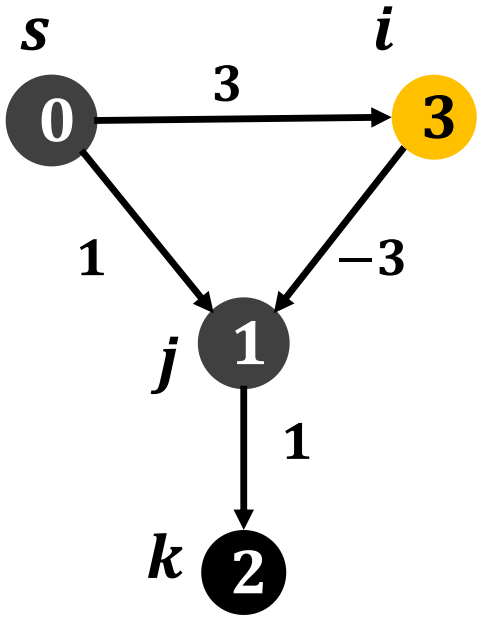

$V - V_A$中顶点

被选中顶点

执行Dijkstra算法

问题背景

- 图中存在负权边，Dijkstra算法不再适用



V_A中顶点

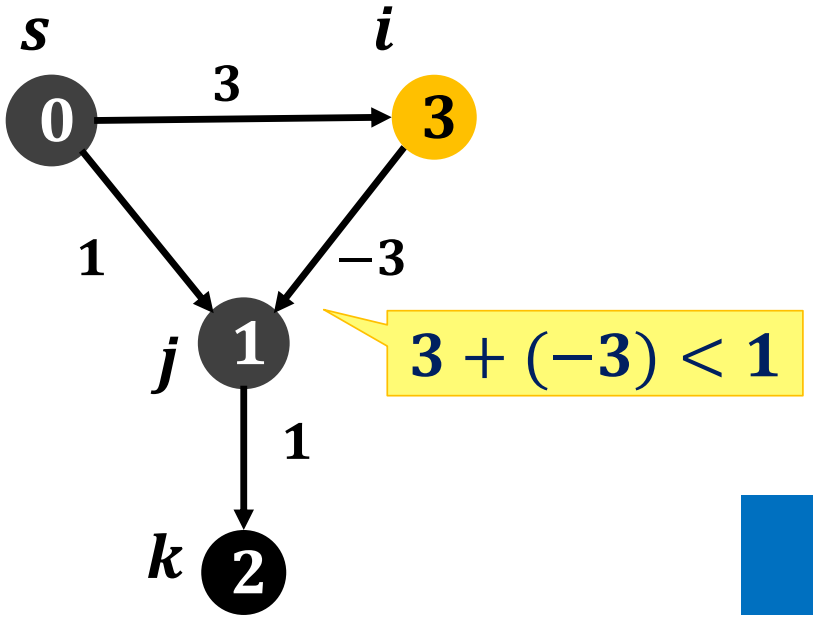
$V - V_A$中顶点

u 被选中顶点

执行Dijkstra算法

问题背景

- 图中存在负权边，Dijkstra算法不再适用



V_A中顶点

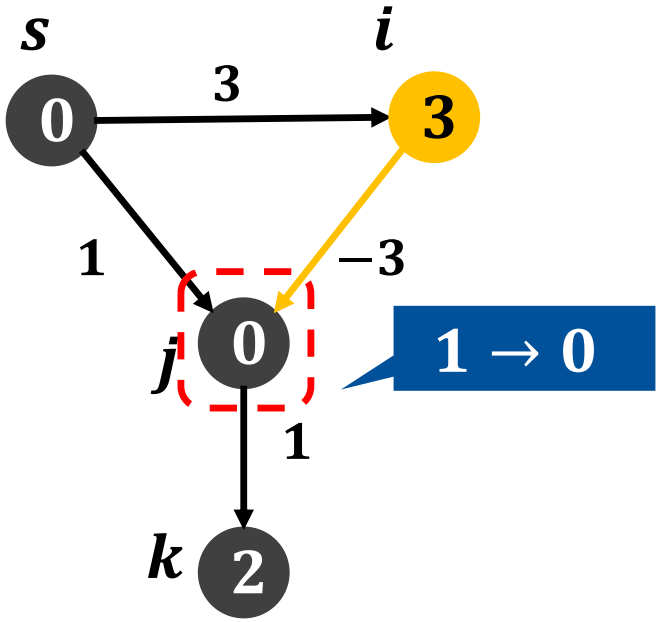
$V - V_A$中顶点

u 被选中顶点

执行Dijkstra算法

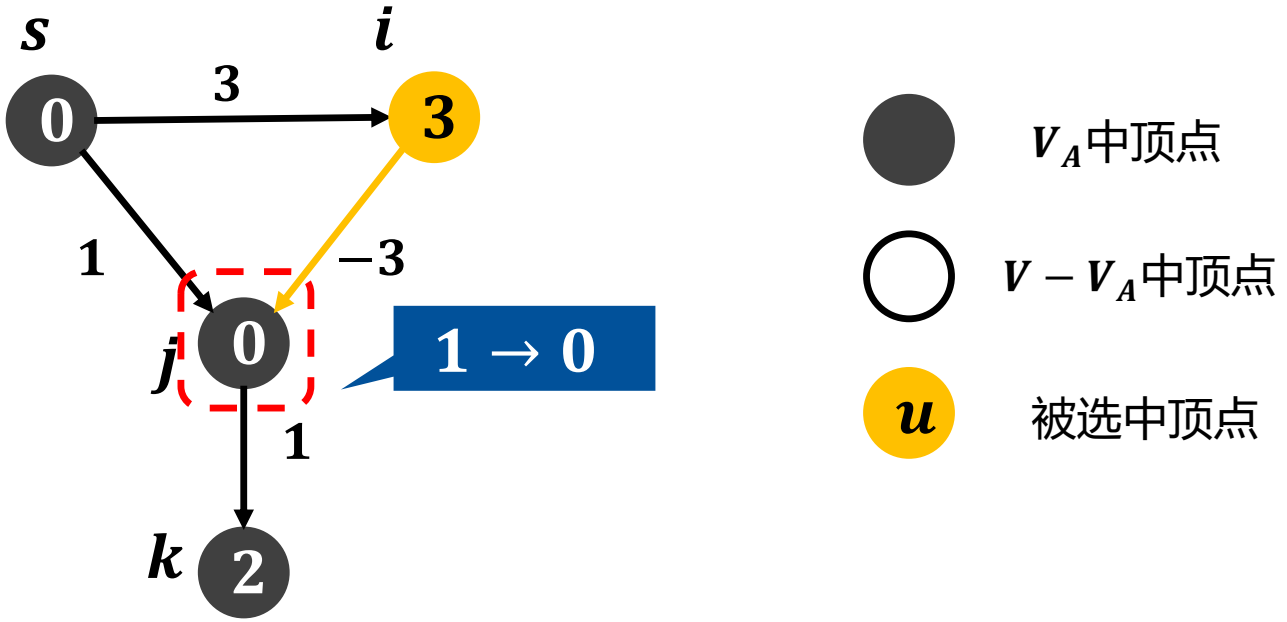
问题背景

- 图中存在负权边，Dijkstra算法不再适用



问题背景

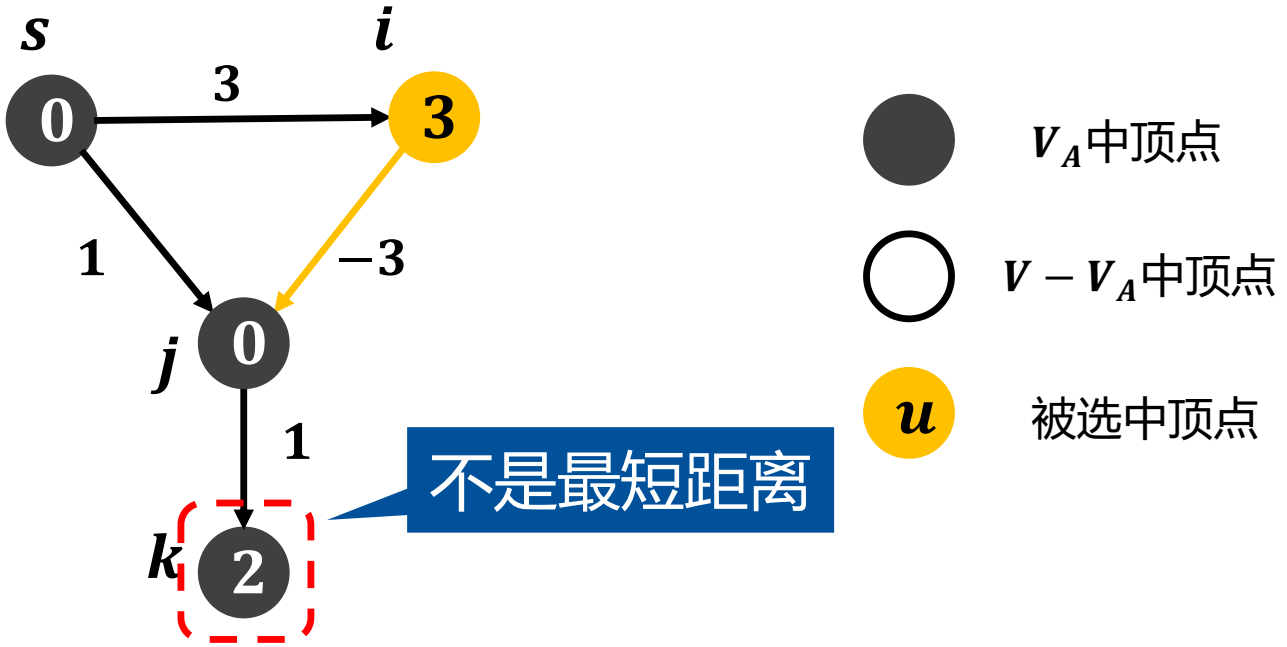
- 图中存在负权边，Dijkstra算法不再适用



Dijkstra算法：到黑色顶点的最短路应该已经计算出

问题背景

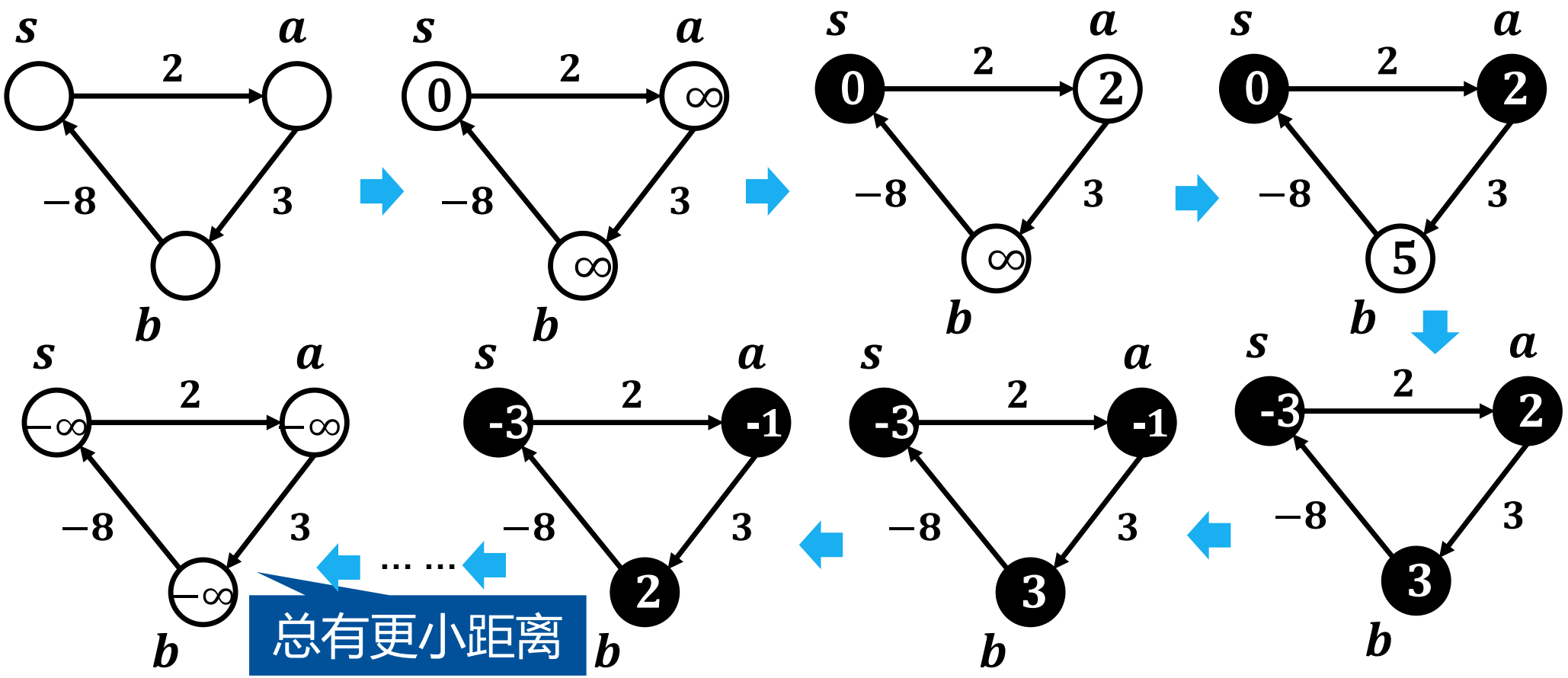
- 图中存在负权边，Dijkstra算法不再适用



Dijkstra算法：到黑色顶点的最短路应该已经计算出

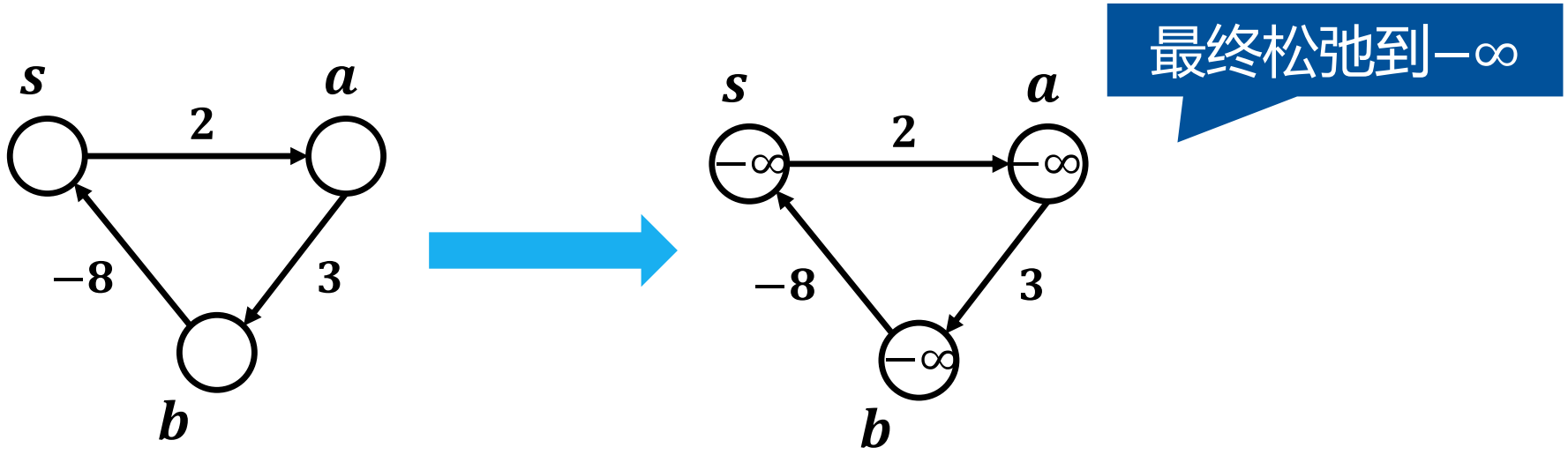
问题背景

- 图中存在负权边，Dijkstra算法不再适用
- 如果源点 s 可达负环，则难以定义最短路径



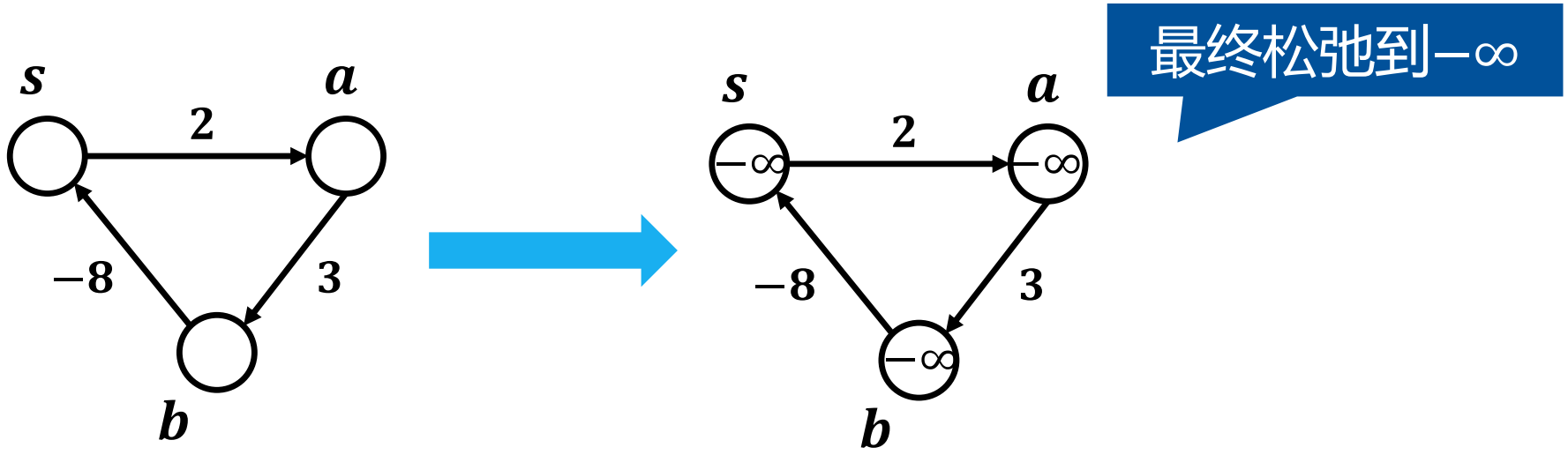
问题背景

- 图中存在负权边，Dijkstra算法不再适用
- 如果源点 s 可达负环，则难以定义最短路径



问题背景

- 图中存在负权边，Dijkstra算法不再适用
- 如果源点 s 可达负环，则难以定义最短路径



若源点 s 无可达负环，则存在源点 s 的单源最短路径

问题定义

单源最短路径问题

Single Source Shortest Paths Problem

输入

- 带权图$\boldsymbol{G} =< \boldsymbol{V}, \boldsymbol{E}, \boldsymbol{W} >$
- 源点编号 $\boldsymbol{s}$

问题定义

单源最短路径问题

Single Source Shortest Paths Problem

输入

- 带权图$\boldsymbol{G} =< \boldsymbol{V}, \boldsymbol{E}, \boldsymbol{W} >$
- 源点编号 $\boldsymbol{s}$

输出

- 源点 $\boldsymbol{s}$ 到所有其他顶点 $\boldsymbol{t}$ 的最短距离 $\boldsymbol{\delta(s,t)}$ 和最短路径 $< \boldsymbol{s}, \ldots, \boldsymbol{t} >$
- 或存在源点 $\boldsymbol{s}$ 可达的负环

问题定义

单源最短路径问题

Single Source Shortest Paths Problem

输入

- 带权图$\boldsymbol{G} =< \boldsymbol{V}, \boldsymbol{E}, \boldsymbol{W} >$
- 源点编号 $\boldsymbol{s}$

输出

- 源点 $\boldsymbol{s}$ 到所有其他顶点 $\boldsymbol{t}$ 的最短距离 $\boldsymbol{\delta(s,t)}$ 和最短路径 $< \boldsymbol{s}, \ldots, \boldsymbol{t} >$
- 或存在源点 $\boldsymbol{s}$ 可达的负环

挑战1：图中存在负权边时，如何求解单源最短路径?

问题定义

单源最短路径问题

Single Source Shortest Paths Problem

输入

- 带权图$G =< V, E, W >$
- 源点编号 s

输出

- 源点 s 到所有其他顶点 t 的最短距离 $\delta(s, t)$ 和最短路径 $< s, \dots, t >$
- 或存在源点 s 可达的负环

挑战1：图中存在负权边时，如何求解单源最短路径?

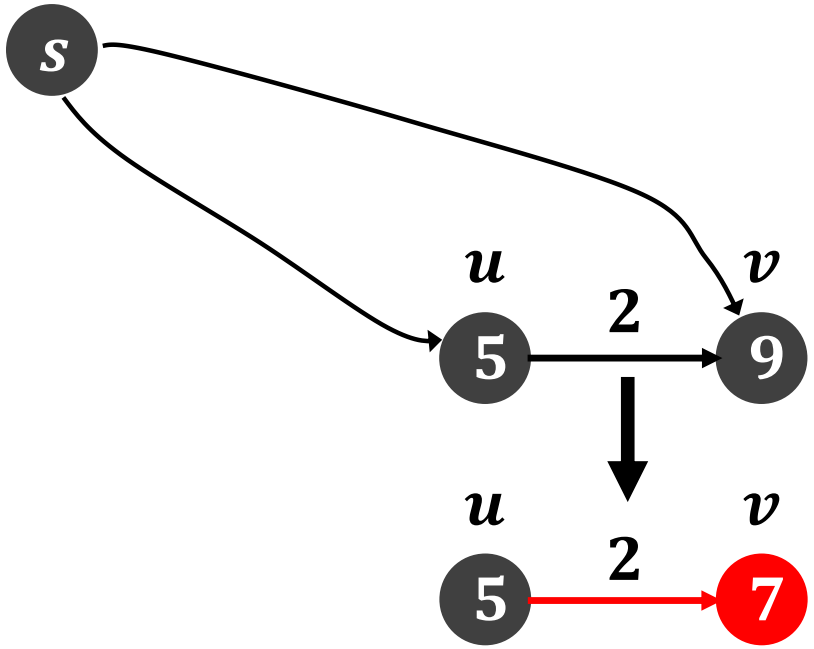
挑战2：图中存在负权边时，如何发现源点可达负环?

提纲

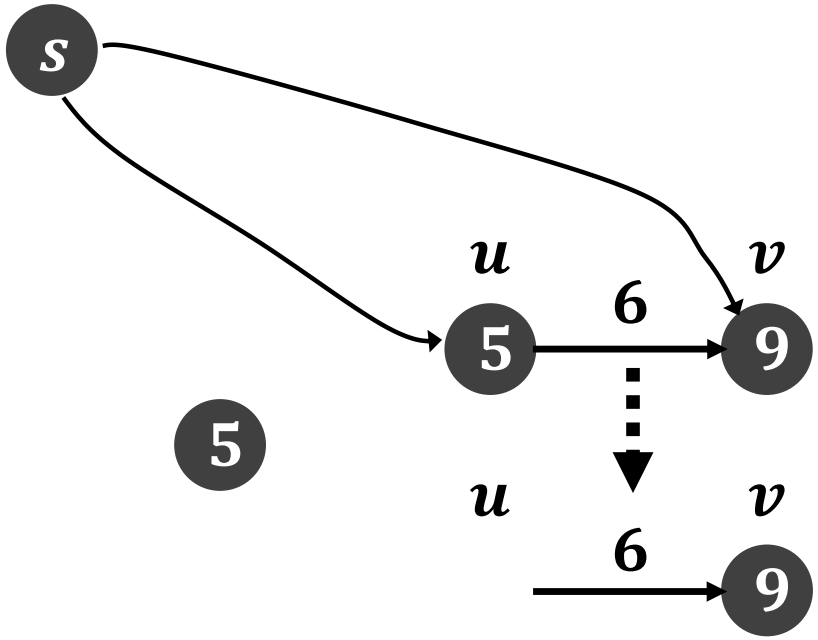
- 问题背景
- 算法思想
- 算法实例
- 算法分析
- 算法性质

算法思想

- **Dijkstra**算法通过松弛操作迭代更新最短距离



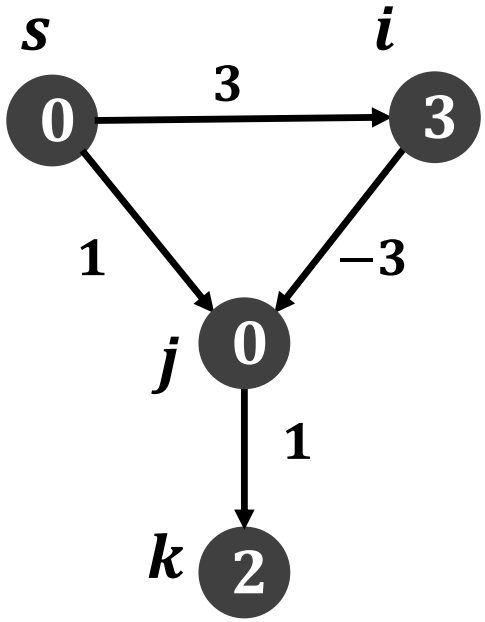
松弛成功



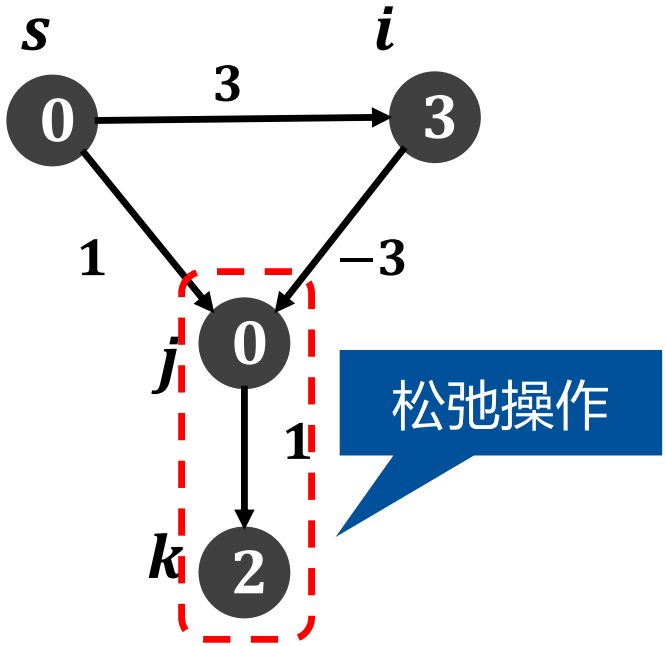
松弛失败

算法思想

- 存在负权边时，需要比Dijkstra算法更多次数的松弛操作



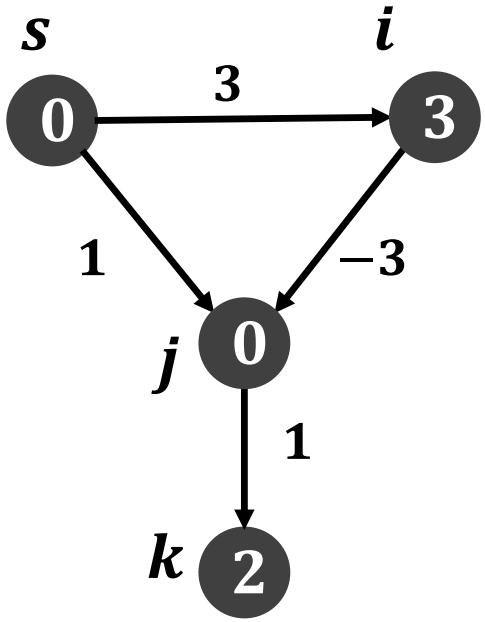
*Dijkstra*算法



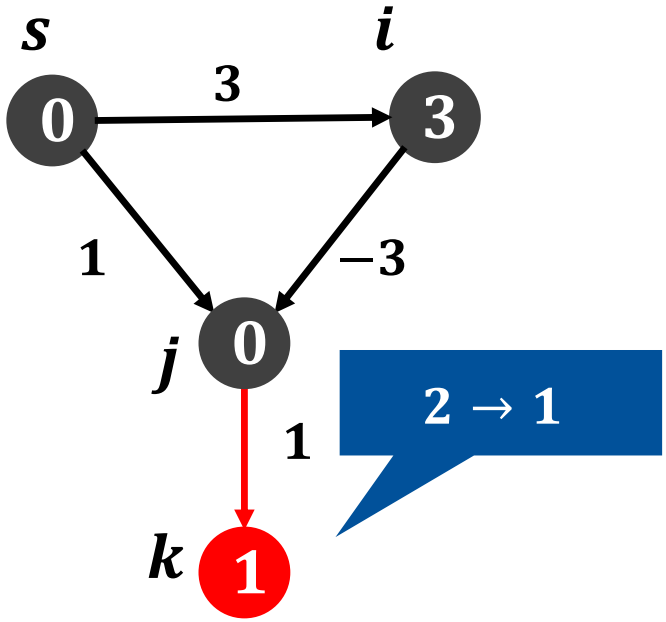
目标算法

算法思想

- 存在负权边时，需要比Dijkstra算法更多次数的松弛操作



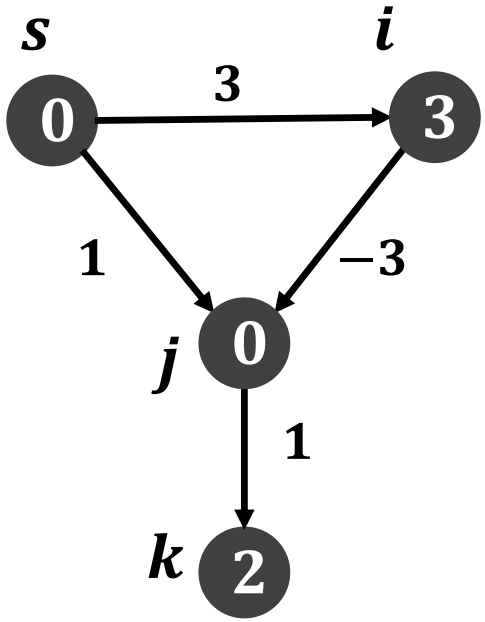
$Dijkstra$算法



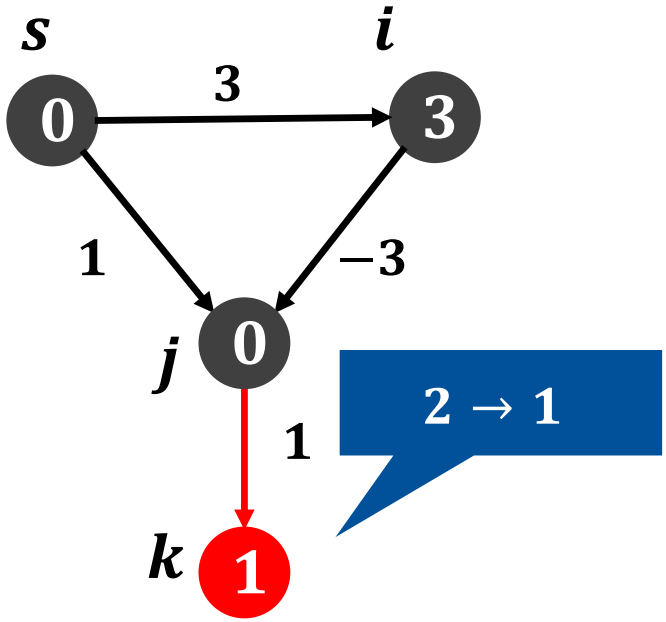
目标算法

算法思想

- 存在负权边时，需要比Dijkstra算法更多次数的松弛操作



*Dijkstra*算法

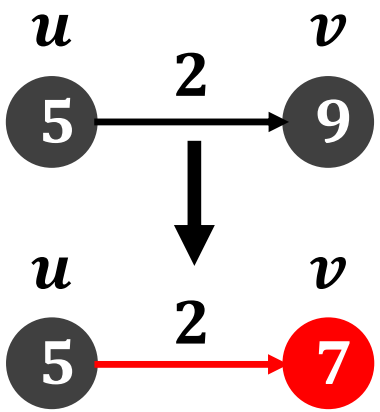


目标算法

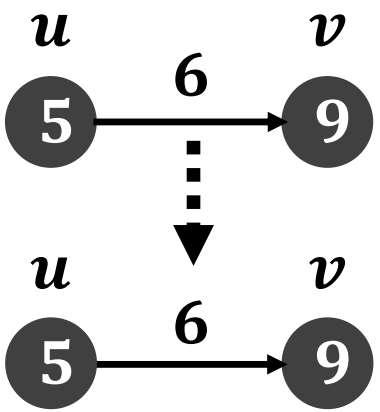
问题：图中存在负权边时，如何利用松弛操作求解单源最短路？

Bellman-Ford 算法思想

- 图中存在负权边时：每轮对所有边进行松弛，持续迭代 $|V|-1$轮



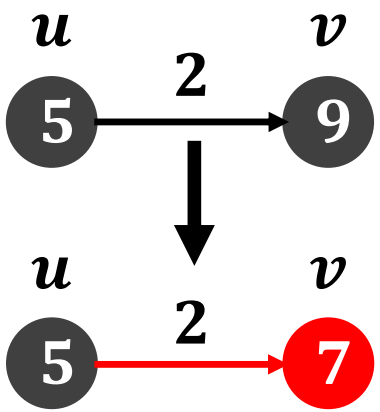
松弛成功



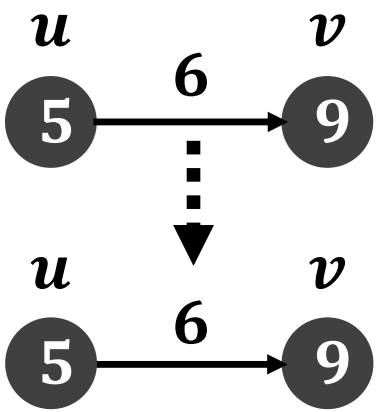
松弛失败

Bellman-Ford 算法思想

- 图中存在负权边时：每轮对所有边进行松弛，持续迭代 $|V|-1$轮
- 图中存在负权边时：若第 $|V|$ 轮仍松弛成功，存在源点s可达的负环



松弛成功



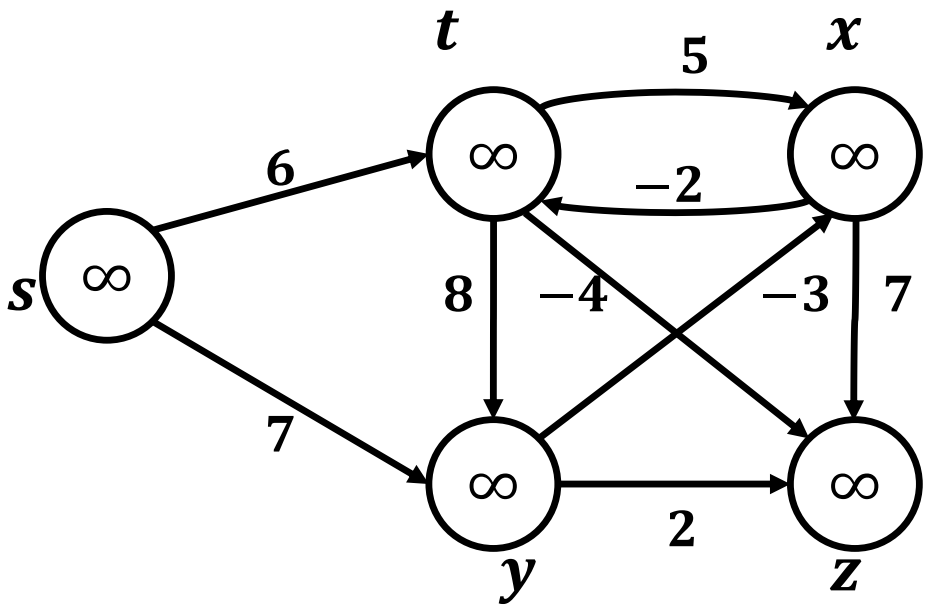
松弛失败

提纲

- 问题背景
- 算法思想
- 算法实例
- 算法分析
- 算法性质

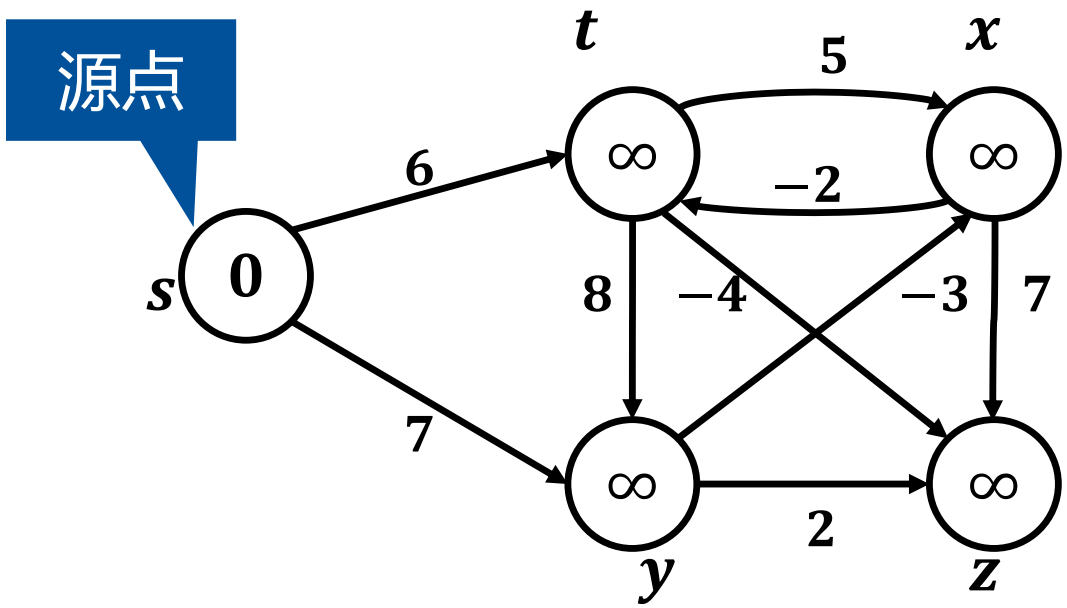
算法实例

V	s	t	x	y	z
$pred$	N	N	N	N	N
$dist$	∞	∞	∞	∞	∞



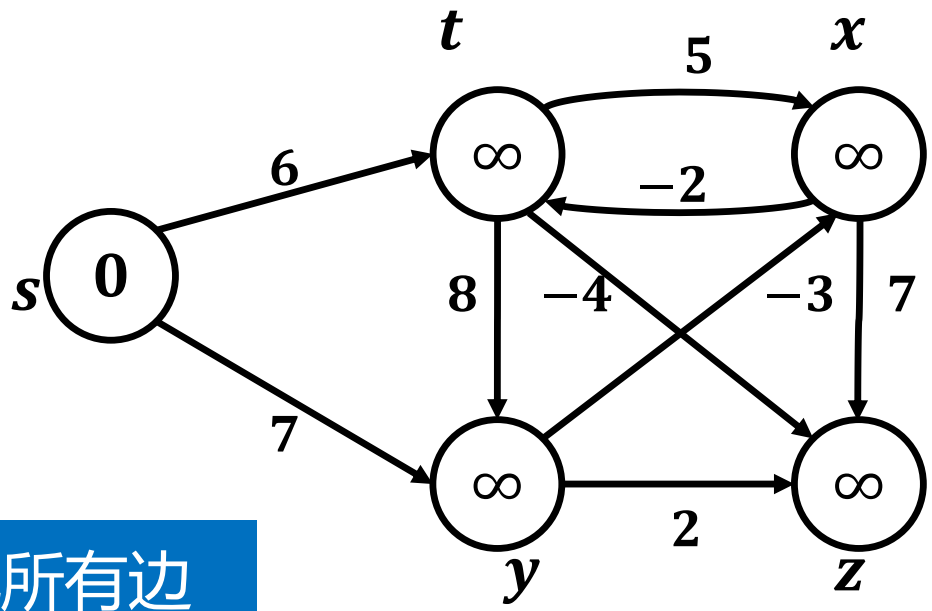
算法实例

V	s	t	x	y	z
$pred$	N	N	N	N	N
$dist$	0	∞	∞	∞	∞



算法实例

V	s	t	x	y	z
$pred$	N	N	N	N	N
$dist$	0	∞	∞	∞	∞



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

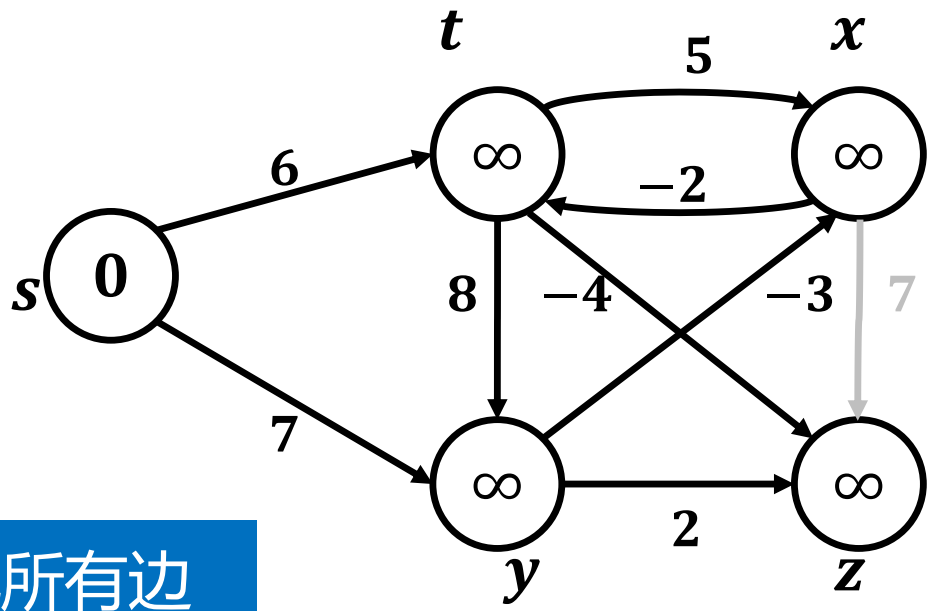
松弛失败

松弛成功

第1轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	N	N	N	N
$dist$	0	∞	∞	∞	∞



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

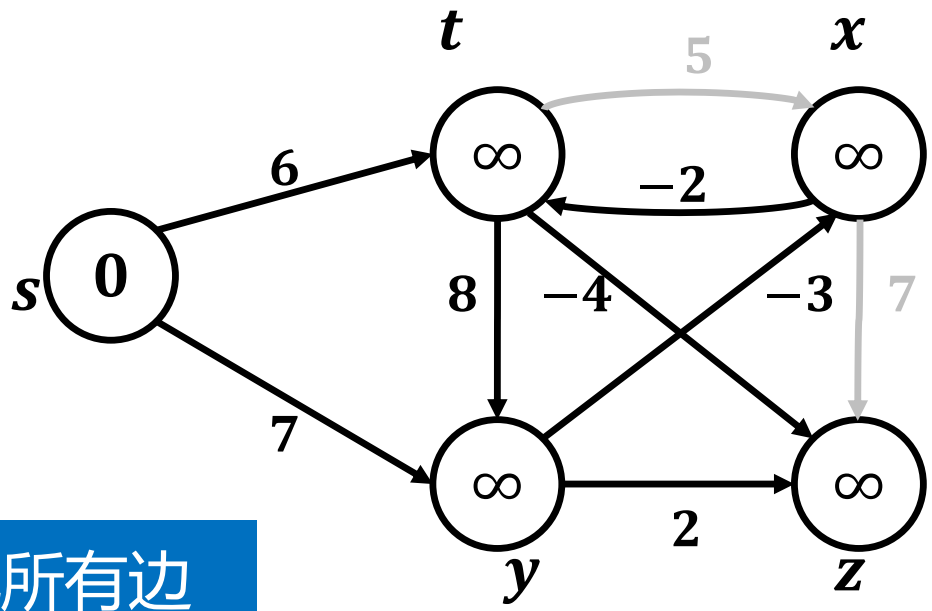
松弛失败

松弛成功

第1轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	N	N	N	N
$dist$	0	∞	∞	∞	∞



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

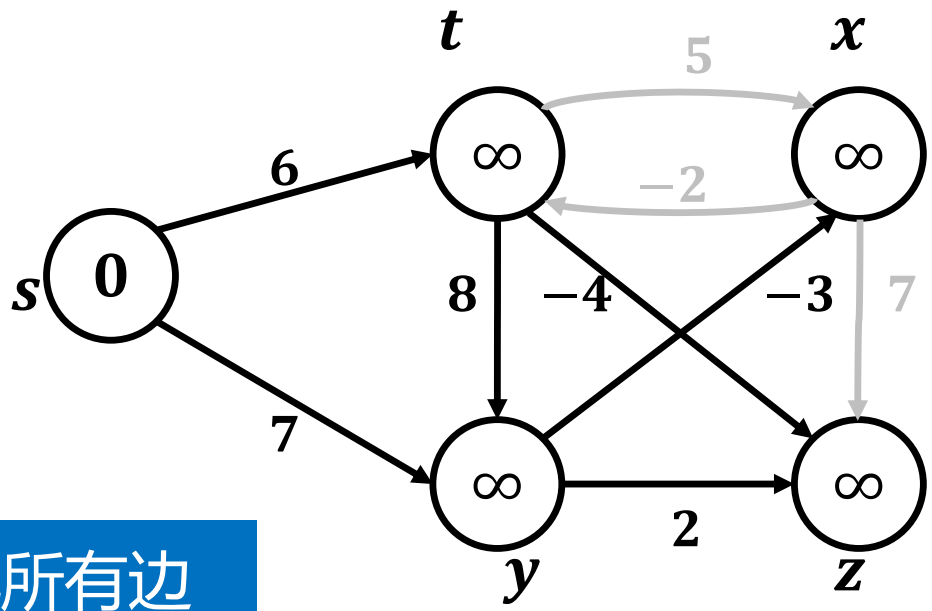
松弛失败

松弛成功

第1轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	N	N	N	N
$dist$	0	∞	∞	∞	∞



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

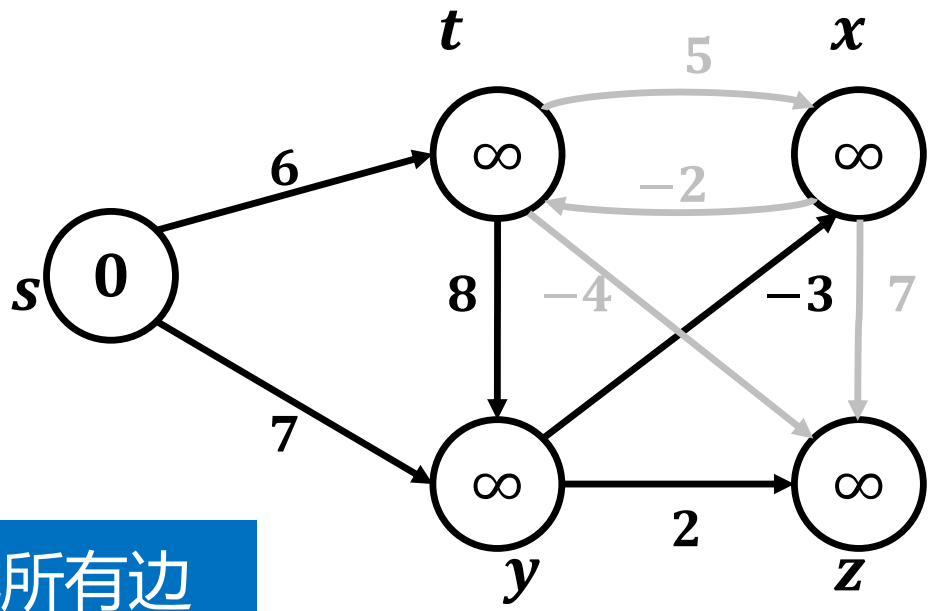
松弛失败

松弛成功

第1轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	N	N	N	N
$dist$	0	∞	∞	∞	∞



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

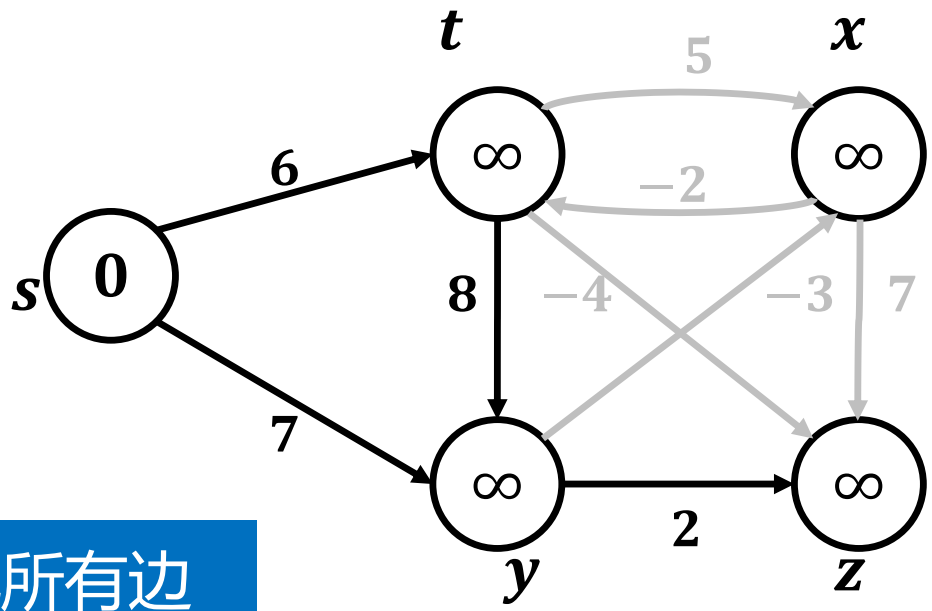
松弛失败

松弛成功

第1轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	N	N	N	N
$dist$	0	∞	∞	∞	∞



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

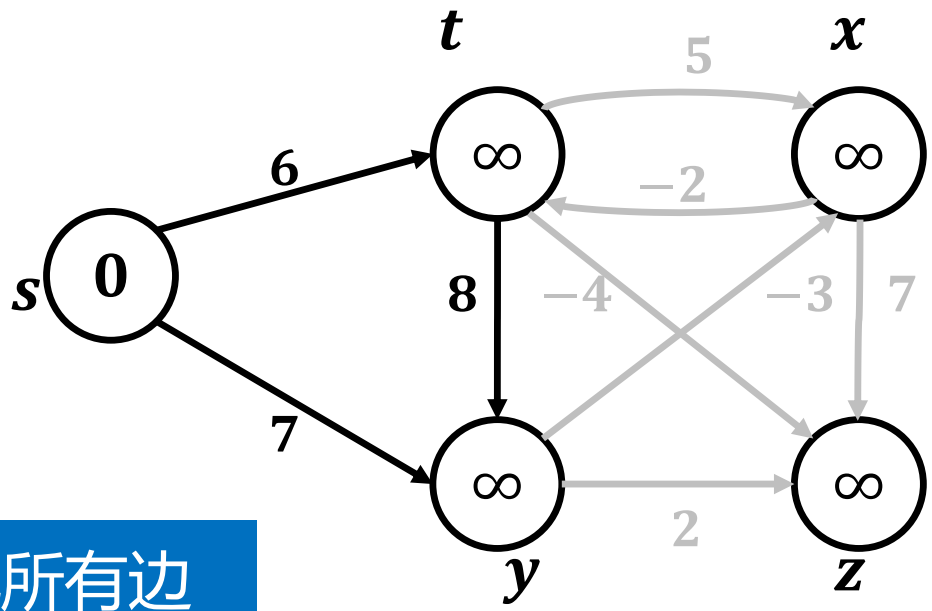
松弛失败

松弛成功

第1轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	N	N	N	N
$dist$	0	∞	∞	∞	∞



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

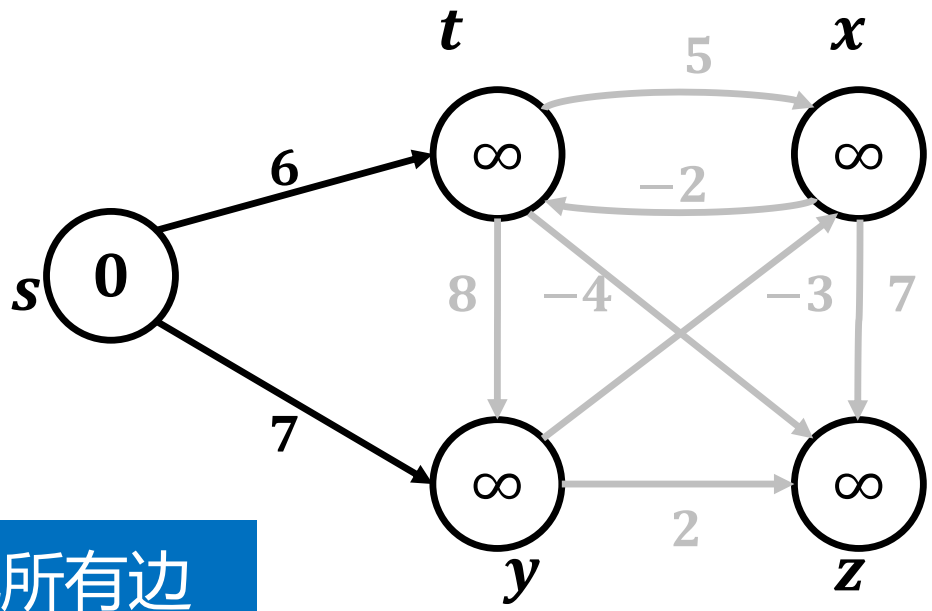
松弛失败

松弛成功

第1轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	N	N	N	N
$dist$	0	∞	∞	∞	∞



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

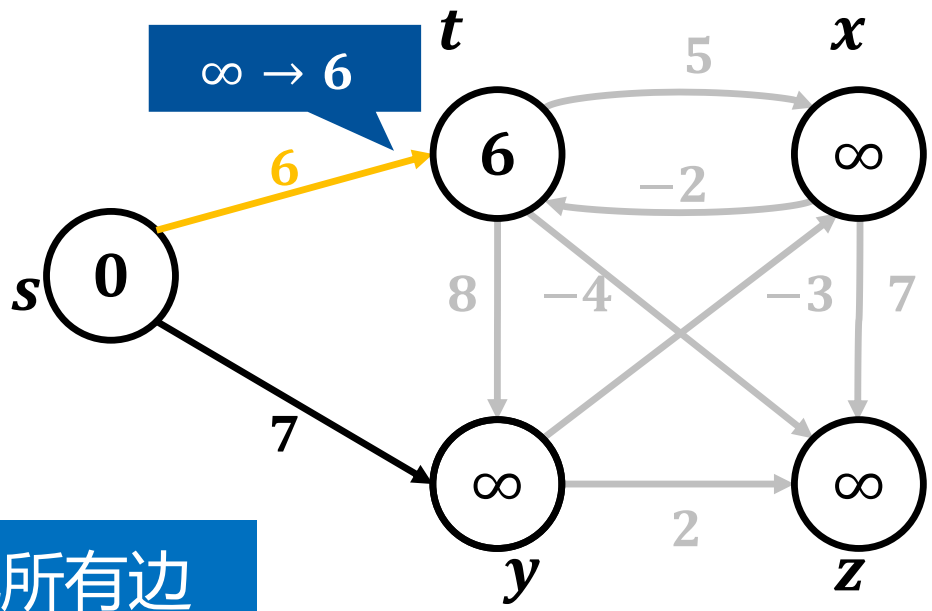
松弛失败

松弛成功

第1轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	s	N	N	N
$dist$	0	6	∞	∞	∞



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

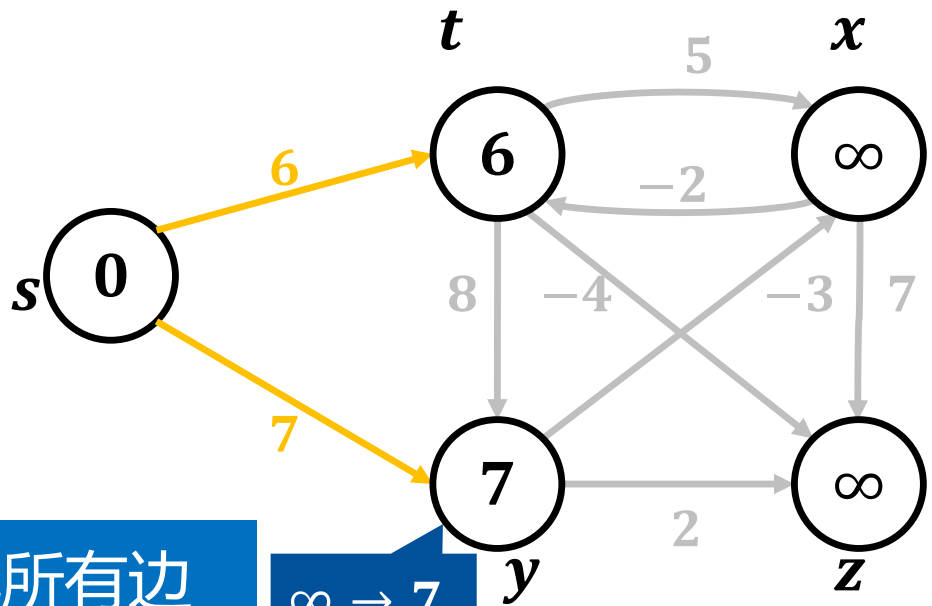
松弛失败

松弛成功

第1轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	s	N	s	N
$dist$	0	6	∞	7	∞



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

松弛失败

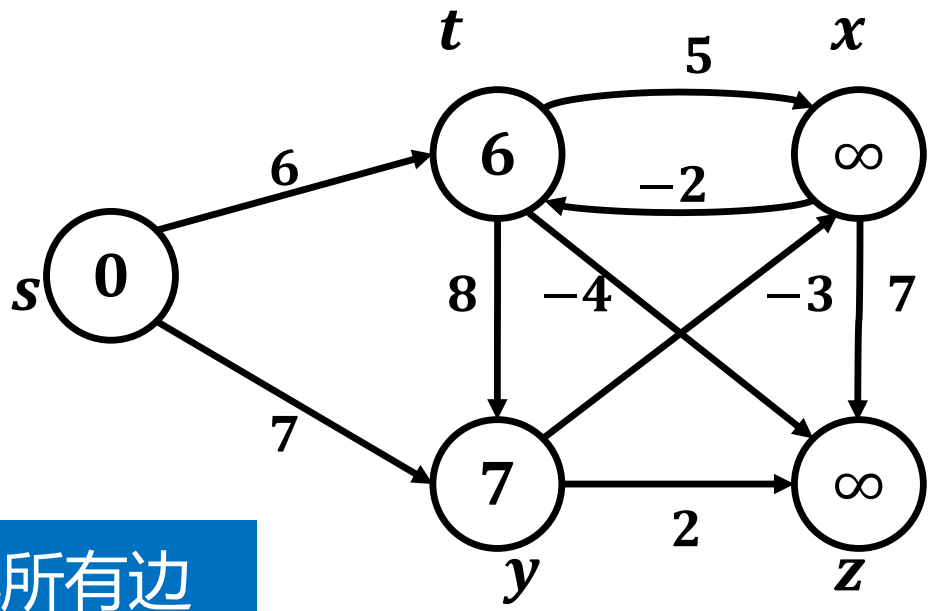
松弛成功

第1轮：松弛所有边

$\infty \rightarrow 7$

算法实例

V	s	t	x	y	z
$pred$	N	s	N	s	N
$dist$	0	6	∞	7	∞



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

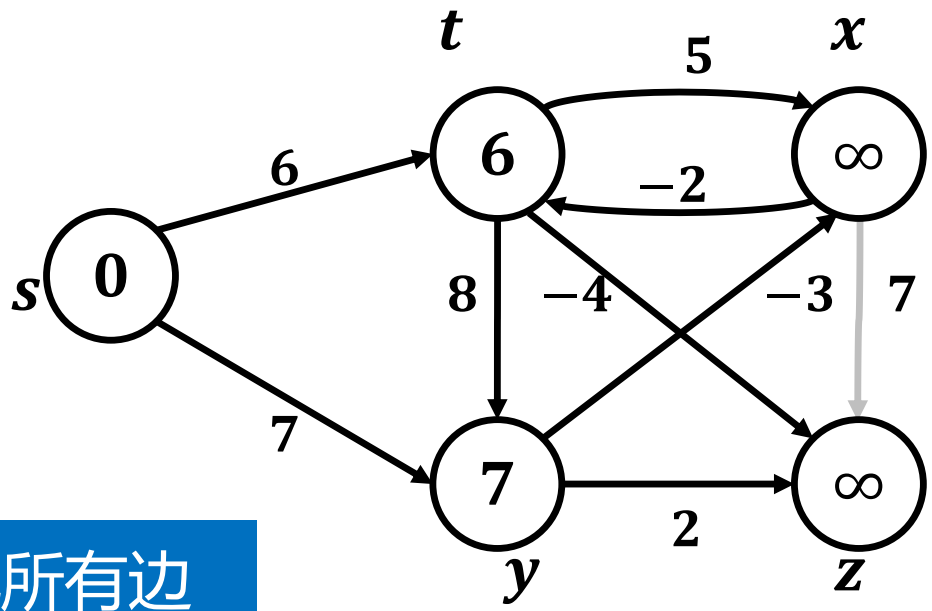
松弛失败

松弛成功

第2轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	s	N	s	N
$dist$	0	6	∞	7	∞



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

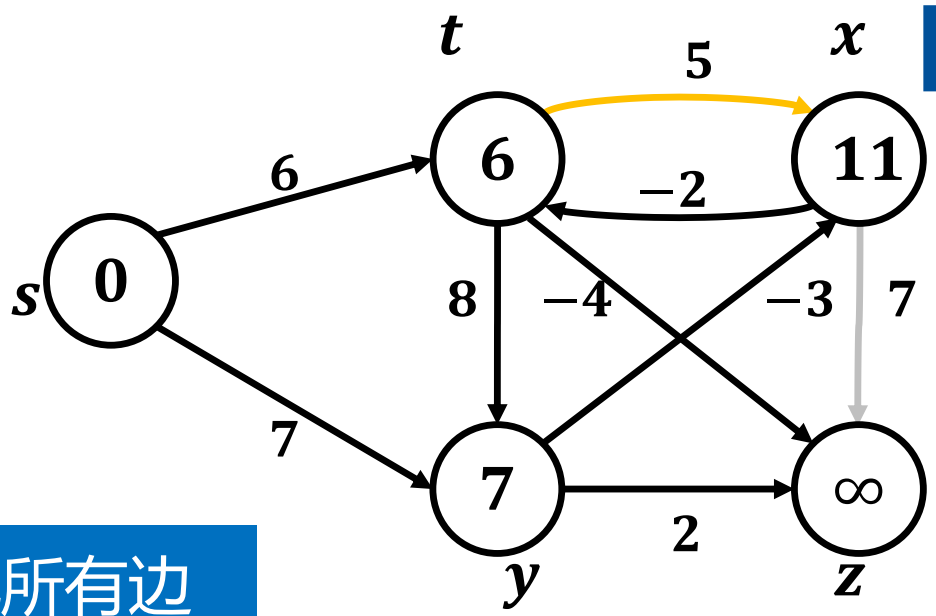
松弛失败

松弛成功

第2轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	s	t	s	N
$dist$	0	6	11	7	∞



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

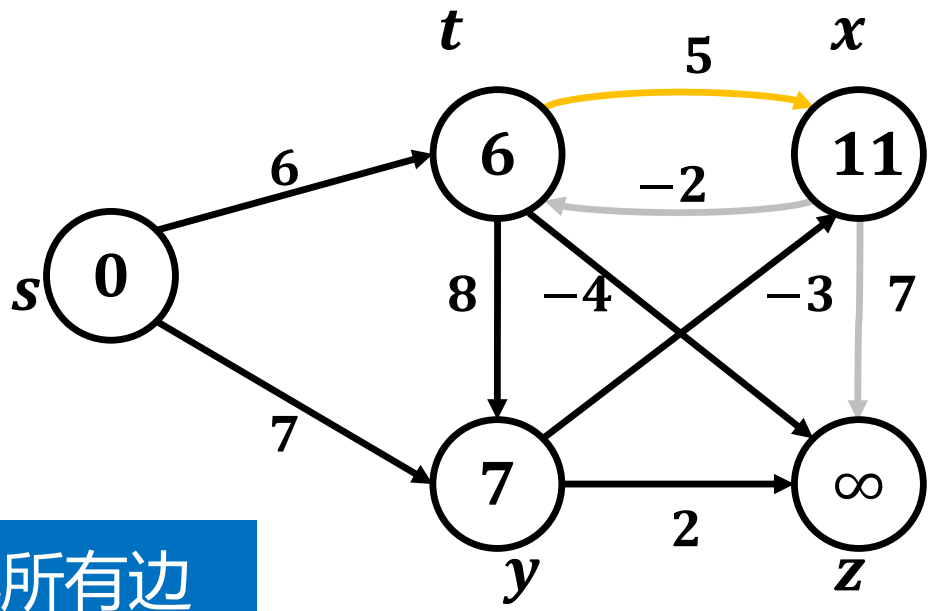
松弛失败

松弛成功

第2轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	s	t	s	N
$dist$	0	6	11	7	∞



松弛顺序：(x,z), (t,x), (x,t), (t,z), (y,x), (y,z), (t,y), (s,t), (s,y)

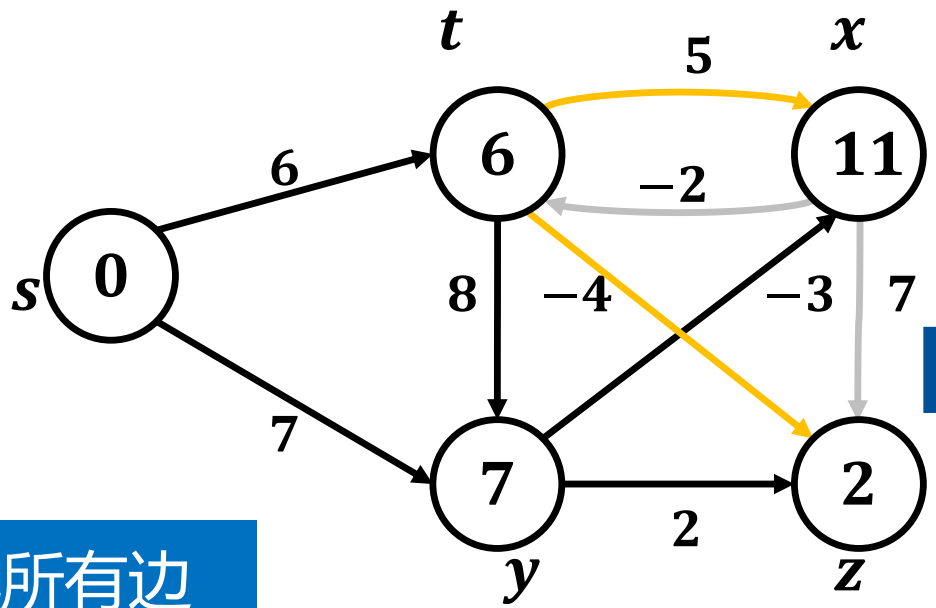
松弛失败

松弛成功

第2轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	s	t	s	t
$dist$	0	6	11	7	2



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

$\infty \rightarrow 2$

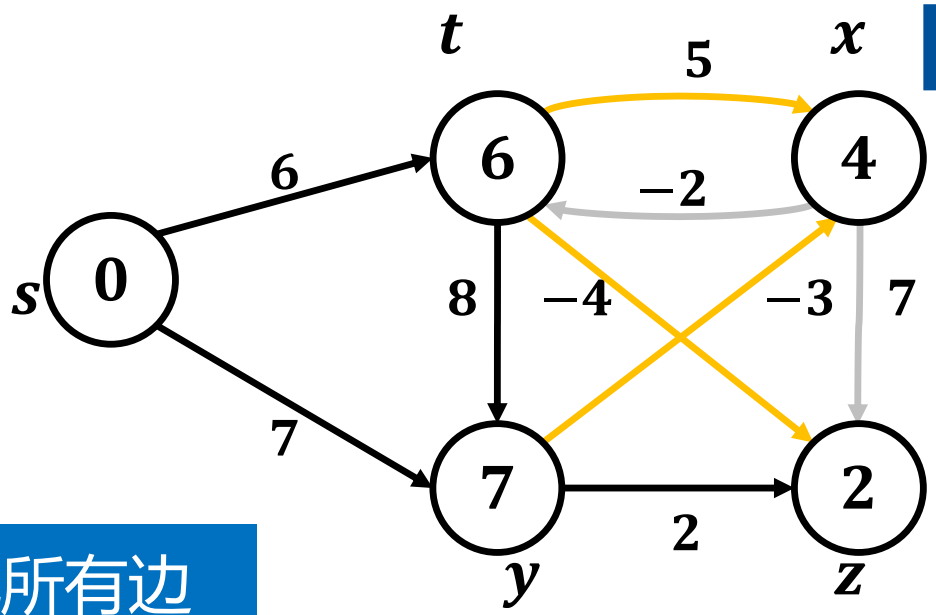
松弛失败

松弛成功

第2轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	s	y	s	t
$dist$	0	6	4	7	2



11 → 4

松弛顺序：(x,z), (t,x), (x,t), (t,z), (y,x), (y,z), (t,y), (s,t), (s,y)

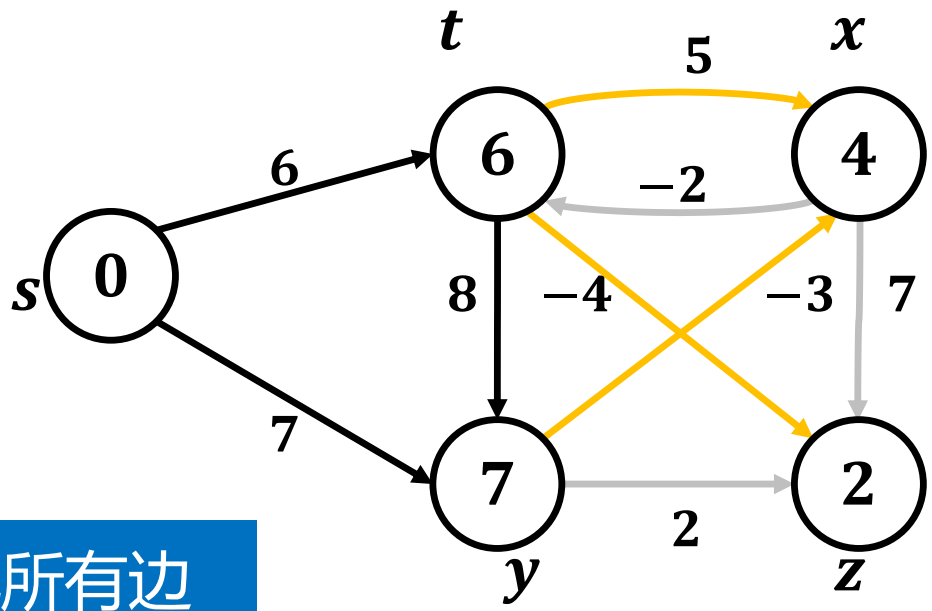
松弛失败

松弛成功

第2轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	s	y	s	t
$dist$	0	6	4	7	2



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

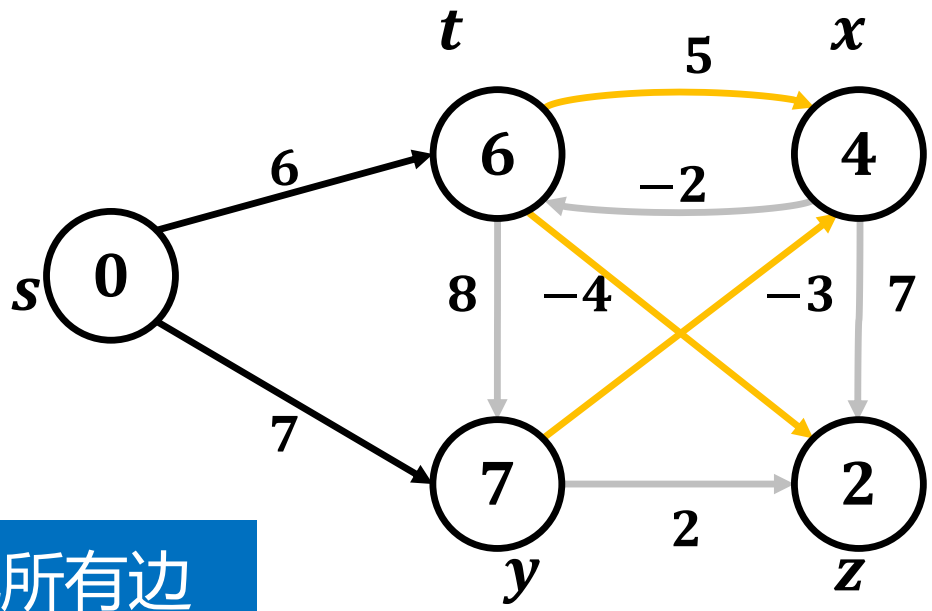
松弛失败

松弛成功

第2轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	s	y	s	t
$dist$	0	6	4	7	2



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

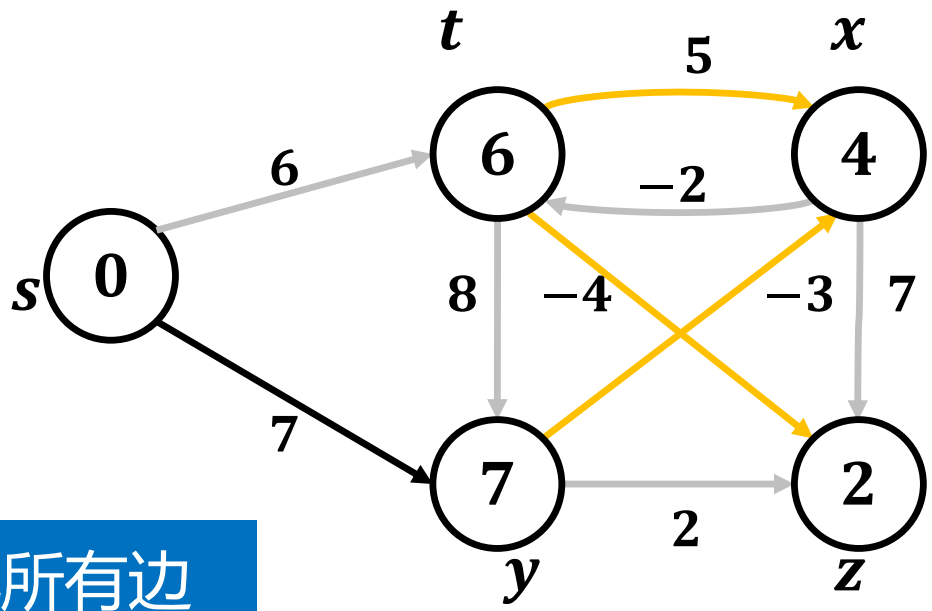
松弛失败

松弛成功

第2轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	s	y	s	t
$dist$	0	6	4	7	2



松弛顺序：(x,z), (t,x), (x,t), (t,z), (y,x), (y,z), (t,y), (s,t), (s,y)

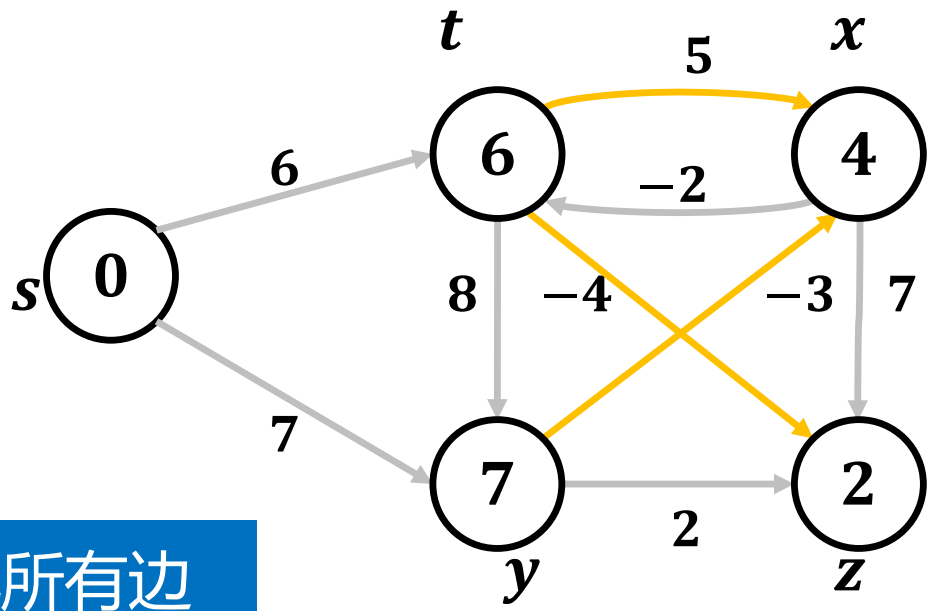
松弛失败

松弛成功

第2轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	s	y	s	t
$dist$	0	6	4	7	2



松弛顺序：(x,z), (t,x), (x,t), (t,z), (y,x), (y,z), (t,y), (s,t), (s,y)

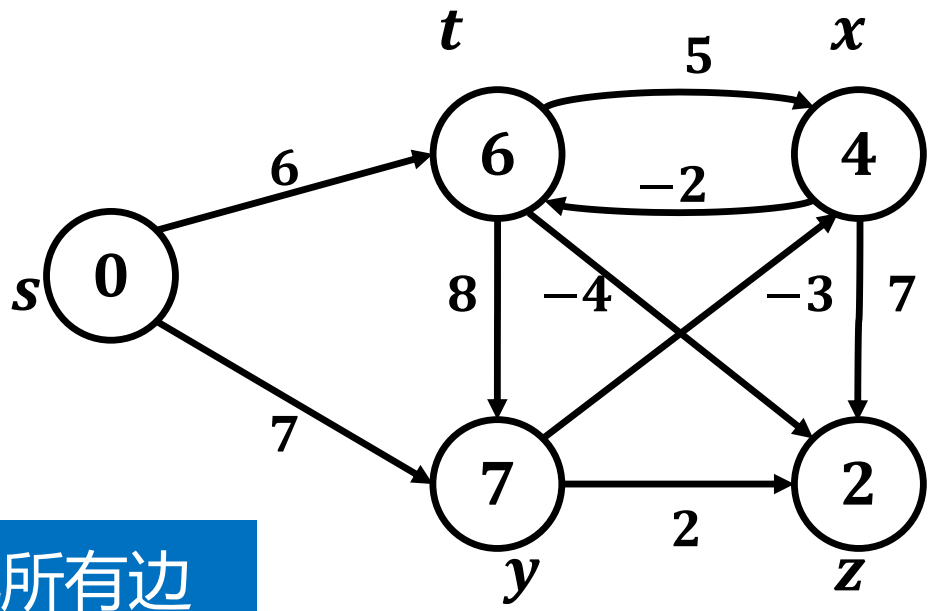
松弛失败

松弛成功

第2轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	s	y	s	t
$dist$	0	6	4	7	2



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

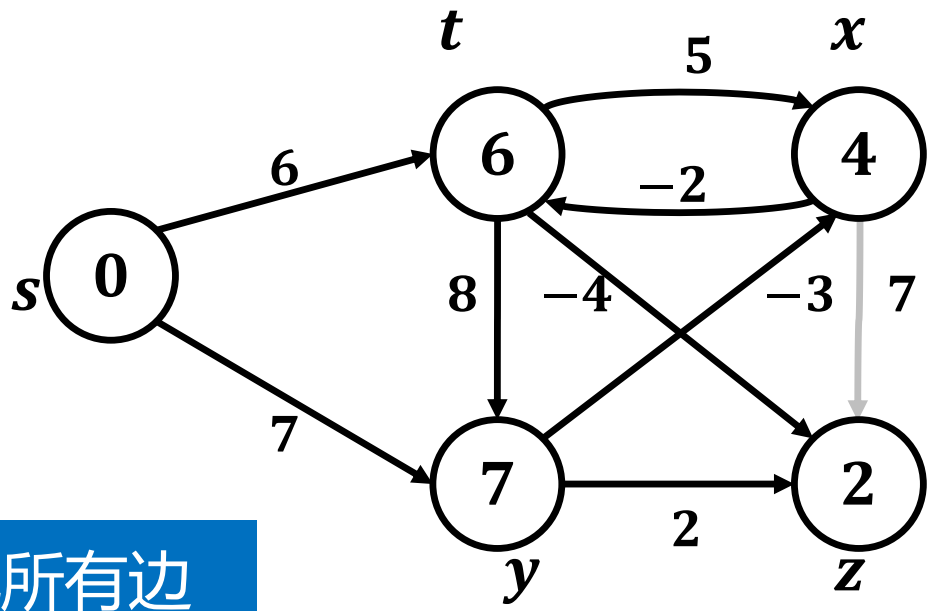
松弛失败

松弛成功

第3轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	s	y	s	t
$dist$	0	6	4	7	2



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

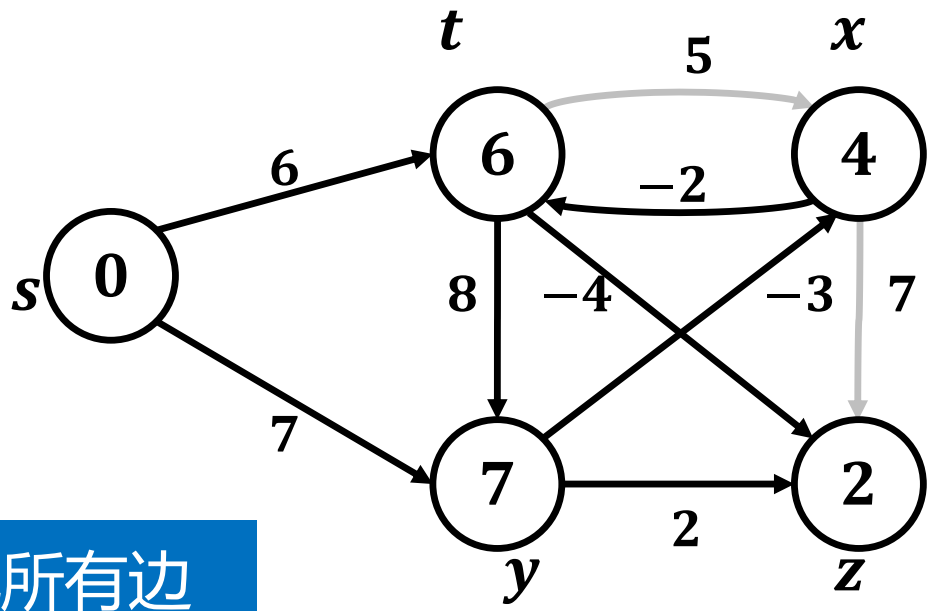
松弛失败

松弛成功

第3轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	s	y	s	t
$dist$	0	6	4	7	2



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

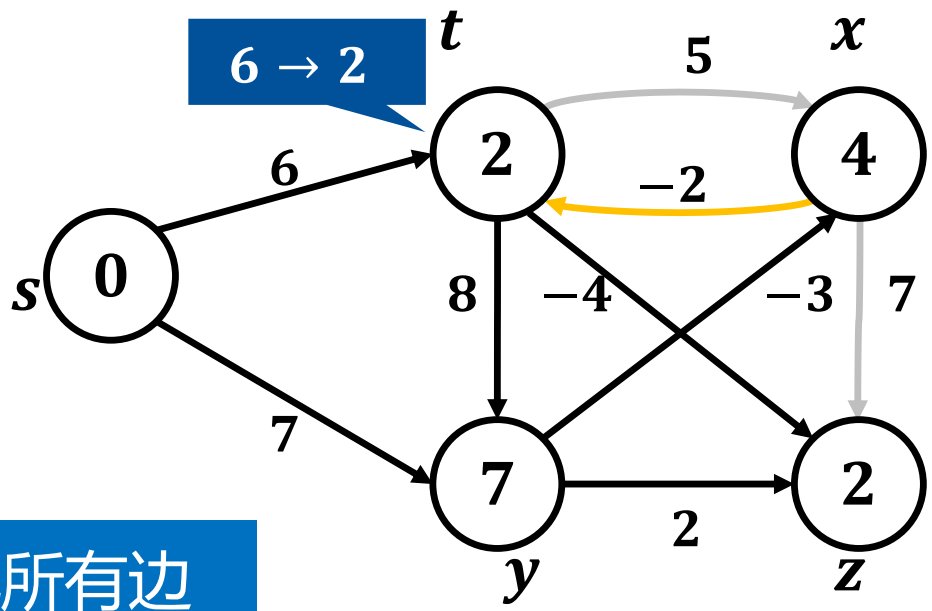
松弛失败

松弛成功

第3轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	x	y	s	t
$dist$	0	2	4	7	2



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

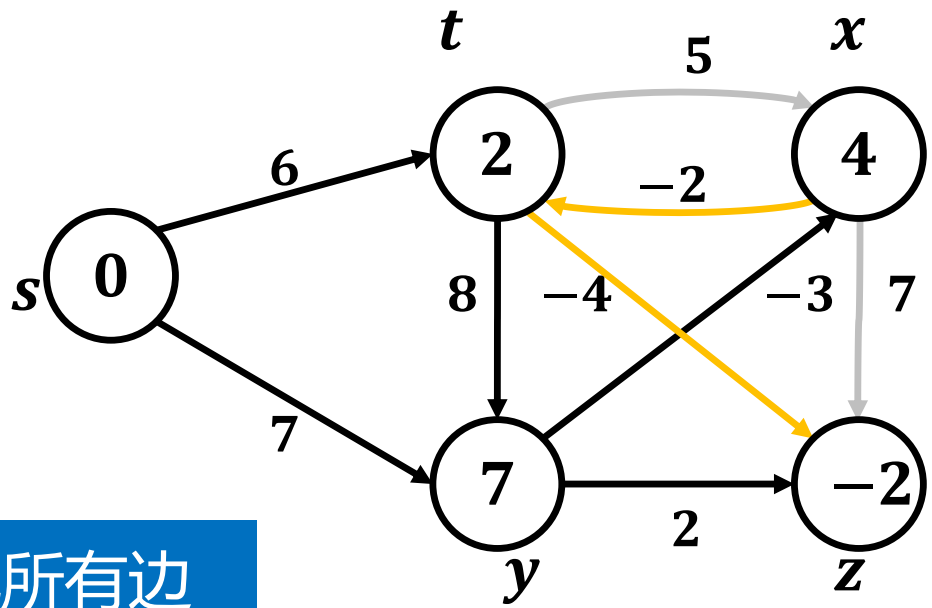
松弛失败

松弛成功

第3轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	x	y	s	t
$dist$	0	2	4	7	-2



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

松弛失败

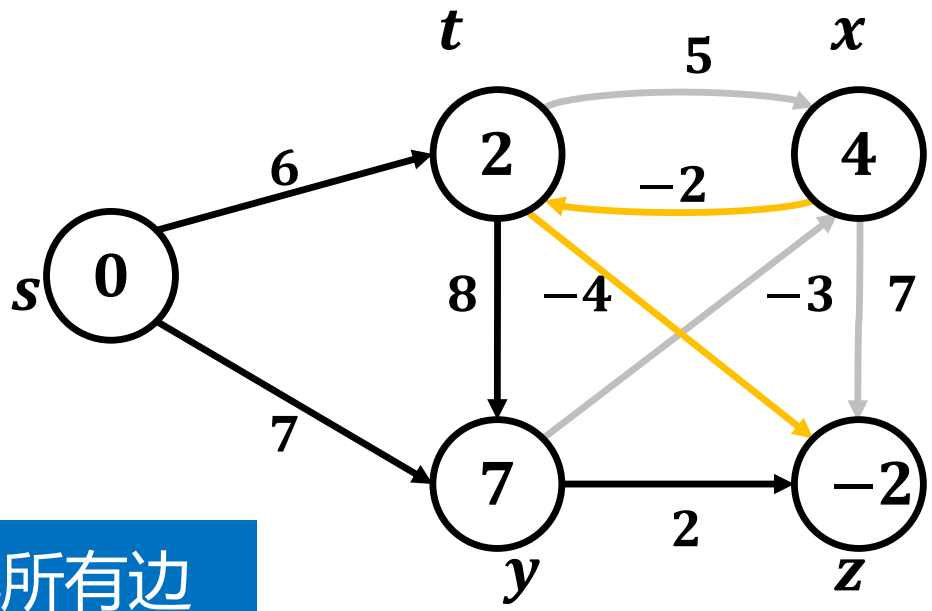
松弛成功

$2 \rightarrow -2$

第3轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	x	y	s	t
$dist$	0	2	4	7	-2



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

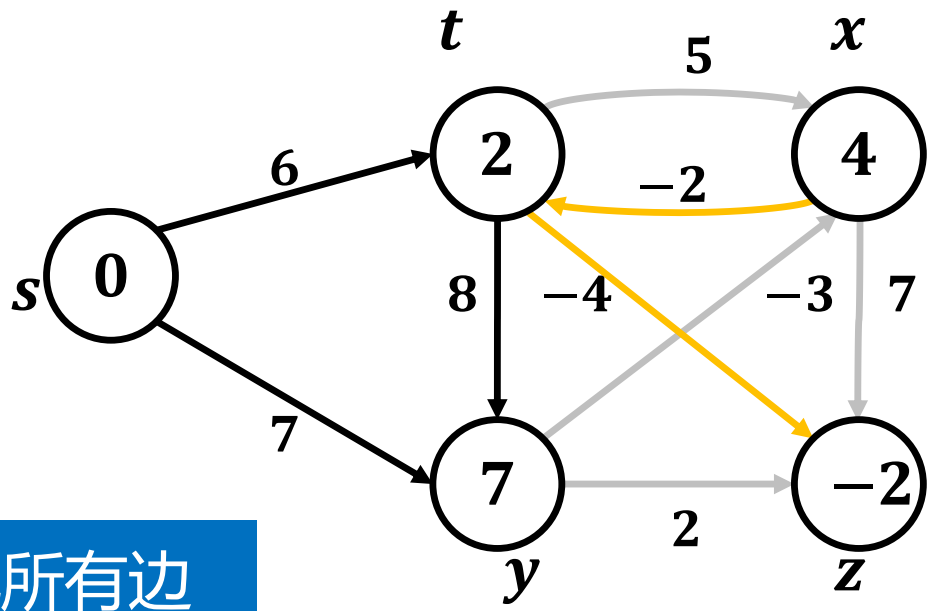
松弛失败

松弛成功

第3轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	x	y	s	t
$dist$	0	2	4	7	-2



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

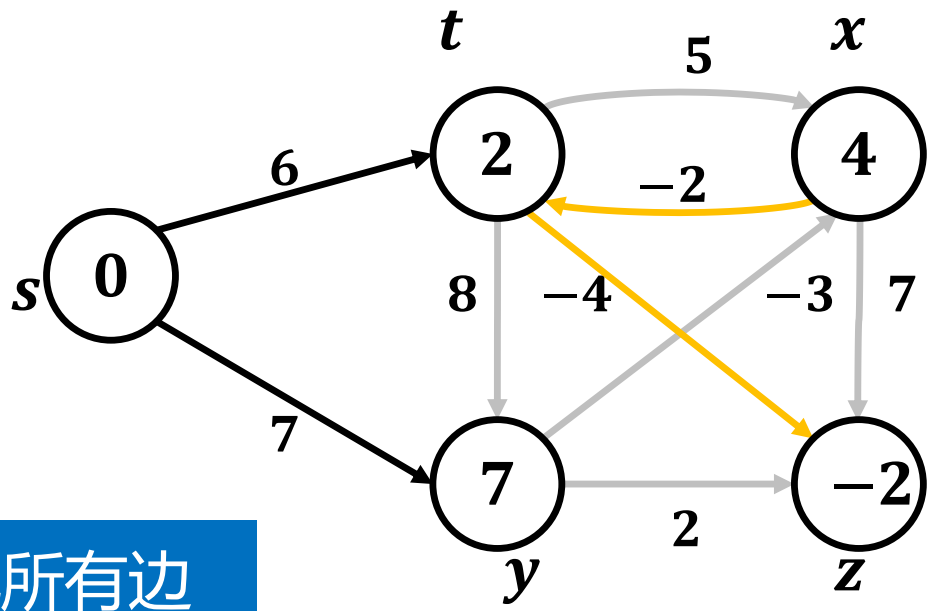
松弛失败

松弛成功

第3轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	x	y	s	t
$dist$	0	2	4	7	-2



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

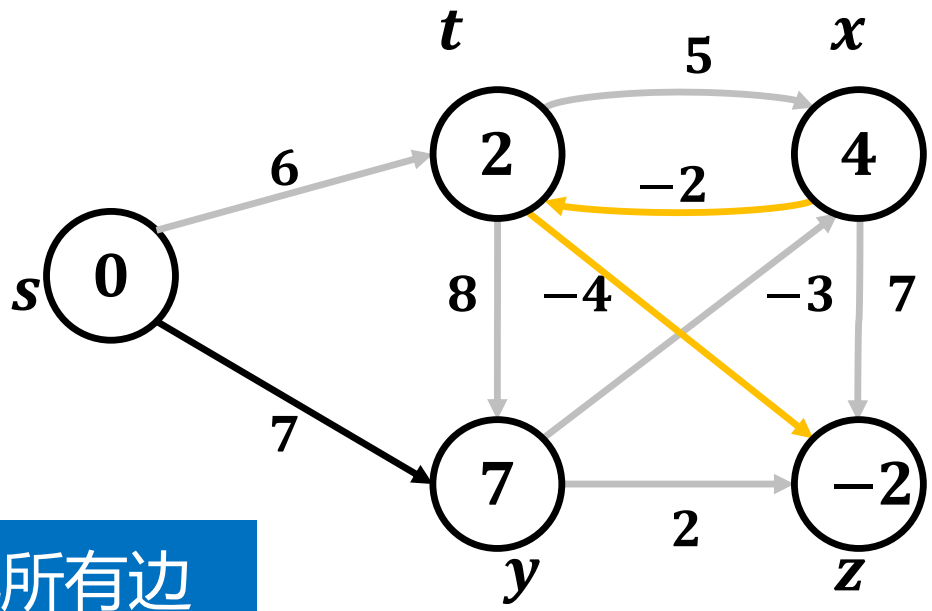
松弛失败

松弛成功

第3轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	x	y	s	t
$dist$	0	2	4	7	-2



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

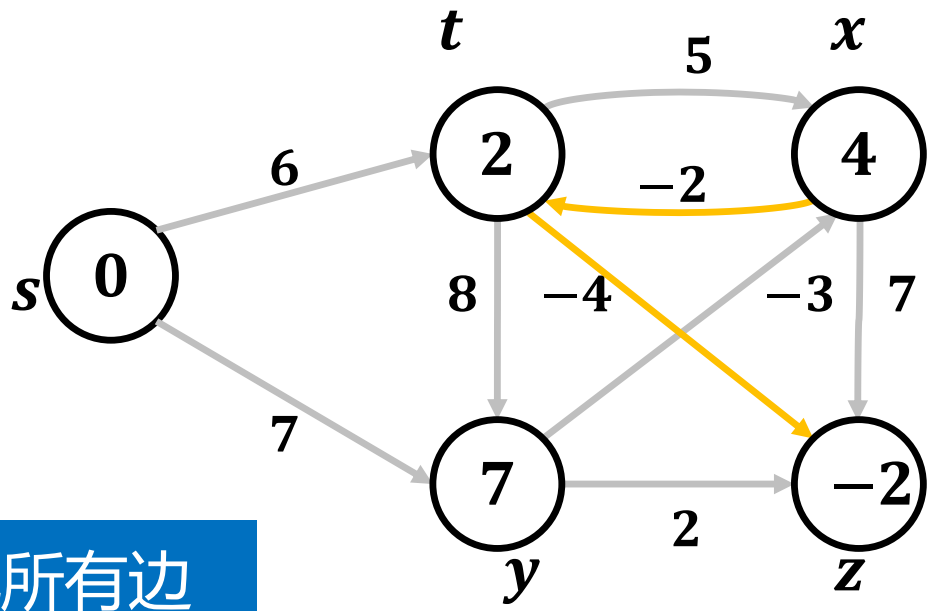
松弛失败

松弛成功

第3轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	x	y	s	t
$dist$	0	2	4	7	-2



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

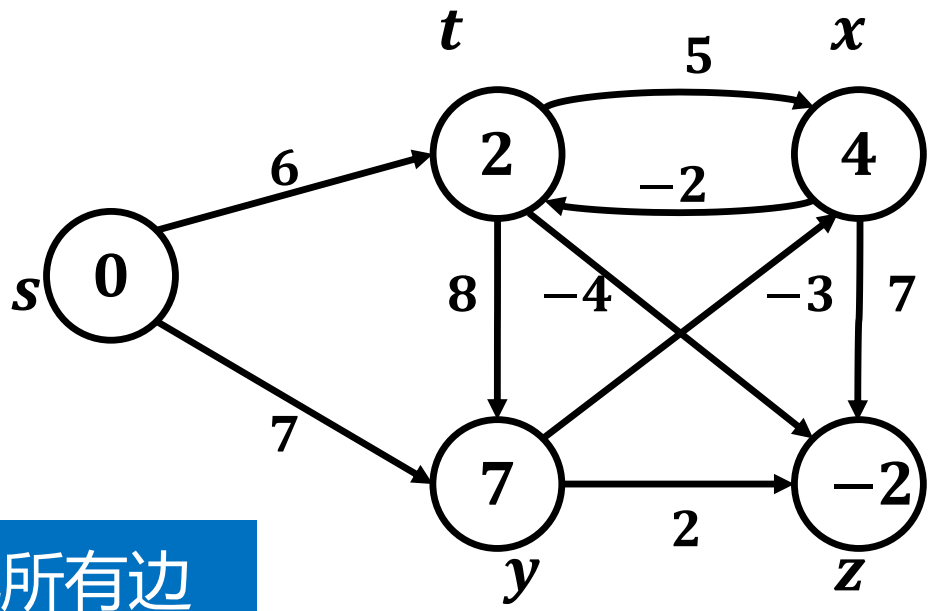
松弛失败

松弛成功

第3轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	x	y	s	t
$dist$	0	2	4	7	-2



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

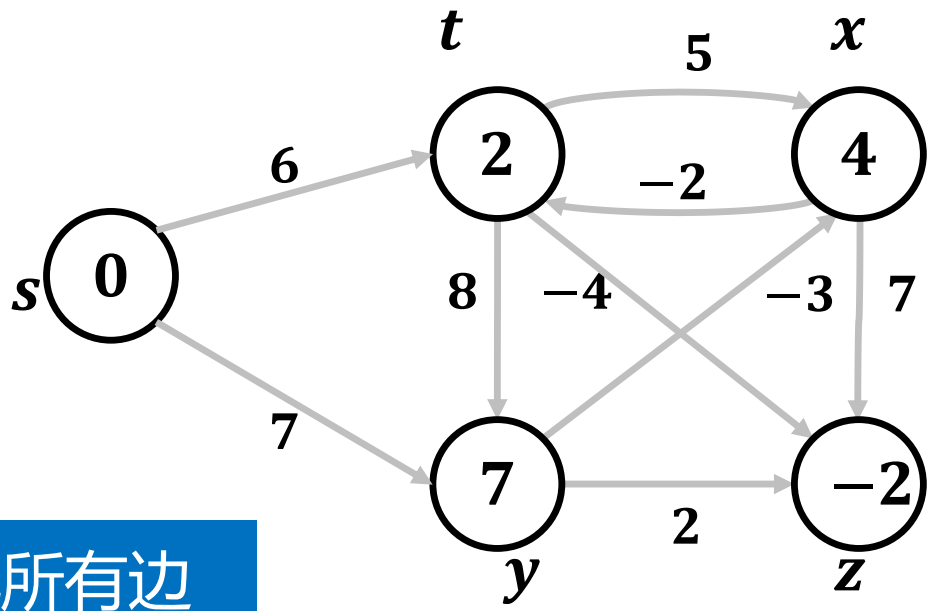
松弛失败

松弛成功

第4轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	x	y	s	t
$dist$	0	2	4	7	-2



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

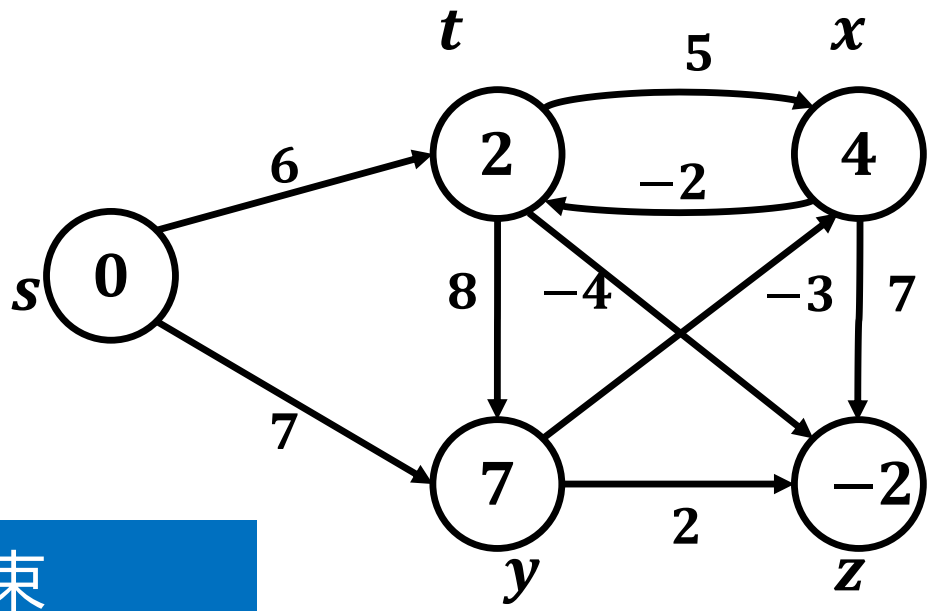
松弛失败

松弛成功

第4轮：松弛所有边

算法实例

V	s	t	x	y	z
$pred$	N	x	y	s	t
$dist$	0	2	4	7	-2



松弛顺序：$(x,z),(t,x),(x,t),(t,z),(y,x),(y,z),(t,y),(s,t),(s,y)$

松弛失败

松弛成功

松弛结束

提纲

- 问题背景
- 算法思想
- 算法实例
- 算法分析
- 算法性质

伪代码

- **Bellman-Ford(G, s)**

输入: 图$G =< V, E, W >$, 源点s
输出: 单源最短路径P
新建一维数组$dist[1..|V|]$, $pred[1..|V|]$
//初始化
for $u \in V$ **do**
  $dist[u] \leftarrow \infty$
  $pred[u] \leftarrow NULL$
end
$dist[s] \leftarrow 0$

初始化辅助数组

伪代码

- **Bellman-Ford(G, s)**

输入: 图$G =< V, E, W >$, 源点s
输出: 单源最短路径P
新建一维数组$dist[1..|V|]$, $pred[1..|V|]$
//初始化
for $u \in V$ do
  $dist[u] \leftarrow \infty$
  $pred[u] \leftarrow NULL$
end
$dist[s] \leftarrow 0$

初始化源点距离

伪代码

- **Bellman-Ford(G, s)**

```
//执行单源最短路径算法
for i ← 1 to |V| − 1 do
    for (u, v) ∈ E do
        if dist[u] + w(u, v) < dist[v] then
            dist[v] ← dist[u] + w(u, v)
            pred[v] ← u
        end
    end
end
for (u, v) ∈ E do
    if dist[u] + w(u, v) < dist[v] then
        print 存在负环
        break
    end
end
```

进行$|V| - 1$轮松弛

伪代码

- **Bellman-Ford(G, s)**

```
//执行单源最短路径算法
for i ← 1 to |V| − 1 do
    for (u, v) ∈ E do
        if dist[u] + w(u, v) < dist[v] then
            dist[v] ← dist[u] + w(u, v)
            pred[v] ← u
        end
    end
end
for (u, v) ∈ E do
    if dist[u] + w(u, v) < dist[v] then
        print 存在负环
        break
    end
end
```

对所有边进行松弛操作

伪代码

- **Bellman-Ford(G, s)**

```
//执行单源最短路径算法
for i ← 1 to |V| − 1 do
    for (u, v) ∈ E do
        if dist[u] + w(u, v) < dist[v] then
            dist[v] ← dist[u] + w(u, v)
            pred[v] ← u
        end
    end
end
for (u, v) ∈ E do
    if dist[u] + w(u, v) < dist[v] then
        print 存在负环
        break
    end
end
```

更新辅助数组

伪代码

- **Bellman-Ford(G, s)**

```
//执行单源最短路径算法
for i ← 1 to |V| − 1 do
    for (u, v) ∈ E do
        if dist[u] + w(u, v) < dist[v] then
            dist[v] ← dist[u] + w(u, v)
            pred[v] ← u
        end
    end
end
for (u, v) ∈ E do
    if dist[u] + w(u, v) < dist[v] then
        print 存在负环
        break
    end
end
```

判断是否存在负环

时间复杂度分析

- **Bellman-Ford(G, s)**

输入: 图$G =< V, E, W >$, 源点s
输出: 单源最短路径P
新建一维数组$dist[1..|V|]$, $pred[1..|V|]$
//初始化
for $u \in V$ do
  $dist[u] \leftarrow \infty$
  $pred[u] \leftarrow NULL$
end
$dist[s] \leftarrow 0$

$O(|V|)$

时间复杂度分析

- **Bellman-Ford(G, s)**

```
//执行单源最短路径算法
for i ← 1 to |V| − 1 do
    for (u, v) ∈ E do
        if dist[u] + w(u, v) < dist[v] then
            dist[v] ← dist[u] + w(u, v)
            pred[v] ← u
        end
    end
end
for (u, v) ∈ E do
    if dist[u] + w(u, v) < dist[v] then
        print 存在负环
        break
    end
end
```

时间复杂度分析

- **Bellman-Ford(G, s)**

```
//执行单源最短路径算法
for i ← 1 to |V| − 1 do
    for (u, v) ∈ E do
        if dist[u] + w(u, v) < dist[v] then
            dist[v] ← dist[u] + w(u, v)
            pred[v] ← u
        end
    end
end
for (u, v) ∈ E do
    if dist[u] + w(u, v) < dist[v] then
        print 存在负环
        break
    end
end
```

$O(|E|)$

$O(|E| \cdot |V|)$

$O(|E|)$

时间复杂度分析

- **Bellman-Ford(G, s)**

```
//执行单源最短路径算法
for i ← 1 to |V| − 1 do
    for (u, v) ∈ E do
        if dist[u] + w(u, v) < dist[v] then
            dist[v] ← dist[u] + w(u, v)
            pred[v] ← u
        end
    end
end
for (u, v) ∈ E do
    if dist[u] + w(u, v) < dist[v] then
        print 存在负环
        break
    end
end
```

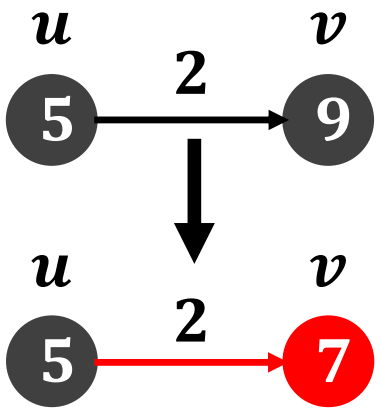
时间复杂度$O(|E| \cdot |V|)$

提纲

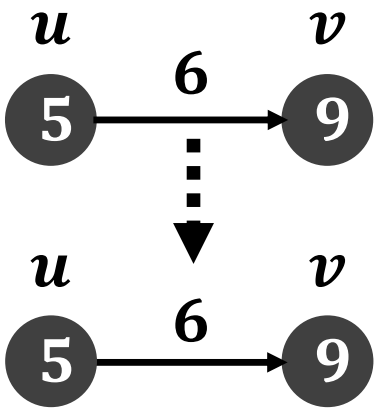
- 问题背景
- 算法思想
- 算法实例
- 算法分析
- 算法性质

算法思想

- 图中存在负权边时：每轮对所有边进行松弛，持续迭代 $|V|-1$轮
- 图中存在负权边时：若第 $|V|$ 轮仍松弛成功，存在源点s可达的负环



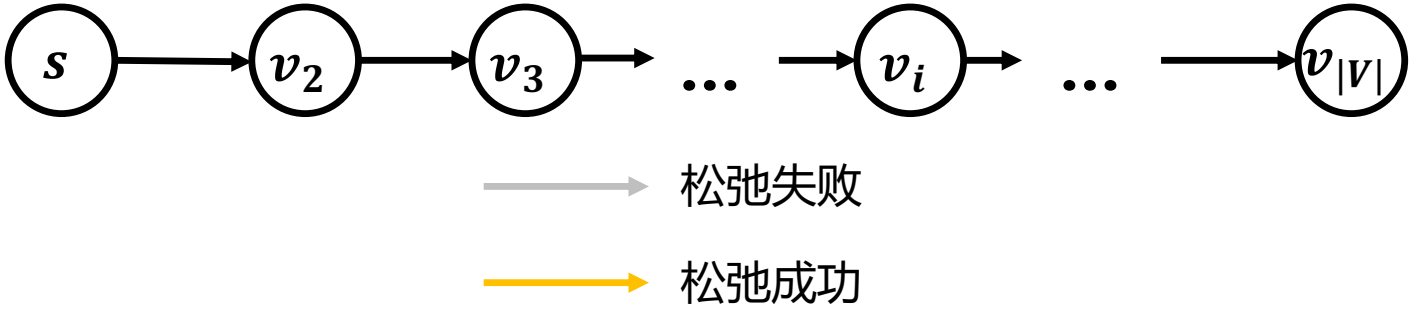
松弛成功



松弛失败

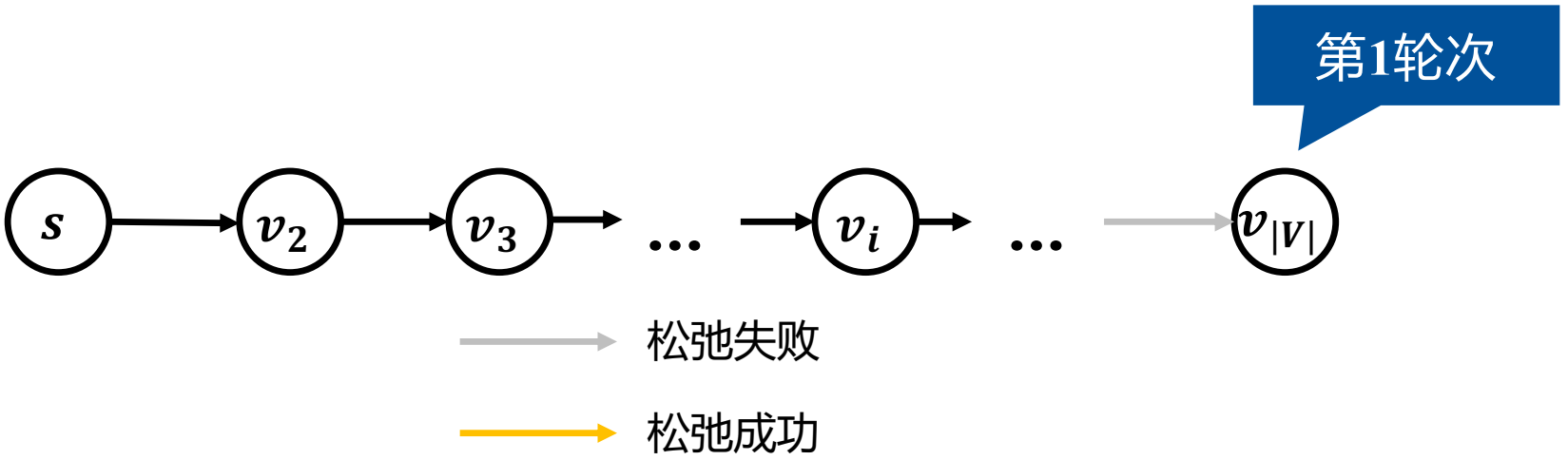
算法正确性

- 图中存在负权边时：每轮对所有边进行松弛，持续迭代 $|V|-1$轮
- 最坏情况
  - 非环路的路径$< s, v_2, v_3, \dots, v_{|V|} >$至多经过$|V|-1$条边



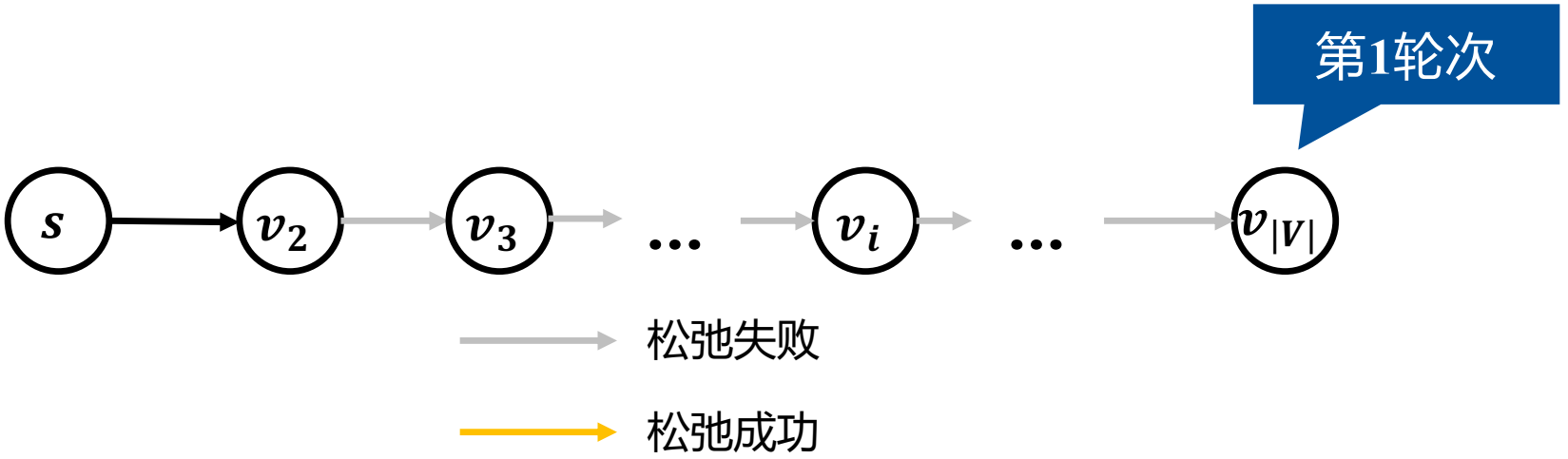
算法正确性

- 图中存在负权边时：每轮对所有边进行松弛，持续迭代 $|V|-1$轮
- 最坏情况
  - 非环路的路径$< s, v_2, v_3, \dots, v_{|V|} >$至多经过$|V|-1$条边



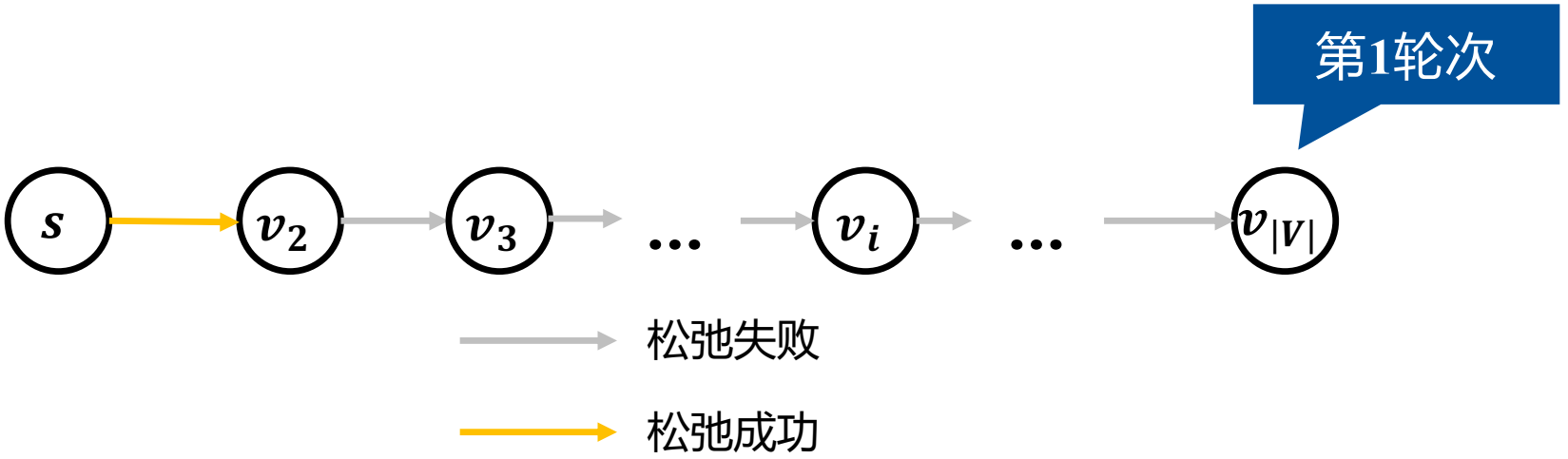
算法正确性

- 图中存在负权边时：每轮对所有边进行松弛，持续迭代 $|V|-1$轮
- 最坏情况
  - 非环路的路径$< s, v_2, v_3, \dots, v_{|V|} >$至多经过$|V|-1$条边



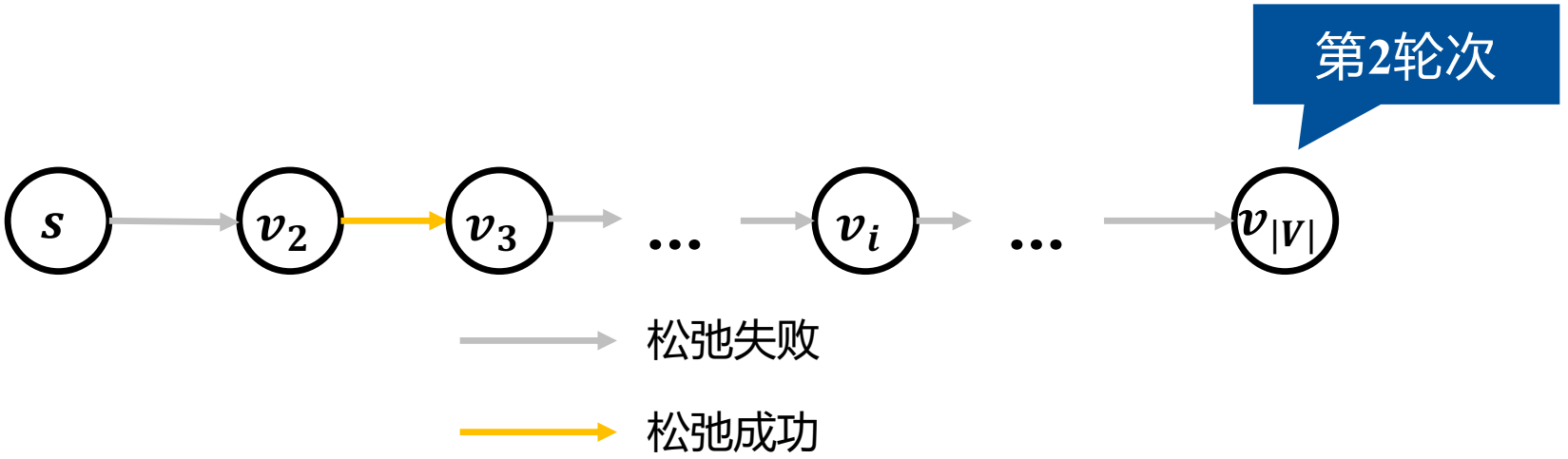
算法正确性

- 图中存在负权边时：每轮对所有边进行松弛，持续迭代 $|V|-1$轮
- 最坏情况
  - 非环路的路径$< s, v_2, v_3, \dots, v_{|V|} >$至多经过$|V|-1$条边



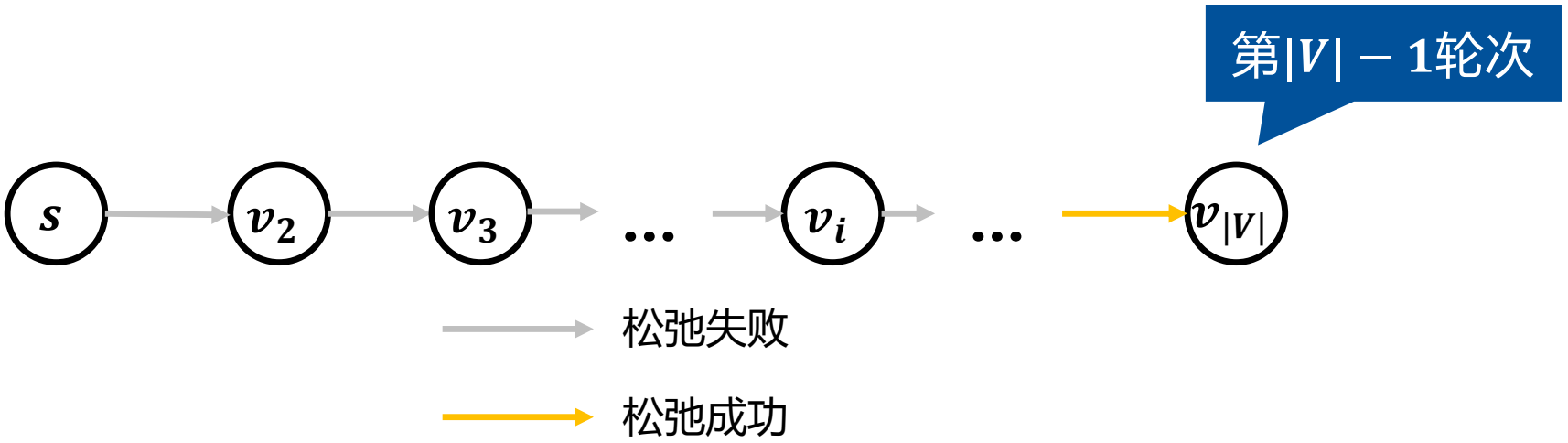
算法正确性

- 图中存在负权边时：每轮对所有边进行松弛，持续迭代 $|V|-1$轮
- 最坏情况
  - 非环路的路径$< s, v_2, v_3, \dots, v_{|V|} >$至多经过$|V|-1$条边



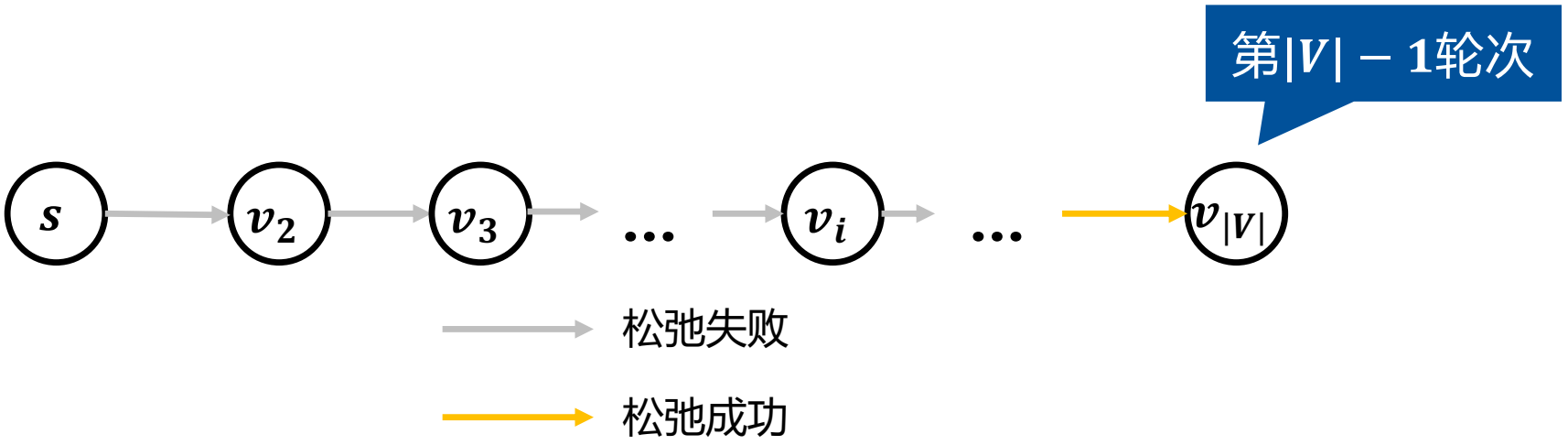
算法正确性

- 图中存在负权边时：每轮对所有边进行松弛，持续迭代 $|V|-1$轮
- 最坏情况
  - 非环路的路径$< s, v_2, v_3, \dots, v_{|V|} >$至多经过$|V|-1$条边



算法正确性

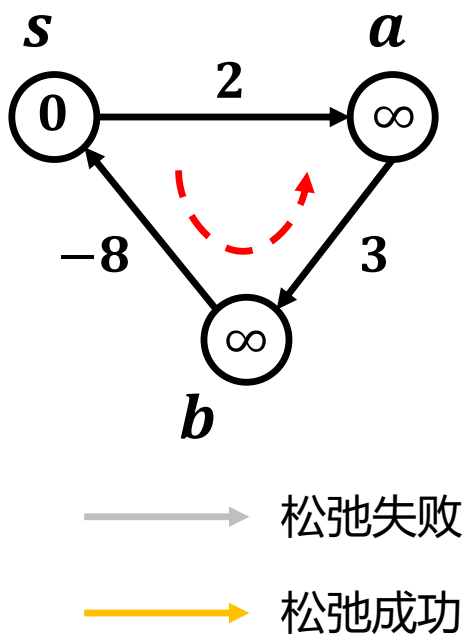
- 图中存在负权边时：每轮对所有边进行松弛，持续迭代 $|V|-1$轮
- 最坏情况
  - 非环路的路径$< s, v_2, v_3, \dots, v_{|V|} >$至多经过$|V|-1$条边



最坏情况下进行 $|V|-1$ 轮松弛操作，可以保证求得单源最短路径

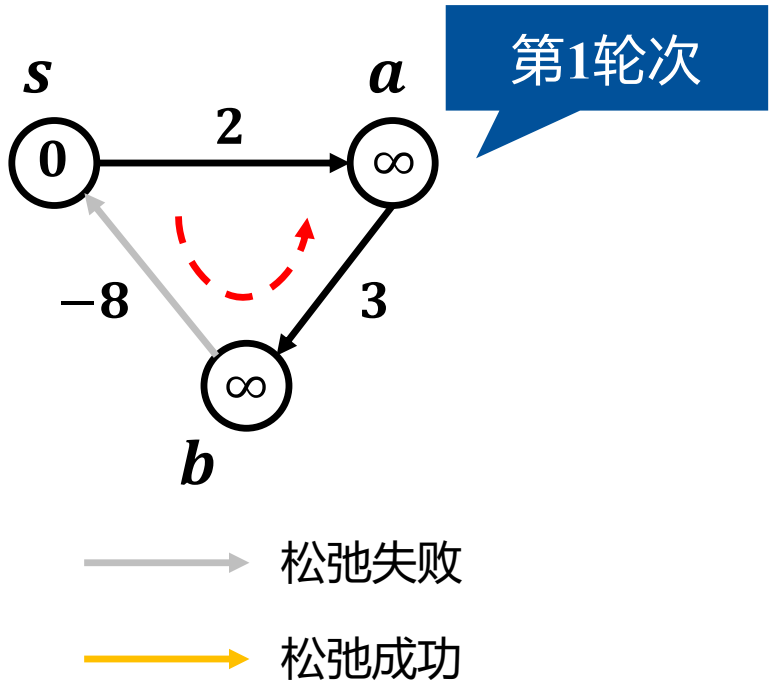
算法正确性

- 图中存在负权边时：若第 $|V|$ 轮仍松弛成功，存在源点s可达的负环
- 若源点s可达负环，可松弛成功无限次



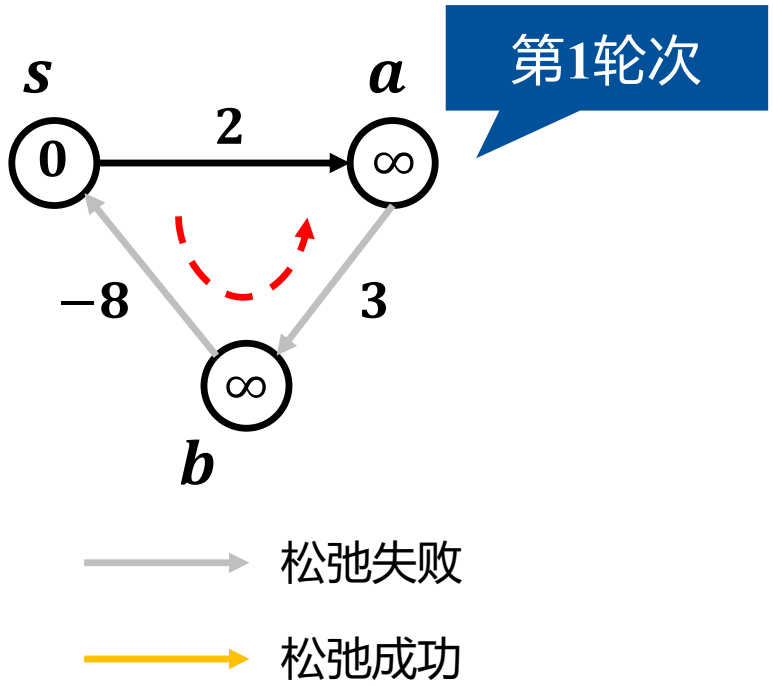
算法正确性

- 图中存在负权边时：若第 $|V|$ 轮仍松弛成功，存在源点s可达的负环
- 若源点s可达负环，可松弛成功无限次



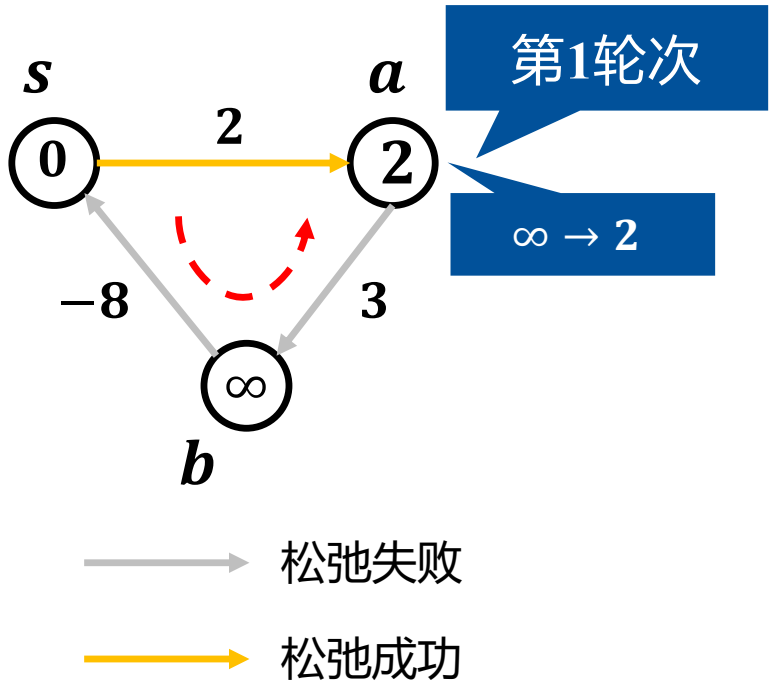
算法正确性

- 图中存在负权边时：若第 $|V|$ 轮仍松弛成功，存在源点s可达的负环
- 若源点s可达负环，可松弛成功无限次



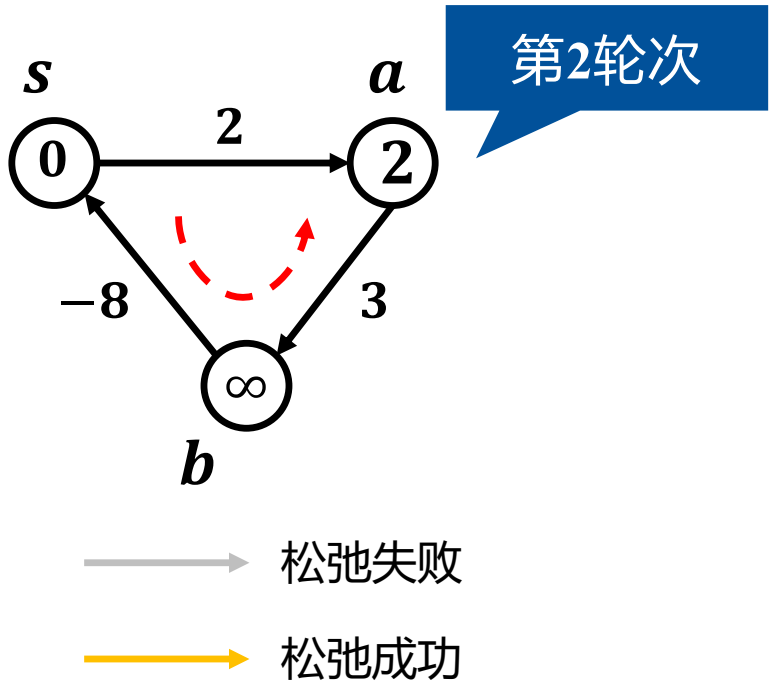
算法正确性

- 图中存在负权边时：若第 $|V|$ 轮仍松弛成功，存在源点s可达的负环
- 若源点s可达负环，可松弛成功无限次



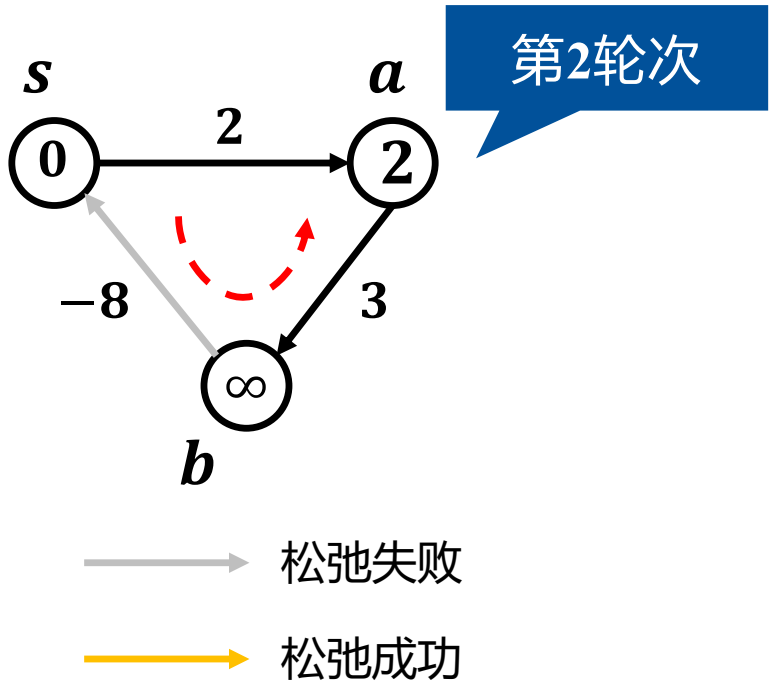
算法正确性

- 图中存在负权边时：若第 $|V|$ 轮仍松弛成功，存在源点s可达的负环
- 若源点s可达负环，可松弛成功无限次



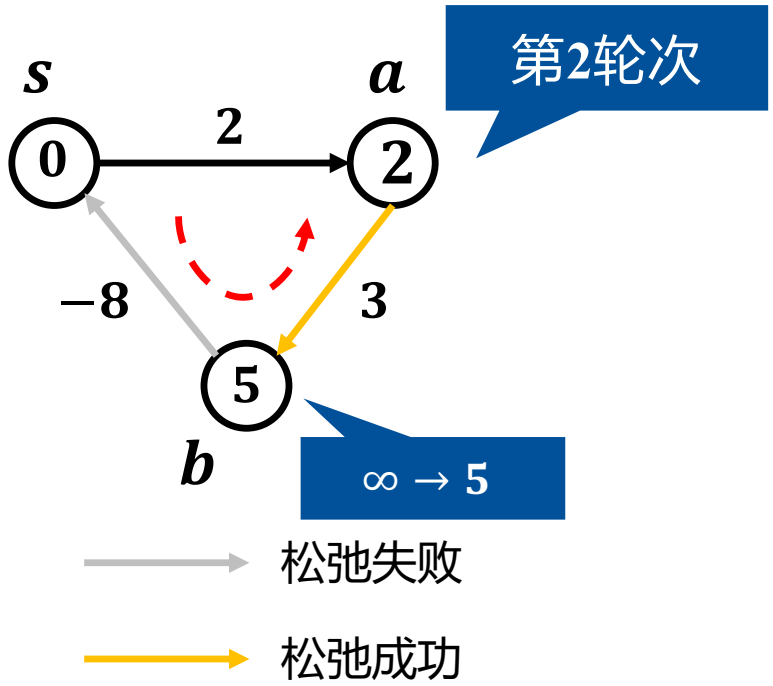
算法正确性

- 图中存在负权边时：若第 $|V|$ 轮仍松弛成功，存在源点s可达的负环
- 若源点s可达负环，可松弛成功无限次



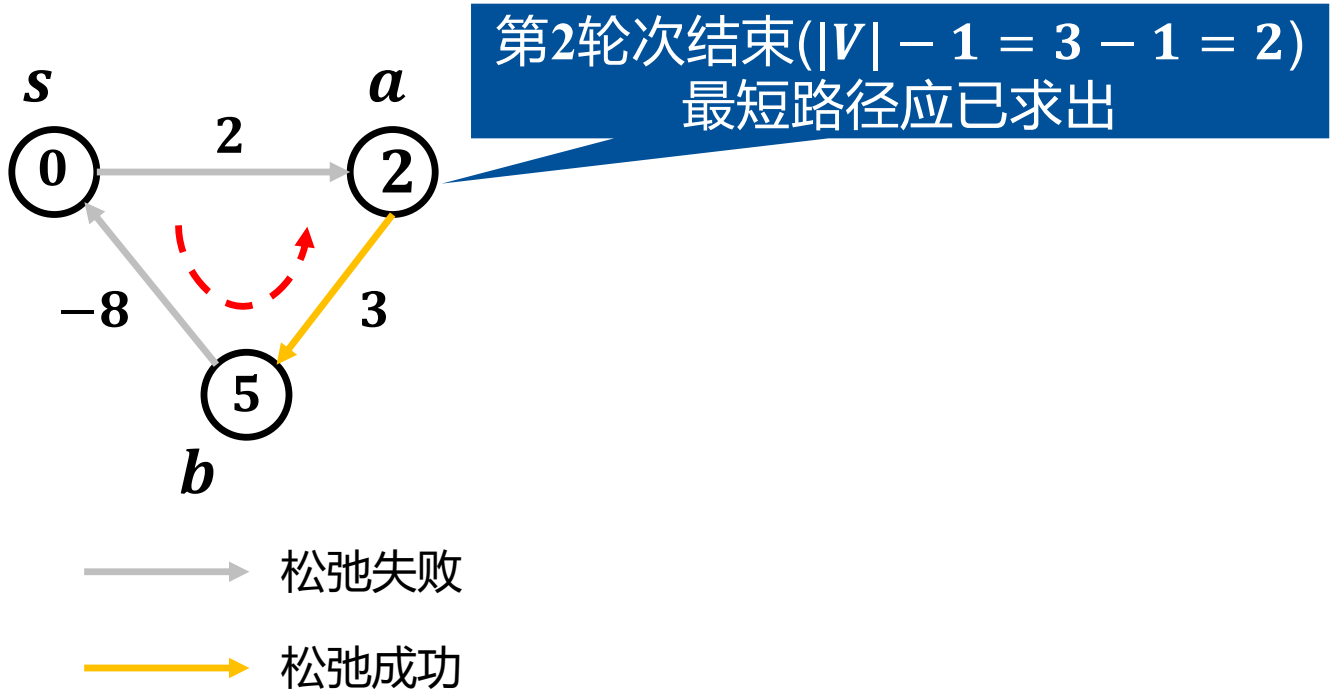
算法正确性

- 图中存在负权边时：若第 $|V|$ 轮仍松弛成功，存在源点s可达的负环
- 若源点s可达负环，可松弛成功无限次



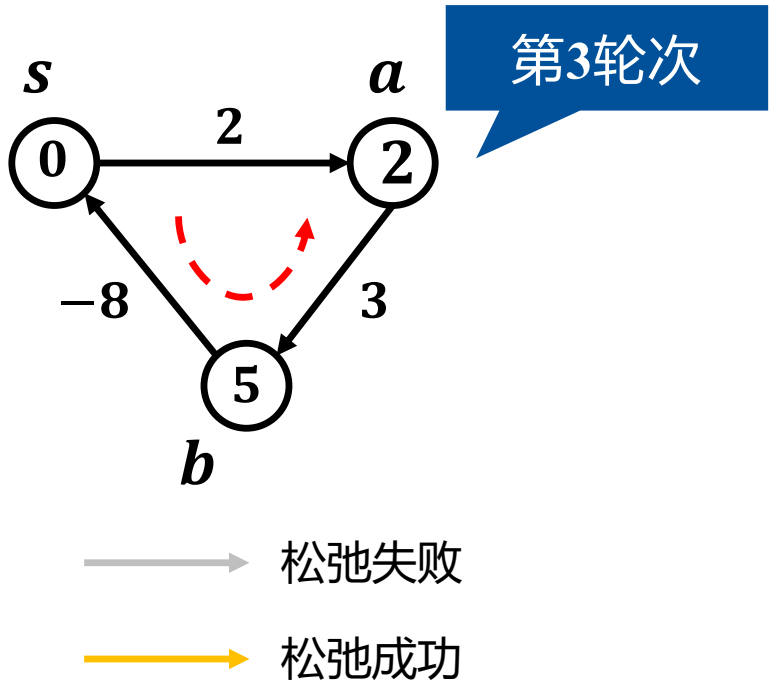
算法正确性

- 图中存在负权边时：若第 $|V|$ 轮仍松弛成功，存在源点s可达的负环
- 若源点s可达负环，可松弛成功无限次



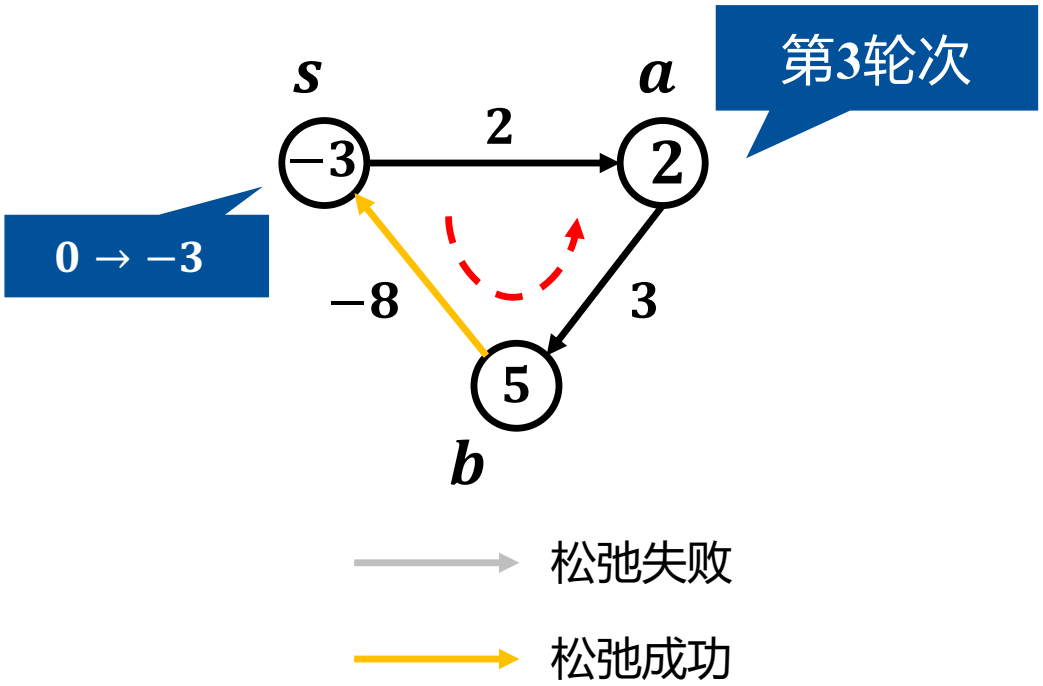
算法正确性

- 图中存在负权边时：若第 $|V|$ 轮仍松弛成功，存在源点s可达的负环
- 若源点s可达负环，可松弛成功无限次



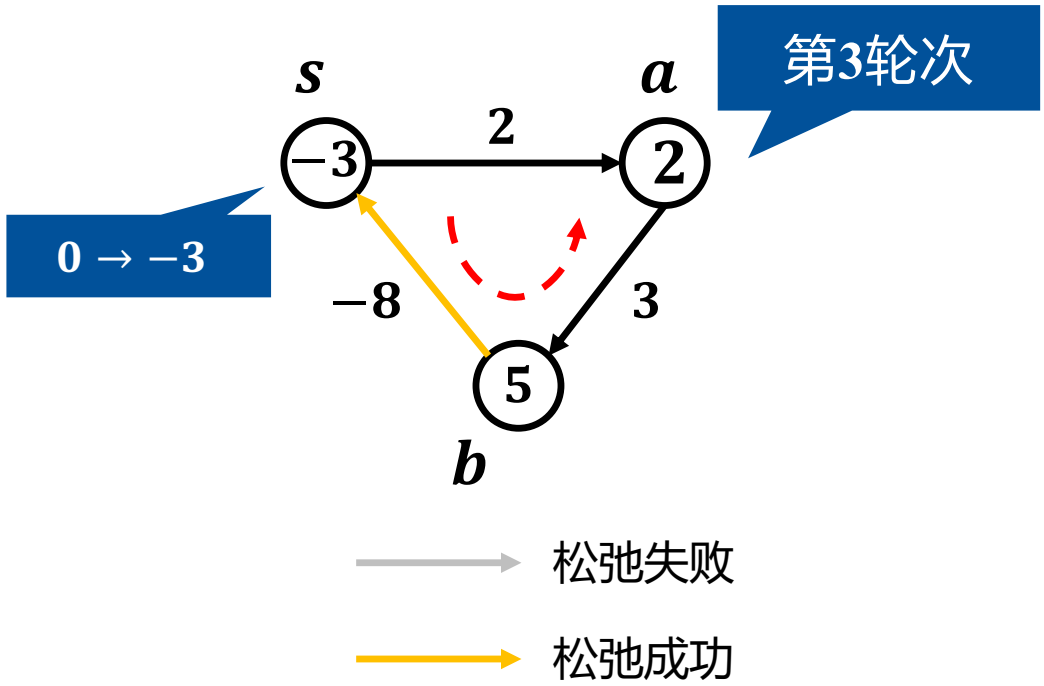
算法正确性

- 图中存在负权边时：若第 $|V|$ 轮仍松弛成功，存在源点s可达的负环
- 若源点s可达负环，可松弛成功无限次



算法正确性

- 图中存在负权边时：若第 $|V|$ 轮仍松弛成功，存在源点s可达的负环
- 若源点s可达负环，可松弛成功无限次



第$|V|$轮仍松弛成功的原因：存在源点可达的负环

小结

	广度优先搜索	Dijkstra算法	Bellman-Ford算法
适用范围	无权图	带权图 （所有边权为正）	带权图
松弛次数	--	$\|E\|$次	$\|V\| \cdot \|E\|$次
数据结构	队列	优先队列	--
运行时间	$O(\|V\| + \|E\|)$	$O((\|V\| + \|E\|) \cdot \log\|V\|)$	$O(\|E\| \cdot \|V\|)$

图算法篇：所有点对最短路径问题

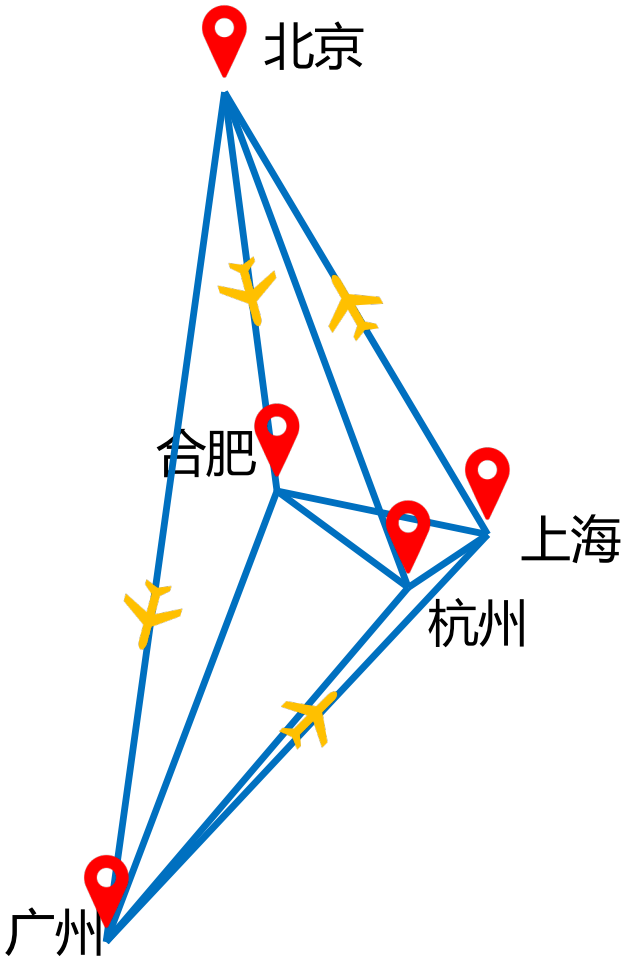
北京航空航天大学
计算机学院

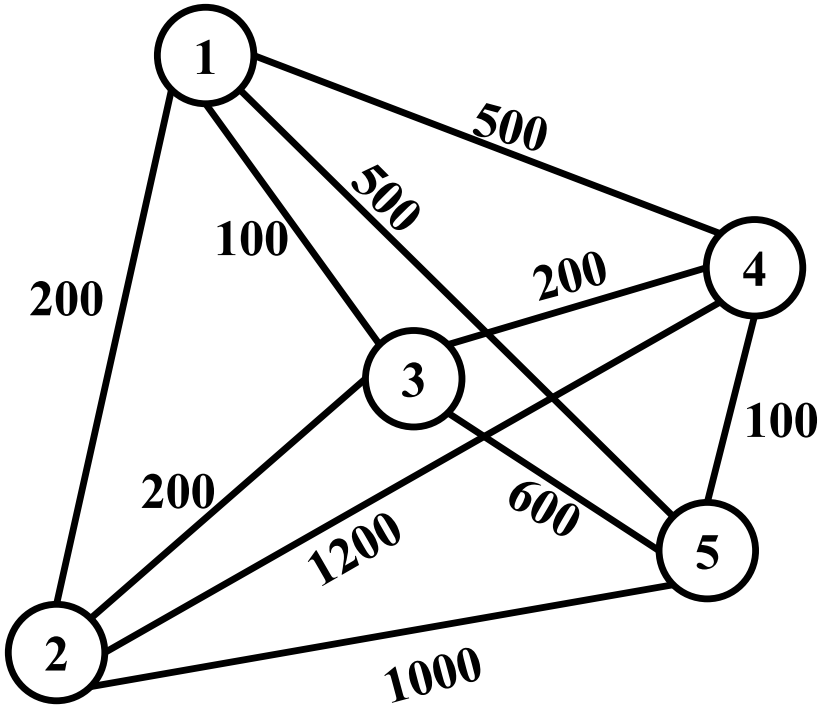
提纲

- 问题定义
- 算法思想
- 算法设计
- 算法实例
- 算法分析

问题背景

- 航班价格





如何求出所有城市之间的最低航班价格？

问题定义

所有点对最短路径问题

All Pairs Shortest Paths

输入

- 带权图 $G =< V, E, W >$, W 为边权

输出

- $\forall u, v \in V$, 从 u 到 v 的最短路径

提纲

问题定义

算法思想

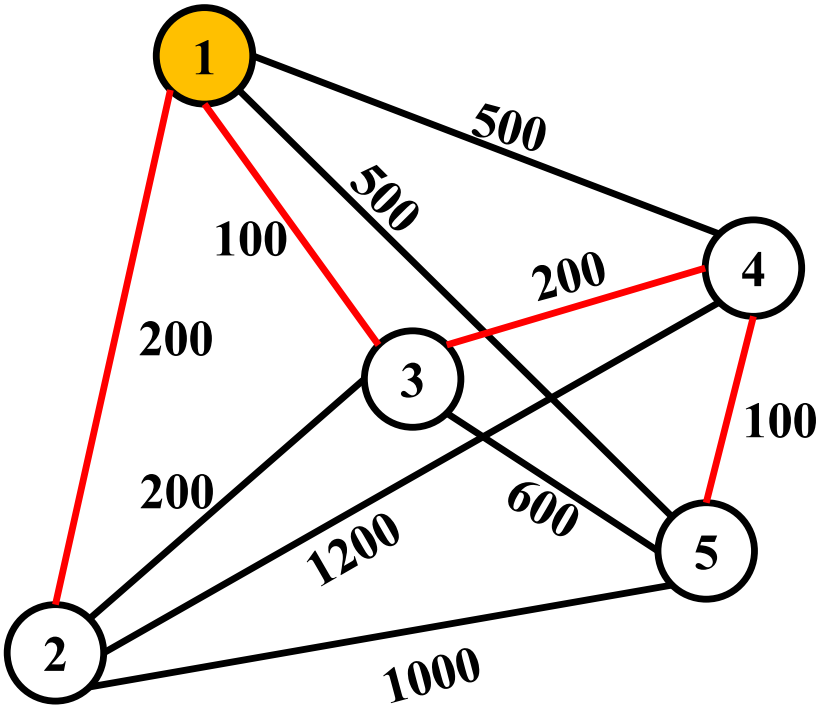
算法设计

算法实例

算法分析

直观思路

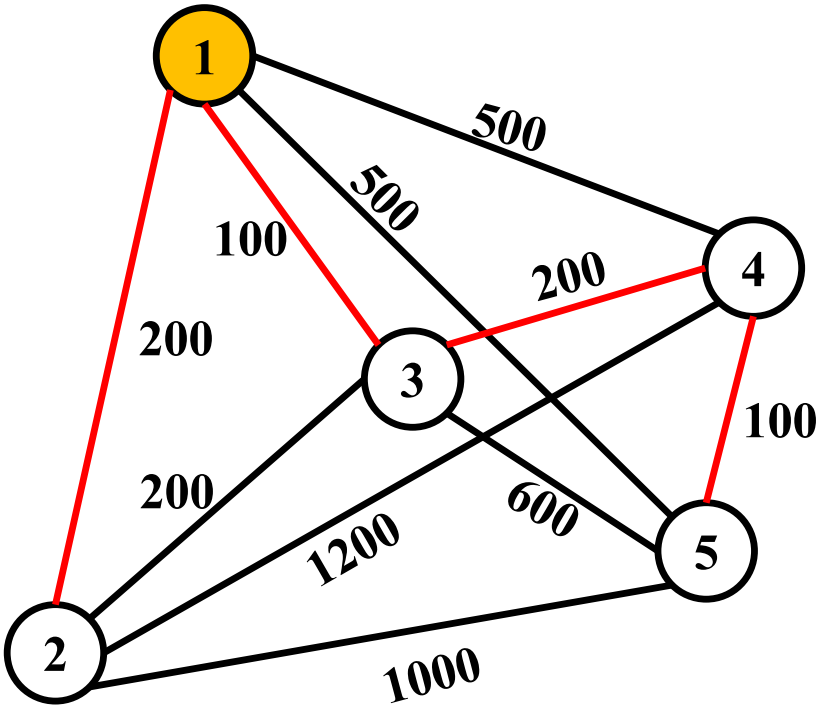
- 使用Dijkstra算法依次求解所有点



u \ v	1	2	3	4	5
1					
2					
3					
4					
5					

直观思路

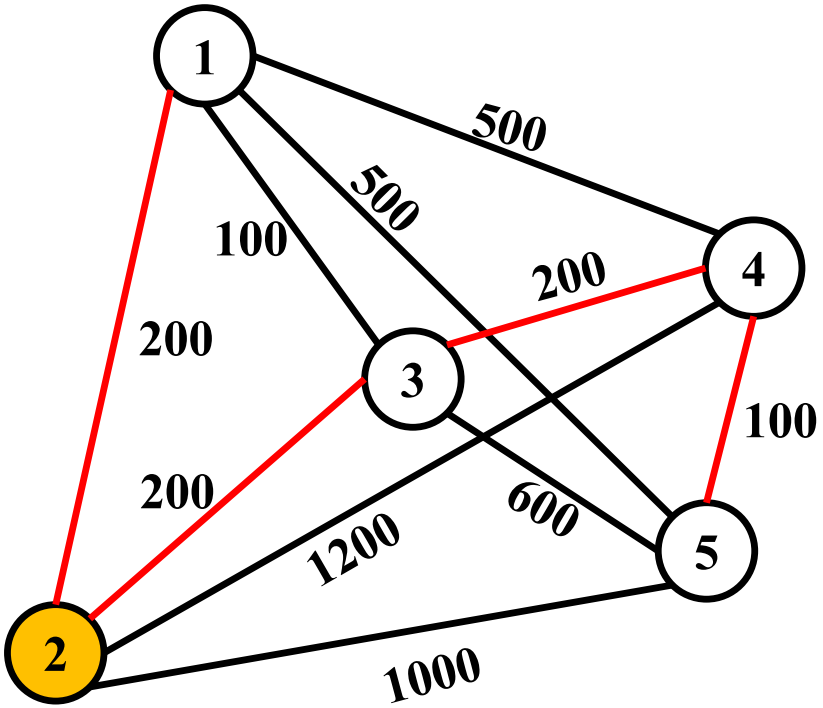
- 使用Dijkstra算法依次求解所有点



u \ v	1	2	3	4	5
1	0	1-2 200	1-3 100	1-3-4 300	1-3-4-5 400
2					
3					
4					
5					

直观思路

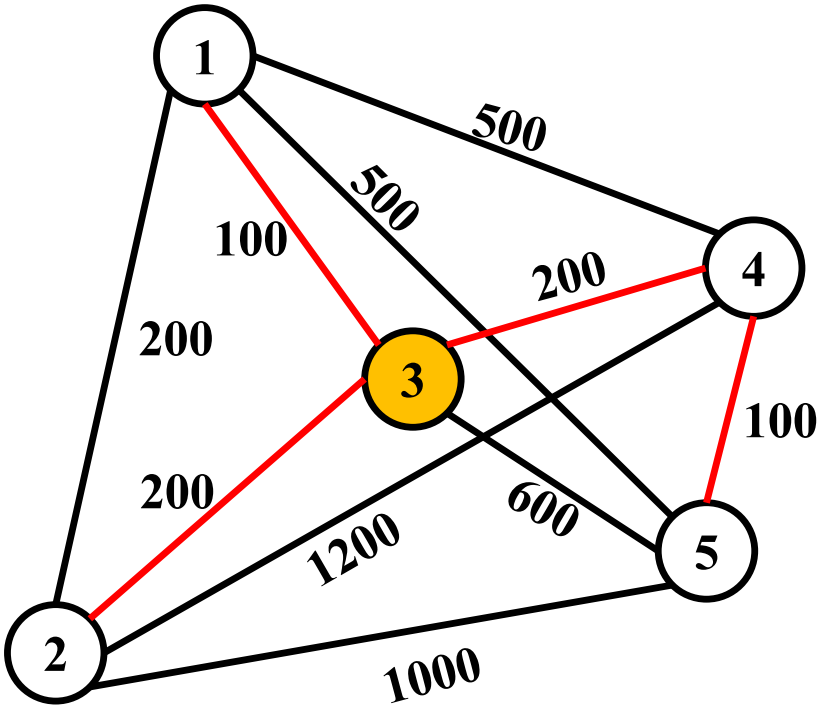
- 使用Dijkstra算法依次求解所有点



v / u	1	2	3	4	5
1	0	1-2 200	1-3 100	1-3-4 300	1-3-4-5 400
2	2-1 200	0	2-3 200	2-3-4 400	2-3-4-5 500
3					
4					
5					

直观思路

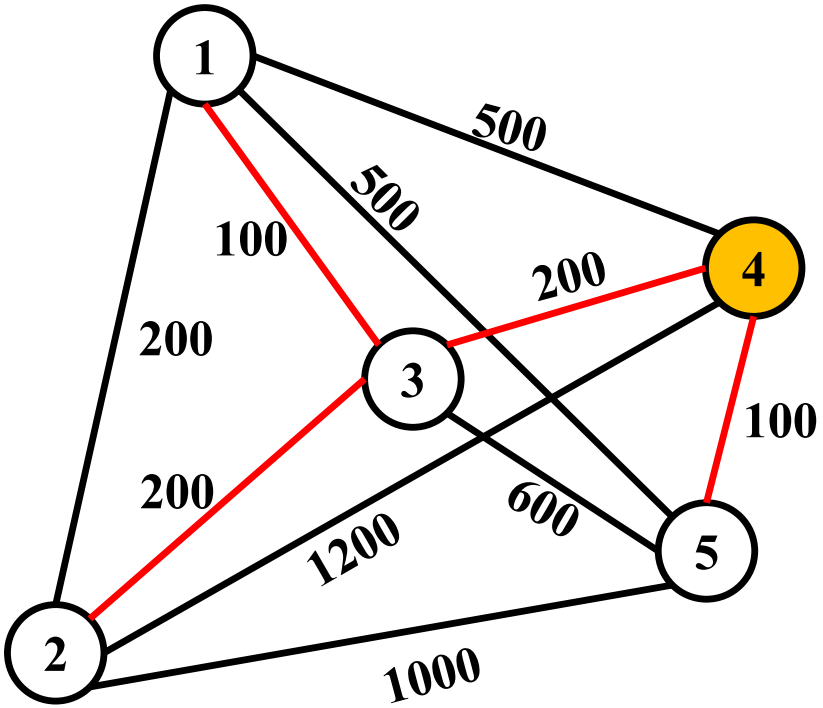
- 使用Dijkstra算法依次求解所有点



v / u	1	2	3	4	5
1	0	1-2 200	1-3 100	1-3-4 300	1-3-4-5 400
2	2-1 200	0	2-3 200	2-3-4 400	2-3-4-5 500
3	3-1 100	3-2 200	0	3-4 200	3-4-5 300
4					
5					

直观思路

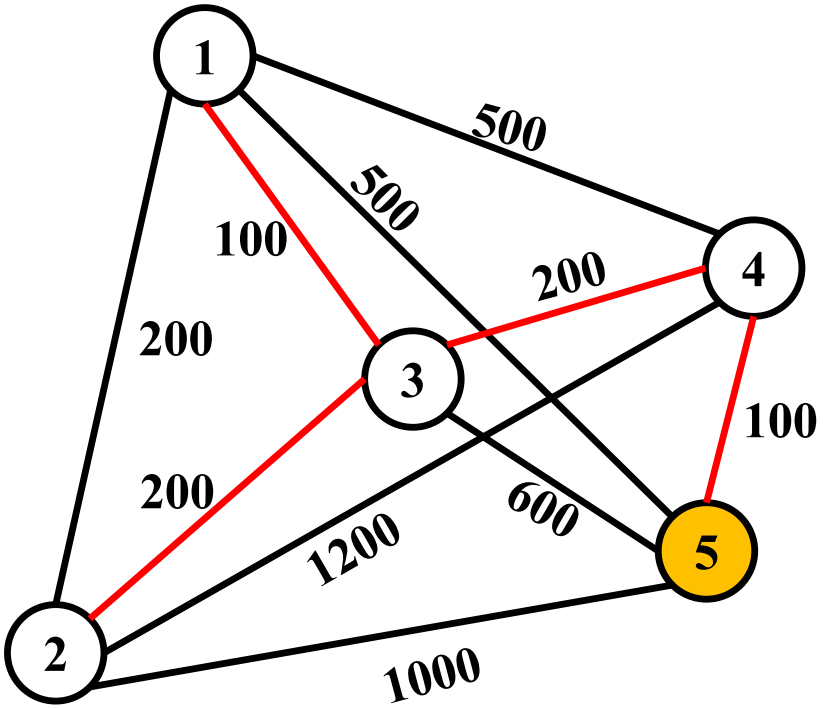
- 使用Dijkstra算法依次求解所有点



v / u	1	2	3	4	5
1	0	1-2 200	1-3 100	1-3-4 300	1-3-4-5 400
2	2-1 200	0	2-3 200	2-3-4 400	2-3-4-5 500
3	3-1 100	3-2 200	0	3-4 200	3-4-5 300
4	4-3-1 300	4-3-2 400	4-3 200	0	4-5 100
5					

直观思路

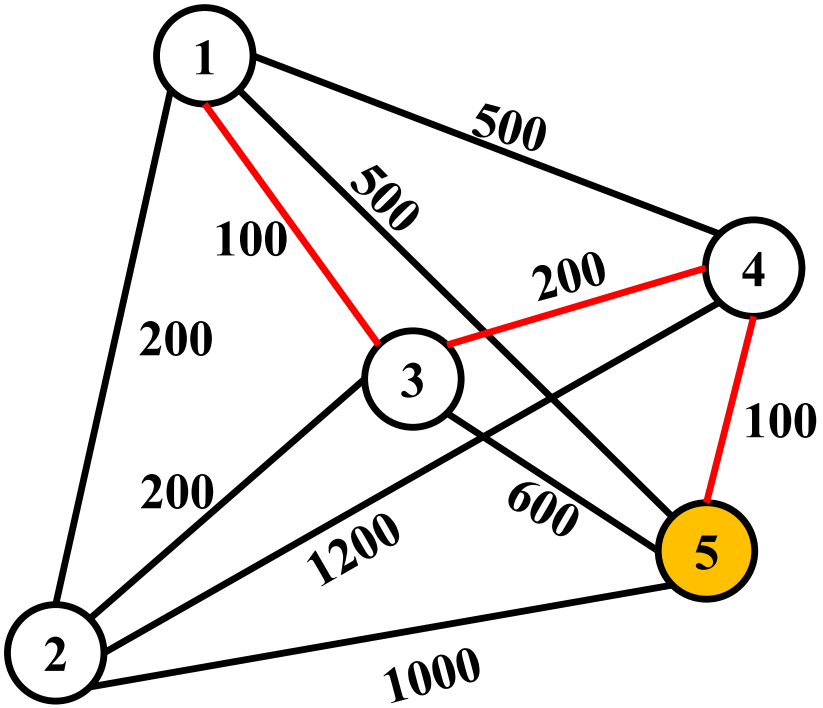
- 使用Dijkstra算法依次求解所有点



v / u	1	2	3	4	5
1	0	1-2 200	1-3 100	1-3-4 300	1-3-4-5 400
2	2-1 200	0	2-3 200	2-3-4 400	2-3-4-5 500
3	3-1 100	3-2 200	0	3-4 200	3-4-5 300
4	4-3-1 300	4-3-2 400	4-3 200	0	4-5 100
5	5-4-3-1 400	5-4-3-2 500	5-4-3 300	5-4 100	0

直观思路

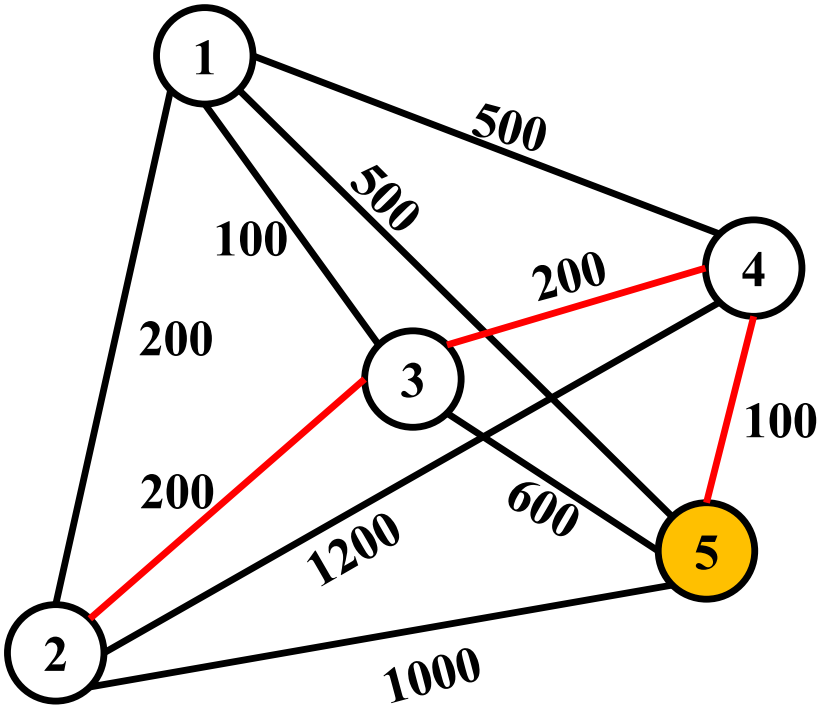
- 使用Dijkstra算法依次求解所有点
- 存在重叠子问题



v / u	1	2	3	4	5
1	0	1-2 200	1-3 100	1-3-4 300	1-3-4-5 400
2	2-1 200	0	2-3 200	2-3-4 400	2-3-4-5 500
3	3-1 100	3-2 200	0	3-4 200	3-4-5 300
4	4-3-1 300	4-3-2 400	4-3 200	0	4-5 100
5	5-4-3-1 400	5-4-3-2 500	5-4-3 300	5-4 100	0

直观思路

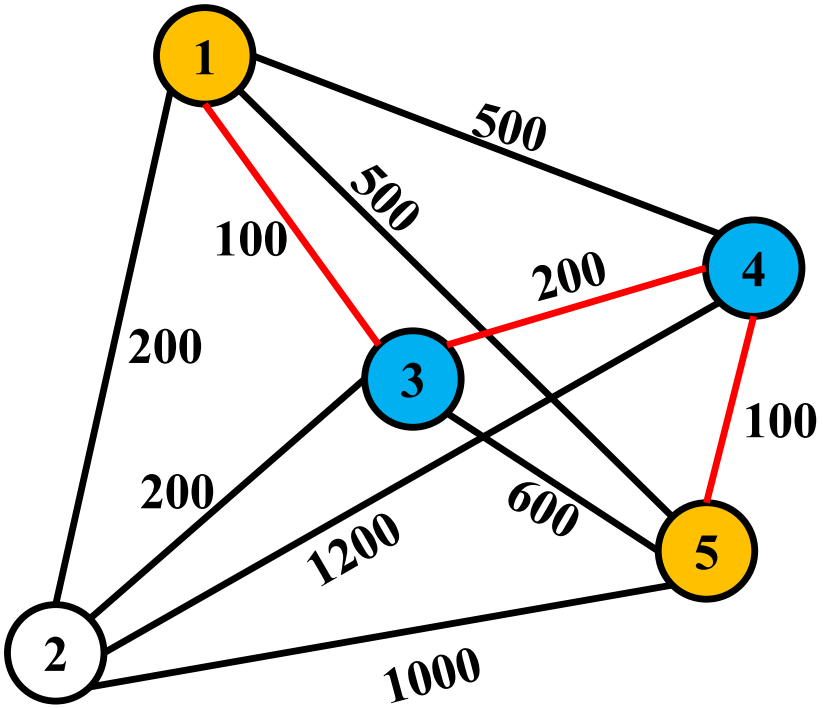
- 使用Dijkstra算法依次求解所有点
- 存在重叠子问题



v / u	1	2	3	4	5
1	0	1-2 200	1-3 100	1-3-4 300	1-3-4-5 400
2	2-1 200	0	2-3 200	2-3-4 400	2-3-4-5 500
3	3-1 100	3-2 200	0	3-4 200	3-4-5 300
4	4-3-1 300	4-3-2 400	4-3 200	0	4-5 100
5	5-4-3-1 400	5-4-3-2 500	5-4-3 300	5-4 100	0

直观思路

- 使用Dijkstra算法依次求解所有点
- 存在重叠子问题

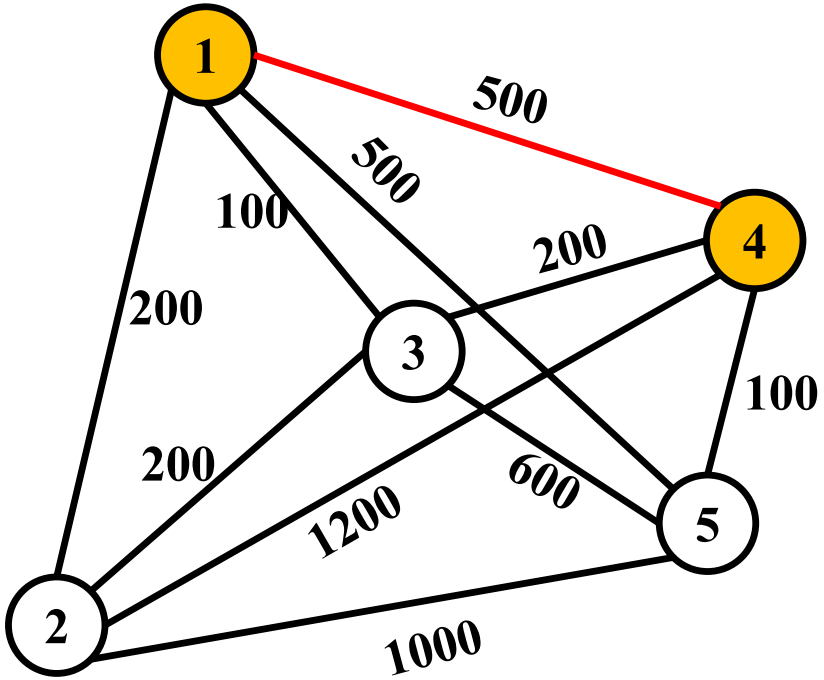


从 1 到 5 的最短路径：1→3→4→5

从 3 到 5 的最短路径：3→4→5

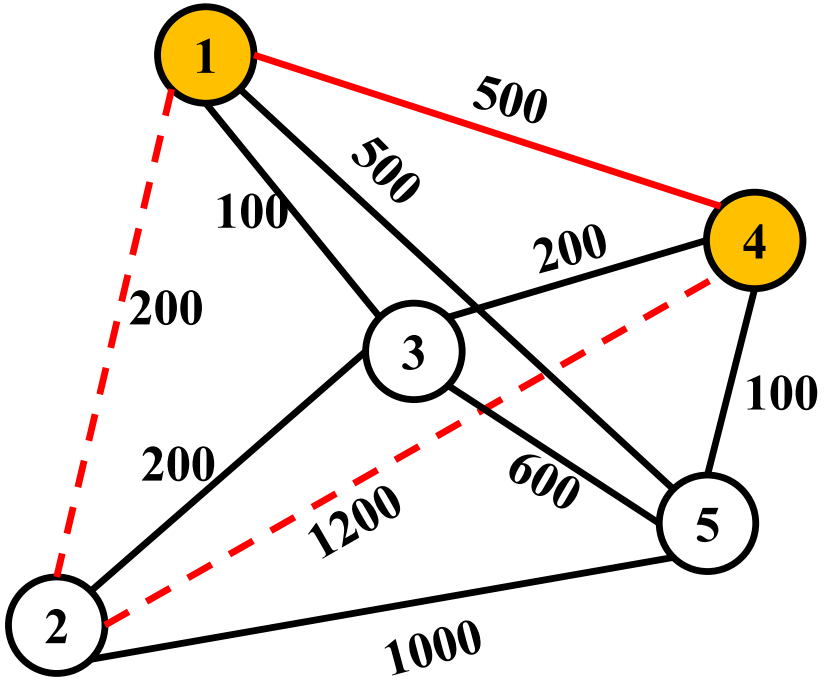
观察松弛过程

- 从 1 到 4 的路径更新
  - 可从前 1 个点（即{1}）中选择点经过：1-4, 500



观察松弛过程

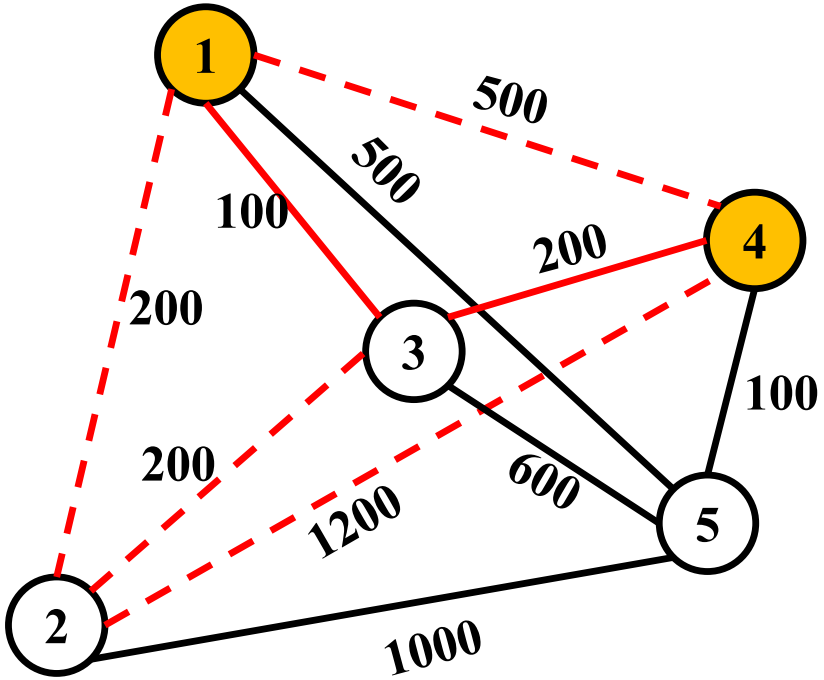
- 从 1 到 4 的路径更新
  - 可从前 1 个点（即{1}）中选择点经过：1-4, 500
  - 可从前 2 个点（即{1, 2}）中选择点经过：1-4, 500；1-2-4, 1400



观察松弛过程

- 从 1 到 4 的路径更新
  - 可从前 1 个点（即{1}）中选择点经过：1-4, 500
  - 可从前 2 个点（即{1, 2}）中选择点经过：1-4, 500；1-2-4, 1400
  - 可从前 3 个点（即{1, 2, 3}）中选择点经过：

    1-4, 500；1-2-4, 1400；1-3-4, 300；1-2-3-4，600

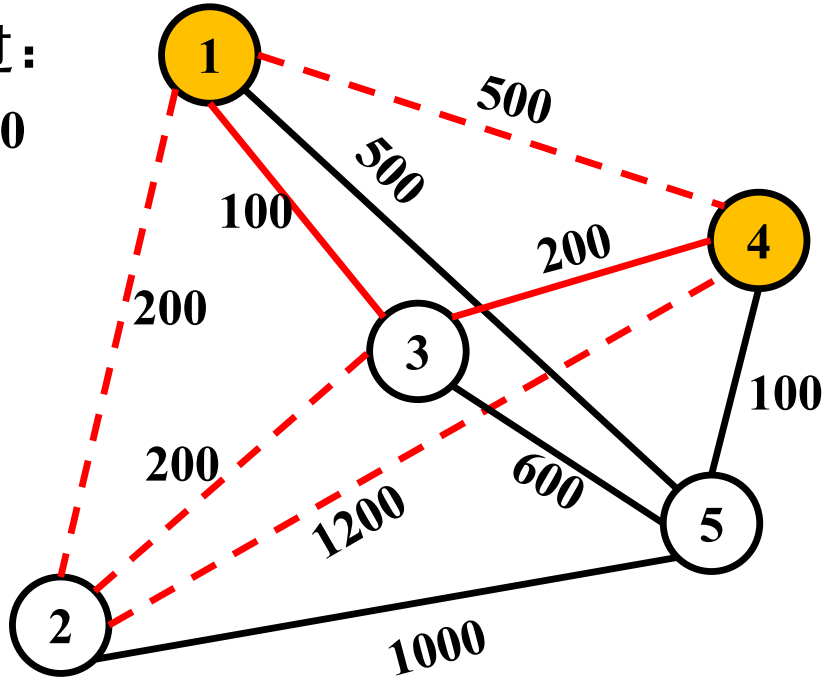


观察松弛过程

- 从 1 到 4 的路径更新
  - 可从前 1 个点（即{1}）中选择点经过：1-4, 500
  - 可从前 2 个点（即{1, 2}）中选择点经过：1-4, 500；1-2-4, 1400
  - 可从前 3 个点（即{1, 2, 3}）中选择点经过：

    1-4, 500；1-2-4, 1400；1-3-4, 300；1-2-3-4，600
  - 可从前 4 个点（即{1, 2, 3, 4}）中选择点经过：

1-4, 500；1-2-4, 1400；1-3-4, 300；1-2-3-4，600



观察松弛过程

- 从 1 到 4 的路径更新
  - 可从前 1 个点（即{1}）中选择点经过：1-4, 500
  - 可从前 2 个点（即{1, 2}）中选择点经过：1-4, 500；1-2-4, 1400
  - 可从前 3 个点（即{1, 2, 3}）中选择点经过：

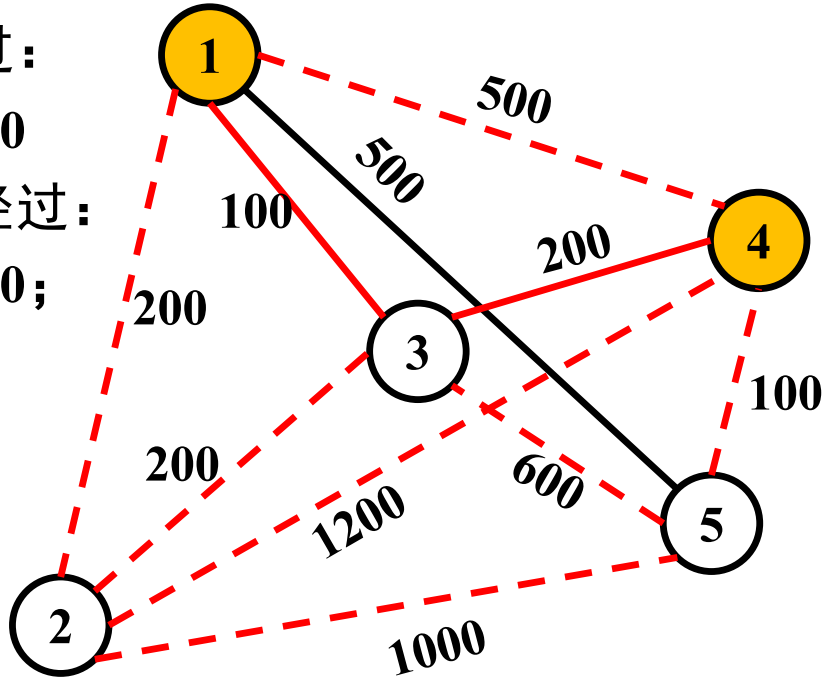
    1-4, 500；1-2-4, 1400；1-3-4, 300；1-2-3-4，600
  - 可从前 4 个点（即{1, 2, 3, 4}）中选择点经过：

    1-4, 500；1-2-4, 1400；1-3-4, 300；1-2-3-4，600
  - 可从前 5 个点（即{1, 2, 3, 4, 5}）中选择点经过：

    1-4, 500；1-2-4, 1400；1-3-4, 300；1-2-3-4，600；

    1-2-5-4, 1300; 1-3-5-4, 800; 1-2-3-5-4, 1100;

    1-3-2-5-4, 1400



观察松弛过程

- 从 1 到 4 的路径更新
  - 可从前 1 个点（即{1}）中选择点经过：1-4, 500
  - 可从前 2 个点（即{1, 2}）中选择点经过：1-4, 500；1-2-4, 1400
  - 可从前 3 个点（即{1, 2, 3}）中选择点经过：

    1-4, 500；1-2-4, 1400；1-3-4, 300；1-2-3-4，600
  - 可从前 4 个点（即{1, 2, 3, 4}）中选择点经过：

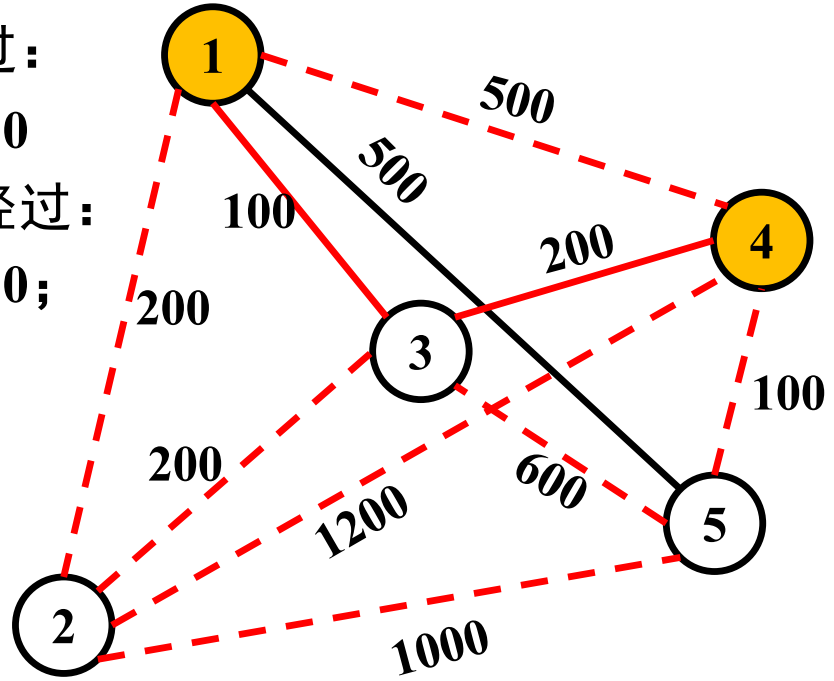
    1-4, 500；1-2-4, 1400；1-3-4, 300；1-2-3-4，600
  - 可从前 5 个点（即{1, 2, 3, 4, 5}）中选择点经过：

    1-4, 500；1-2-4, 1400；1-3-4, 300；1-2-3-4，600；

    1-2-5-4, 1300; 1-3-5-4, 800; 1-2-3-5-4, 1100;

    1-3-2-5-4, 1400

可经过的中间点越多距离逐渐变短



观察松弛过程

- 从 1 到 4 的路径更新
  - 可从前 1 个点（即{1}）中选择点经过：1-4, 500
  - 可从前 2 个点（即{1, 2}）中选择点经过：1-4, 500；1-2-4, 1400
  - 可从前 3 个点（即{1, 2, 3}）中选择点经过：

    1-4, 500；1-2-4, 1400；1-3-4, 300；1-2-3-4，600
  - 可从前 4 个点（即{1, 2, 3, 4}）中选择点经过：

    1-4, 500；1-2-4, 1400；1-3-4, 300；1-2-3-4，600
  - 可从前 5 个点（即{1, 2, 3, 4, 5}）中选择点经过：

    1-4, 500；1-2-4, 1400；1-3-4, 300；1-2-3-4，600；

    1-2-5-4, 1300; 1-3-5-4, 800; 1-2-3-5-4, 1100;

    1-3-2-5-4, 1400

最短路径不经过 2

最短路径经过 3

最短路径不经过 5

可经过的中间点越多距离逐渐变短

观察松弛过程

- 从 1 到 4 的路径更新
  - 可从前 1 个点（即{1}）中选择点经过：1-4, 500
  - 可从前 2 个点（即{1, 2}）中选择点经过：1-4, 500；1-2-4, 1400
  - 可从前 3 个点（即{1, 2, 3}）中选择点经过：

    1-4, 500；1-2-4, 1400；1-3-4, 300；1-2-3-4，600

最短路径不经过 2

最短路径经过 3

观察松弛过程

- 从 1 到 4 的路径更新
  - 可从前 1 个点（即{1}）中选择点经过：1-4, 500
  - 可从前 2 个点（即{1, 2}）中选择点经过：1-4, 500；1-2-4, 1400
  - 可从前 3 个点（即{1, 2, 3}）中选择点经过：
    1-4, 500；1-2-4, 1400；1-3-4, 300；1-2-3-4，600

最短路径不经过 2

最短路径经过 3

- 两种情况：
  - 加入可经过的新点 k 后，最短路径不经过新点，最短路径长度维持不变
  - 加入可经过的新点 k 后，最短路径经过新点，且最短路径长度变小

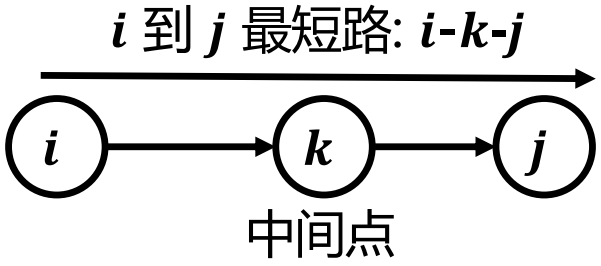
观察松弛过程

- 从 1 到 4 的路径更新
  - 可从前 1 个点（即{1}）中选择点经过：1-4, 500
  - 可从前 2 个点（即{1, 2}）中选择点经过：1-4, 500；1-2-4, 1400
  - 可从前 3 个点（即{1, 2, 3}）中选择点经过：
    1-4, 500；1-2-4, 1400；1-3-4, 300；1-2-3-4，600

最短路径不经过 2

最短路径经过 3

- 两种情况：
  - 加入可经过的新点 k 后，最短路径不经过新点，最短路径长度维持不变
  - 加入可经过的新点 k 后，最短路径经过新点，且最短路径长度变小



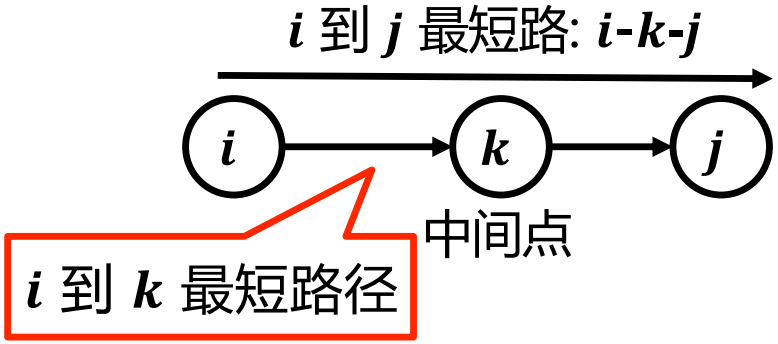
观察松弛过程

- 从 1 到 4 的路径更新
  - 可从前 1 个点（即{1}）中选择点经过：1-4, 500
  - 可从前 2 个点（即{1, 2}）中选择点经过：1-4, 500；1-2-4, 1400
  - 可从前 3 个点（即{1, 2, 3}）中选择点经过：
    1-4, 500；1-2-4, 1400；1-3-4, 300；1-2-3-4，600

最短路径不经过 2

最短路径经过 3

- 两种情况：
  - 加入可经过的新点 k 后，最短路径不经过新点，最短路径长度维持不变
  - 加入可经过的新点 k 后，最短路径经过新点，且最短路径长度变小



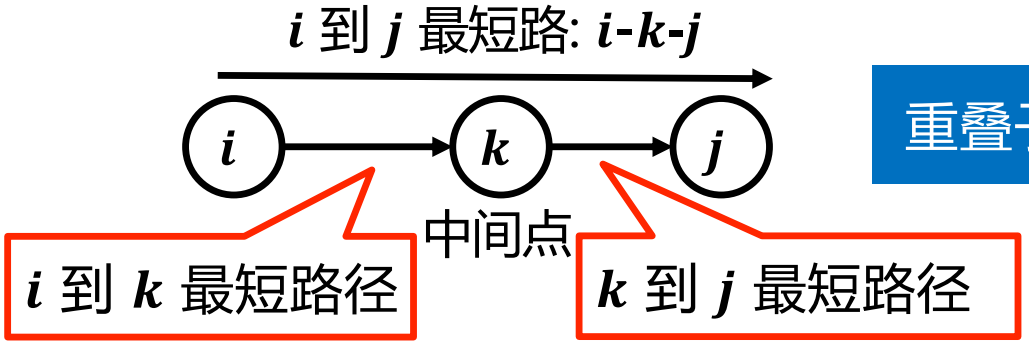
观察松弛过程

- 从 1 到 4 的路径更新
  - 可从前 1 个点（即{1}）中选择点经过：1-4, 500
  - 可从前 2 个点（即{1, 2}）中选择点经过：1-4, 500；1-2-4, 1400
  - 可从前 3 个点（即{1, 2, 3}）中选择点经过：
    1-4, 500；1-2-4, 1400；1-3-4, 300；1-2-3-4，600

最短路径不经过 2

最短路径经过 3

- 两种情况：
  - 加入可经过的新点 k 后，最短路径不经过新点，最短路径长度维持不变
  - 加入可经过的新点 k 后，最短路径经过新点，且最短路径长度变小



重叠子问题、最优子结构启发使用动态规划求解

提纲

问题定义

算法思想

算法设计

算法实例

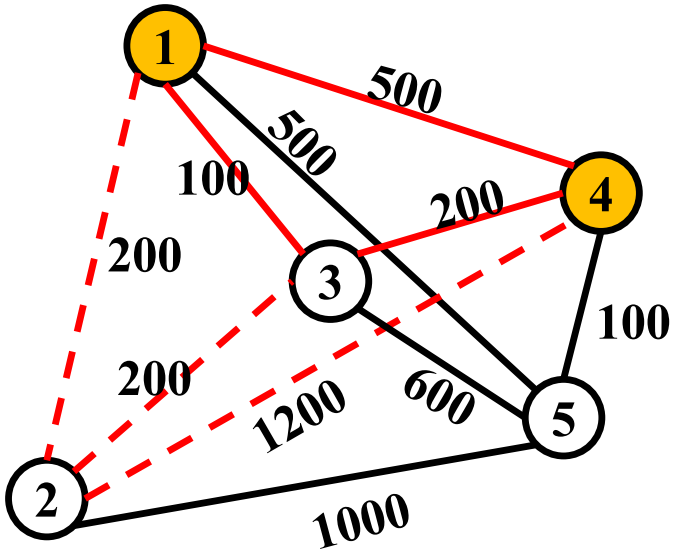
算法分析

动态规划：问题结构分析

- 给出问题表示
  - $D[k,i,j]$：可从前 k 个点选点经过时，i 到 j 的最短距离

动态规划：问题结构分析

- 给出问题表示
  - $D[k,i,j]$：可从前 k 个点选点经过时，i 到 j 的最短距离
- 从 1 到 4 的路径更新
  - 可从前 1 个点中选择点经过：$D[1,1,4]=500$ （1-4, 500）
  - 可从前 2 个点中选择点经过：$D[2,1,4]=500$ （1-4, 500）
  - 可从前 3 个点中选择点经过：$D[3,1,4]=300$ （1-3-4，300）

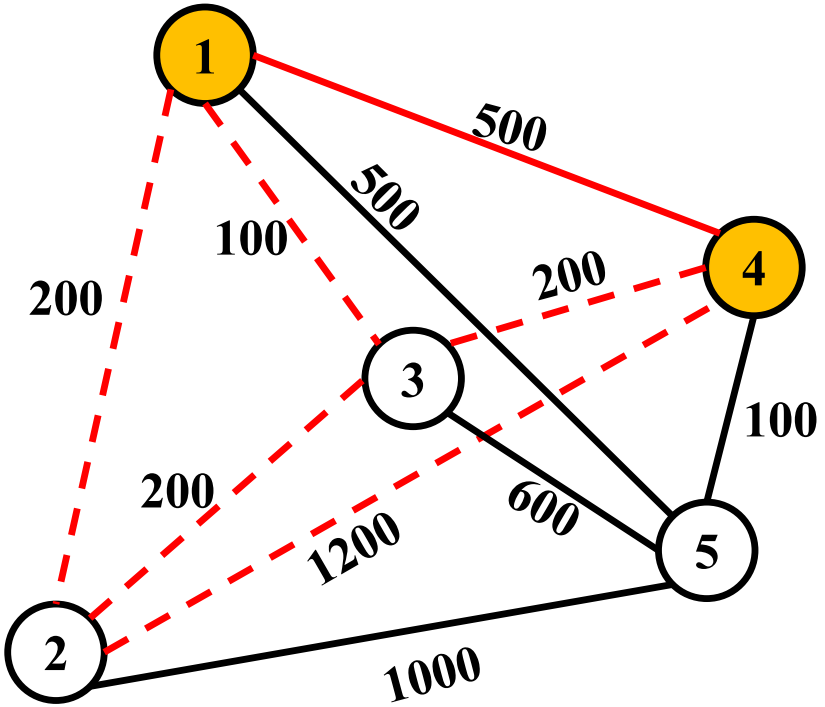


动态规划：递推关系建立

- 给出问题表示
  - $D[k,i,j]$：可从前 k 个点选点经过时，i 到 j 的最短距离
- 如果不选第 k 个点经过
  - $D[k,i,j]=D[k-1,i,j]$

动态规划：递推关系建立

- 从 1 到 4 的路径更新
  - 可从前 1 个点（即{1}）中选择点经过：1-4, 500
  - 可从前 2 个点（即{1, 2}）中选择点经过：1-4, 500



此时：$k = 2, i = 1, j = 4$

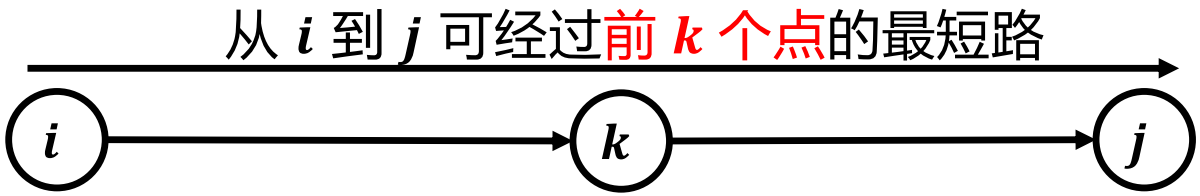
$$D[2,1,4] = D[1,1,4] = 500$$

动态规划：递推关系建立

- 给出问题表示
  - $D[k,i,j]$：可从前 k 个点选点经过时，i 到 j 的最短距离
- 如果不选第 k 个点经过
  - $D[k,i,j]=D[k-1,i,j]$
- 如果选择第 k 个点经过

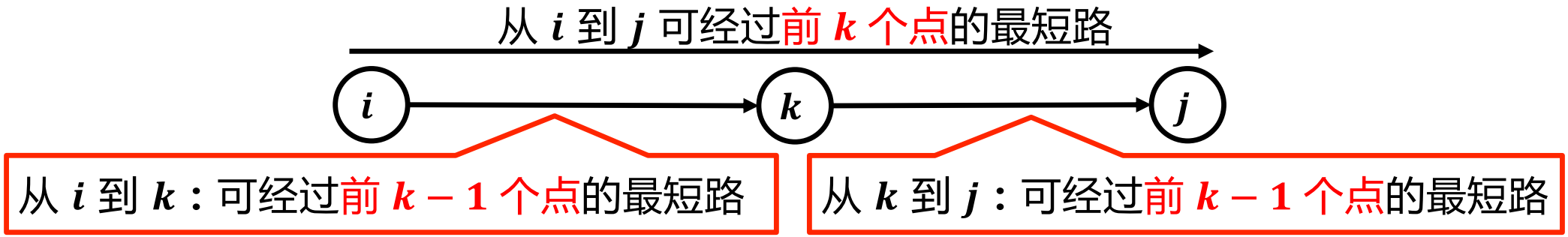
动态规划：递推关系建立

- 给出问题表示
  - $D[k,i,j]$：可从前 k 个点选点经过时，i 到 j 的最短距离
- 如果不选第 k 个点经过
  - $D[k,i,j] = D[k-1,i,j]$
- 如果选择第 k 个点经过



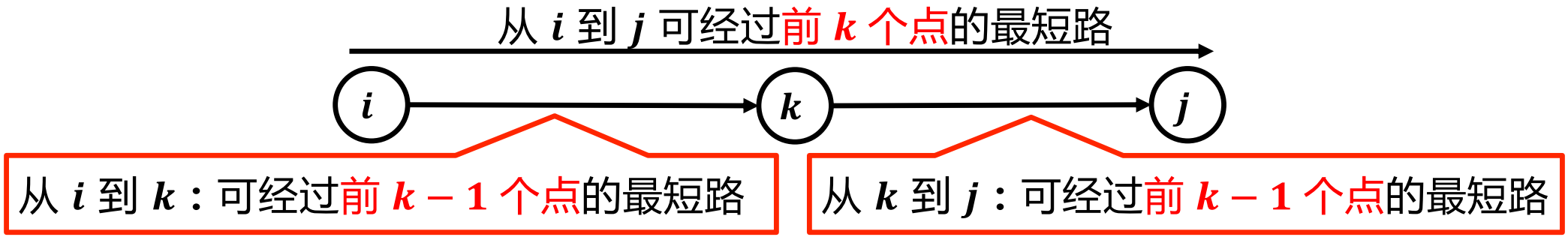
动态规划：递推关系建立

- 给出问题表示
  - $D[k,i,j]$：可从前 k 个点选点经过时，i 到 j 的最短距离
- 如果不选第 k 个点经过
  - $D[k,i,j]=D[k-1,i,j]$
- 如果选择第 k 个点经过



动态规划：递推关系建立

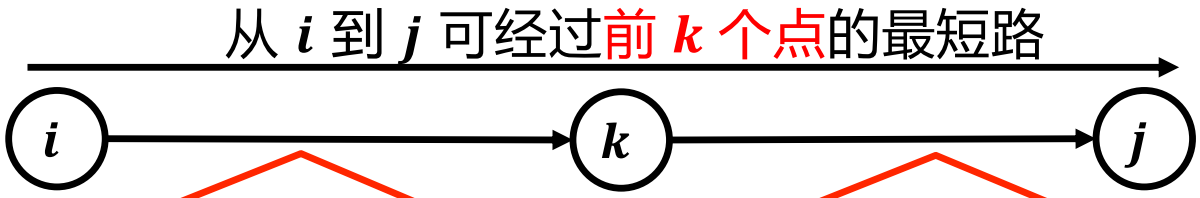
- 给出问题表示
  - $D[k,i,j]$：可从前 k 个点选点经过时，i 到 j 的最短距离
- 如果不选第 k 个点经过
  - $D[k,i,j]=D[k-1,i,j]$
- 如果选择第 k 个点经过
  - $D[k,i,j]=D[k-1,i,k]+D[k-1,k,j]$



动态规划：递推关系建立

- 给出问题表示
  - $D[k,i,j]$：可从前 k 个点选点经过时，i 到 j 的最短距离
- 如果不选第 k 个点经过
  - $D[k,i,j] = D[k-1,i,j]$
- 如果选择第 k 个点经过
  - $D[k,i,j] = D[k-1,i,k] + D[k-1,k,j]$

$$D[k,i,j] = \min\{ D[k-1,i,j], D[k-1,i,k] + D[k-1,k,j]\}$$



从 i 到 k：可经过前 $k-1$ 个点的最短路

从 k 到 j：可经过前 $k-1$ 个点的最短路

动态规划：递推关系建立

- 给出问题表示
  - $D[k,i,j]$：可从前 k 个点选点经过时，i 到 j 的最短距离
- 如果不选第 k 个点经过
  - $D[k,i,j] = D[k-1,i,j]$
- 如果选择第 k 个点经过
  - $D[k,i,j] = D[k-1,i,k] + D[k-1,k,j]$

$$D[k,i,j] = \min\{D[k-1,i,j], D[k-1,i,k] + D[k-1,k,j]\}$$

最优子结构

动态规划：自底向上计算（确定计算顺序）

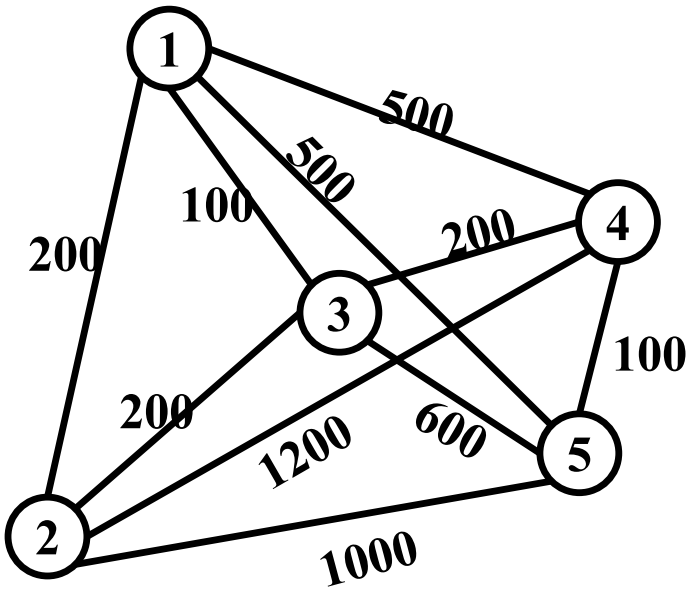
- 初始化

  $D[k,i,j]$：可从前 k 个点选点经过时，i 到 j 的最短距离

  - $D[0,i,i]=0$：起终点重合，路径长度为0

初始化的表格：$k=0$

i \ j	1	2	3	4	5
1	0				
2		0			
3			0		
4				0	
5					0



动态规划：自底向上计算（确定计算顺序）

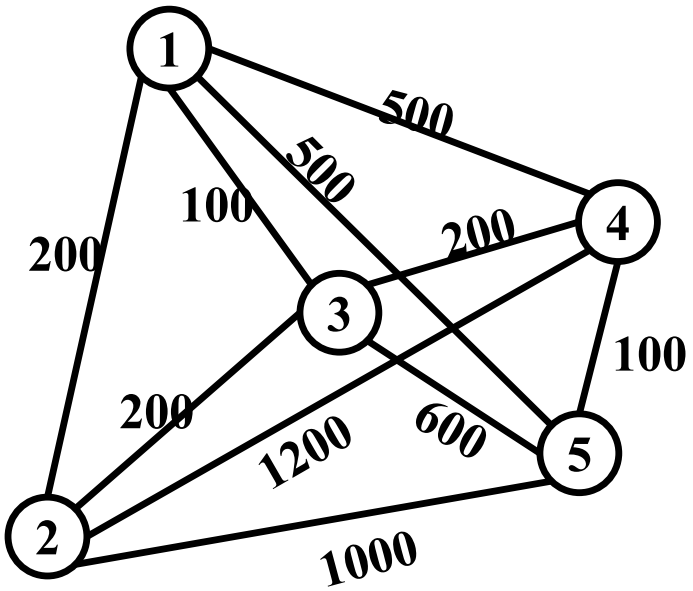
- 初始化

$D[k,i,j]$：可从前 k 个点选点经过时，i 到 j 的最短距离

  - $D[0,i,i]=0$：起终点重合，路径长度为0
  - $D[0,i,j]=e[i,j]$：任意两点直达距离为边权

初始化的表格：$k=0$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	1200	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0



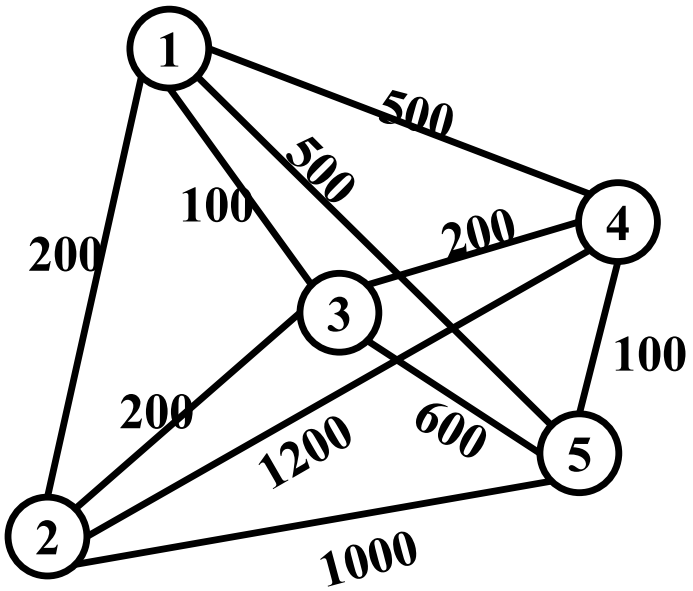
动态规划：自底向上计算（确定计算顺序）

$D[k,i,j]$：可从前 k 个点选点经过时，i 到 j 的最短距离

- 初始化
  - $D[0,i,i]=0$：起终点重合，路径长度为0
  - $D[0,i,j]=e[i,j]$：任意两点直达距离为边权
- $D[1,i,j]=\min\{D[0,i,j],D[0,i,1]+D[0,1,j]\}$

初始化的表格：$k=1$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	1200	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0



动态规划：自底向上计算（确定计算顺序）

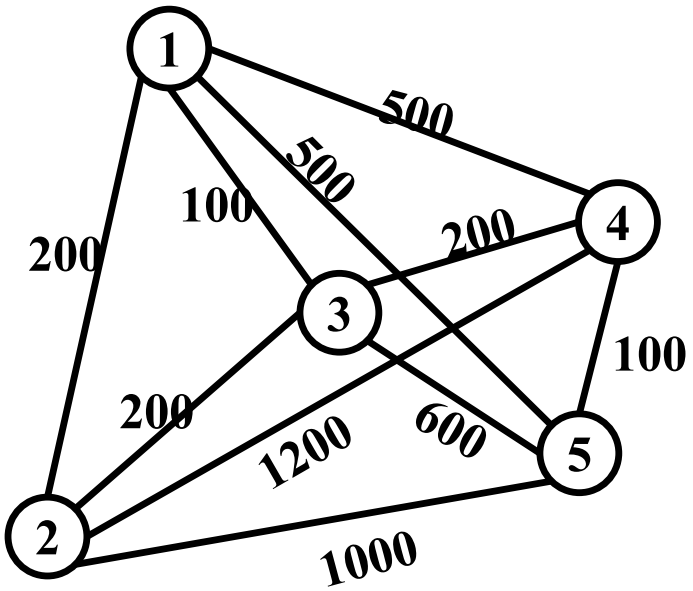
- 初始化

  $D[k,i,j]$：可从前 k 个点选点经过时，i 到 j 的最短距离

  - $D[0,i,i]=0$：起终点重合，路径长度为0
  - $D[0,i,j]=e[i,j]$：任意两点直达距离为边权
- $D[1,i,j]=\min\{D[0,i,j],D[0,i,1]+D[0,1,j]\}$

初始化的表格：$k=1$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0



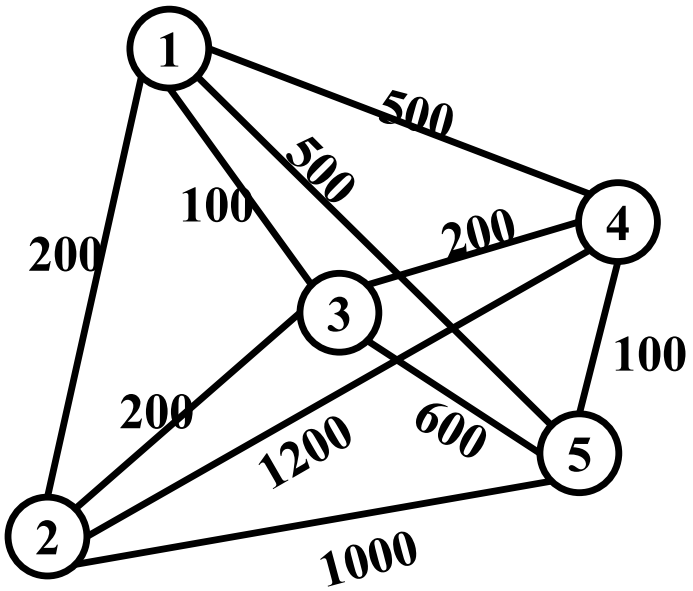
动态规划：自底向上计算（确定计算顺序）

- 初始化
  - $D[0,i,i]=0$：起终点重合，路径长度为0
  - $D[0,i,j]=e[i,j]$：任意两点直达距离为边权
- $D[1,i,j]=\min\{D[0,i,j],D[0,i,1]+D[0,1,j]\}$

$D[k,i,j]$：可从前 k 个点选点经过时，i 到 j 的最短距离

初始化的表格：$k=1$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0



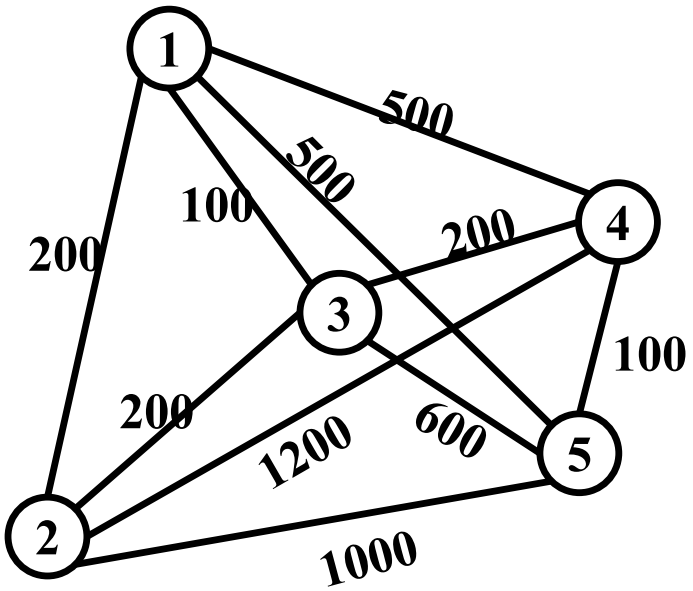
动态规划：自底向上计算（确定计算顺序）

- 初始化
  - $D[0,i,i]=0$：起终点重合，路径长度为0
  - $D[0,i,j]=e[i,j]$：任意两点直达距离为边权
- $D[1,i,j]=\min\{D[0,i,j],D[0,i,1]+D[0,1,j]\}$

$D[k,i,j]$：可从前 k 个点选点经过时，i 到 j 的最短距离

初始化的表格：$k=1$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	1000	600	100	0



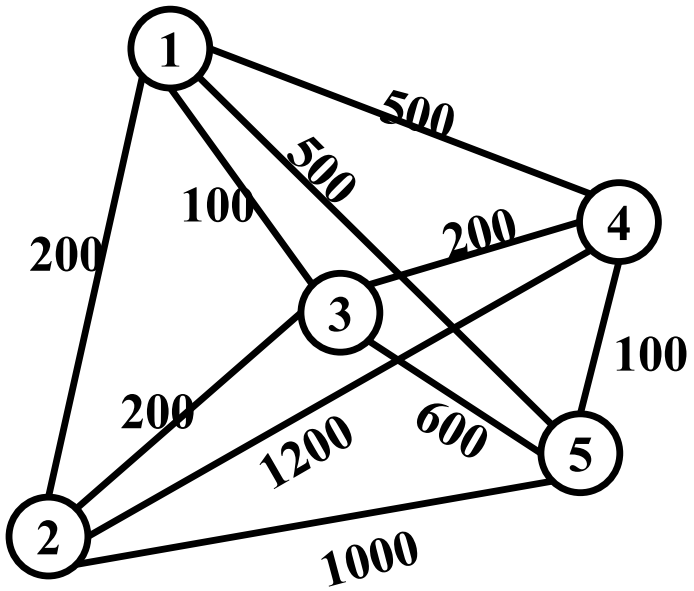
动态规划：自底向上计算（确定计算顺序）

$D[k,i,j]$：可从前 k 个点选点经过时，i 到 j 的最短距离

- 初始化
  - $D[0,i,i]=0$：起终点重合，路径长度为0
  - $D[0,i,j]=e[i,j]$：任意两点直达距离为边权
- $D[1,i,j]=\min\{D[0,i,j],D[0,i,1]+D[0,1,j]\}$

初始化的表格：$k=1$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	700	600	100	0



动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j] = \min\{D[k-1,i,j], D[k-1,i,k] + D[k-1,k,j]\}$

最终的表格：$k = |V|$

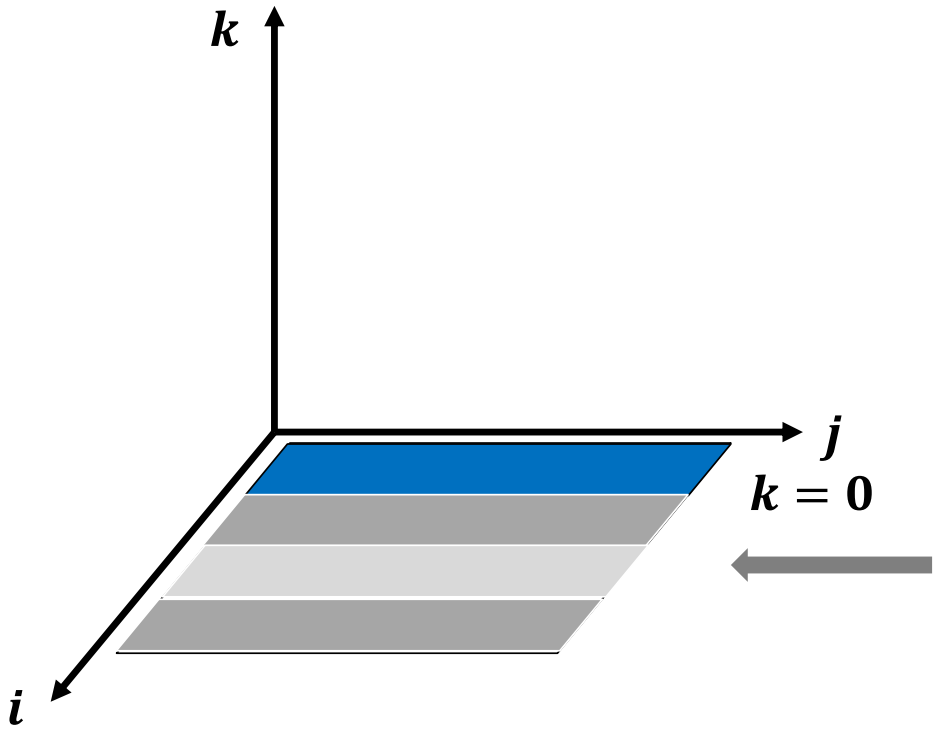
i \ j	1	…	…	…	…
1					
…			?		
…					
…					
…					

初始化的表格：$k = 0$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	1200	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j] = \min\{D[k-1,i,j], D[k-1,i,k] + D[k-1,k,j]$



最终的表格：$k = |V|$

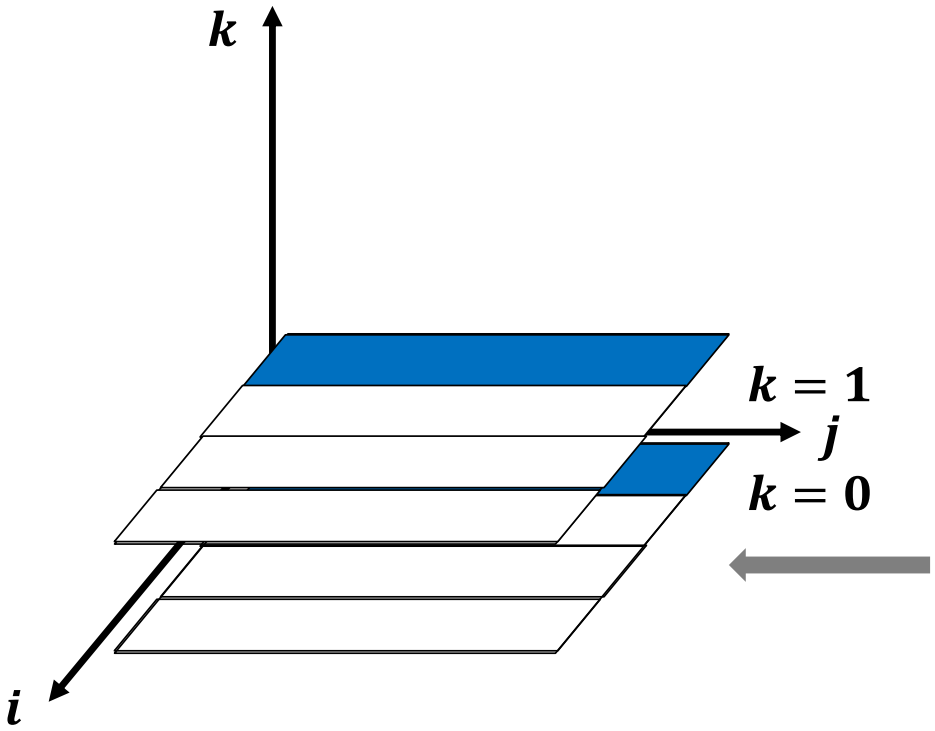
i \ j	1	…	…	…	…
1					
…			?		
…					
…					
…					

初始化的表格：$k = 0$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	1200	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j] = \min\{D[k-1,i,j], D[k-1,i,k] + D[k-1,k,j]$



最终的表格：$k = |V|$

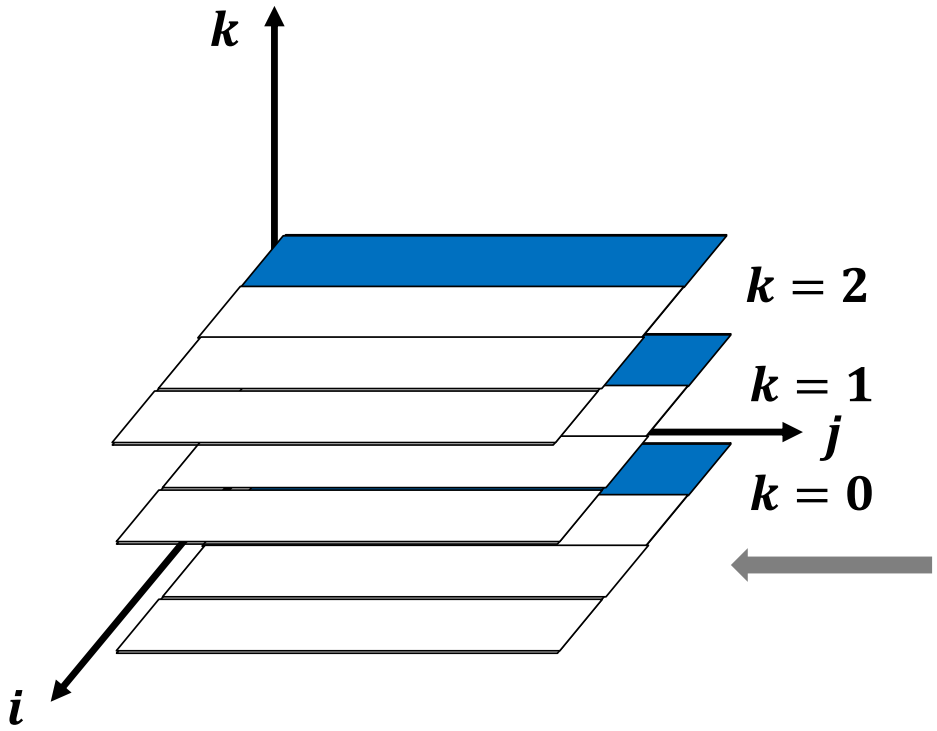
i \ j	1	…	…	…	…
1					
…			?		
…					
…					
…					

初始化的表格：$k = 0$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	1200	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j] = \min\{D[k-1,i,j], D[k-1,i,k] + D[k-1,k,j]$



最终的表格：$k = |V|$

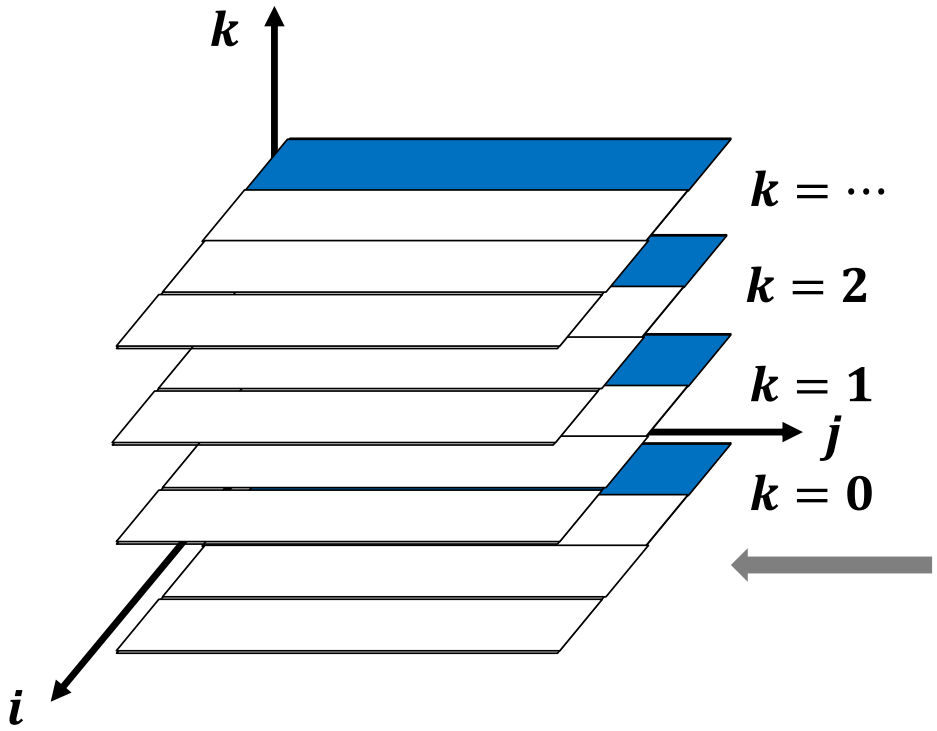
i \ j	1	…	…	…	…
1					
…			?		
…					
…					
…					

初始化的表格：$k = 0$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	1200	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j] = \min\{D[k-1,i,j], D[k-1,i,k] + D[k-1,k,j]$



最终的表格：$k = |V|$

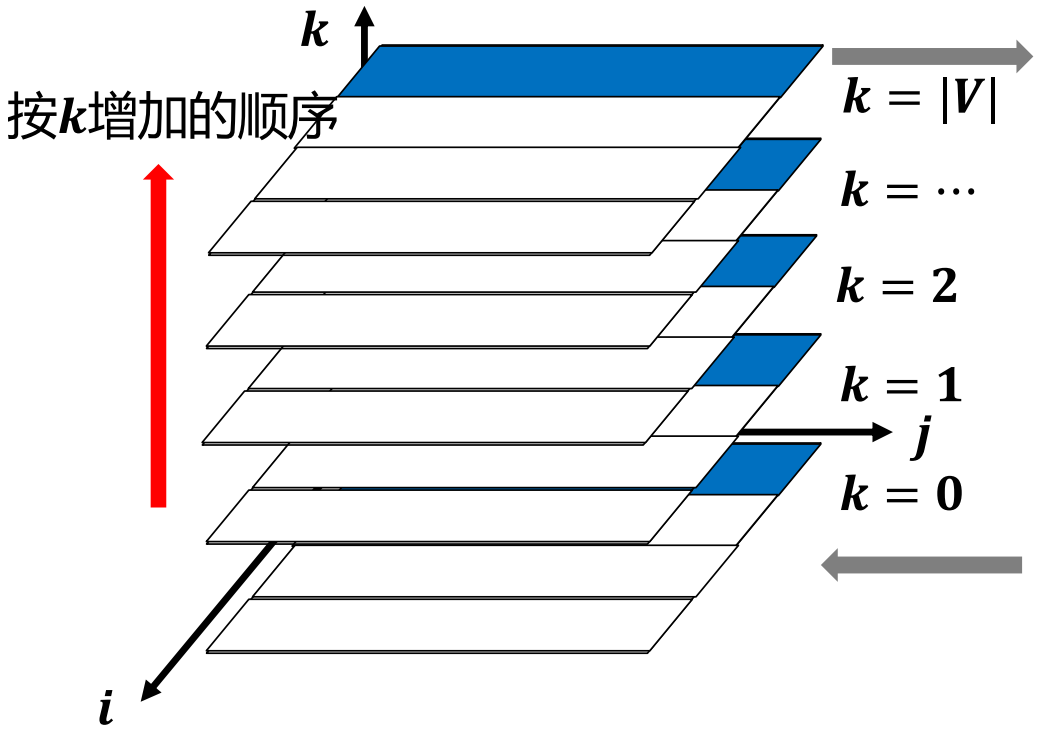
i \ j	1	…	…	…	…
1					
…			?		
…					
…					
…					

初始化的表格：$k = 0$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	1200	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j]=\min\{D[k-1,i,j], D[k-1,i,k]+D[k-1,k,j]$



最终的表格：$k=|V|$

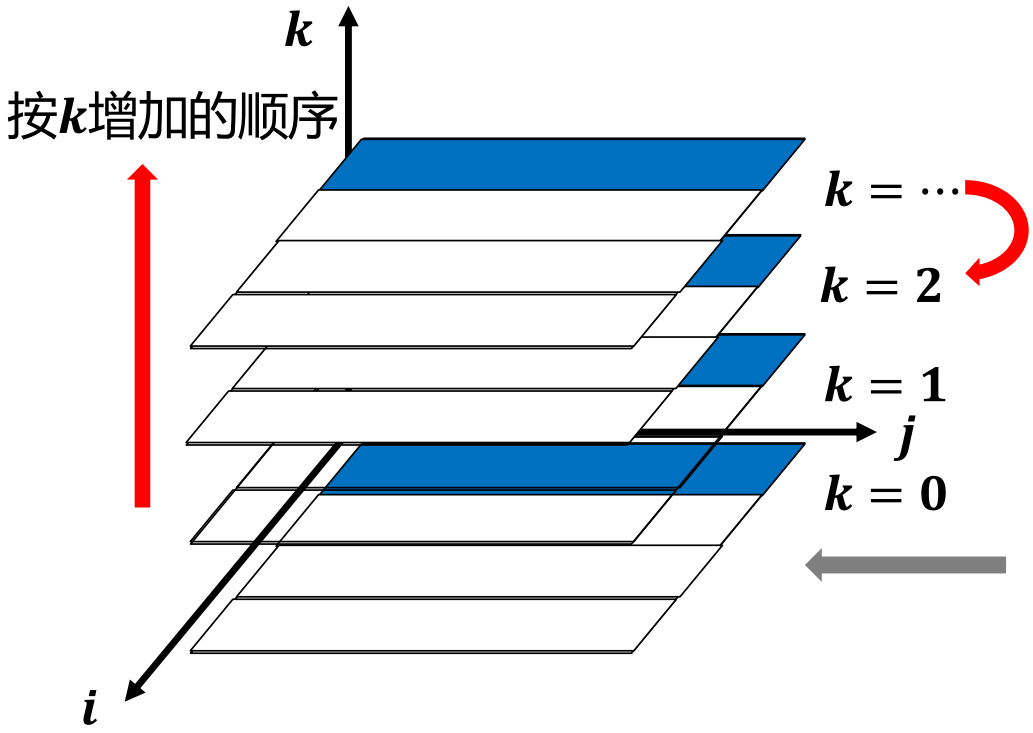
i \ j	1	…	…	…	…
1					
…			?		
…					
…					
…					

初始化的表格：$k=0$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	1200	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j] = \min\{D[k-1,i,j], D[k-1,i,k] + D[k-1,k,j]$



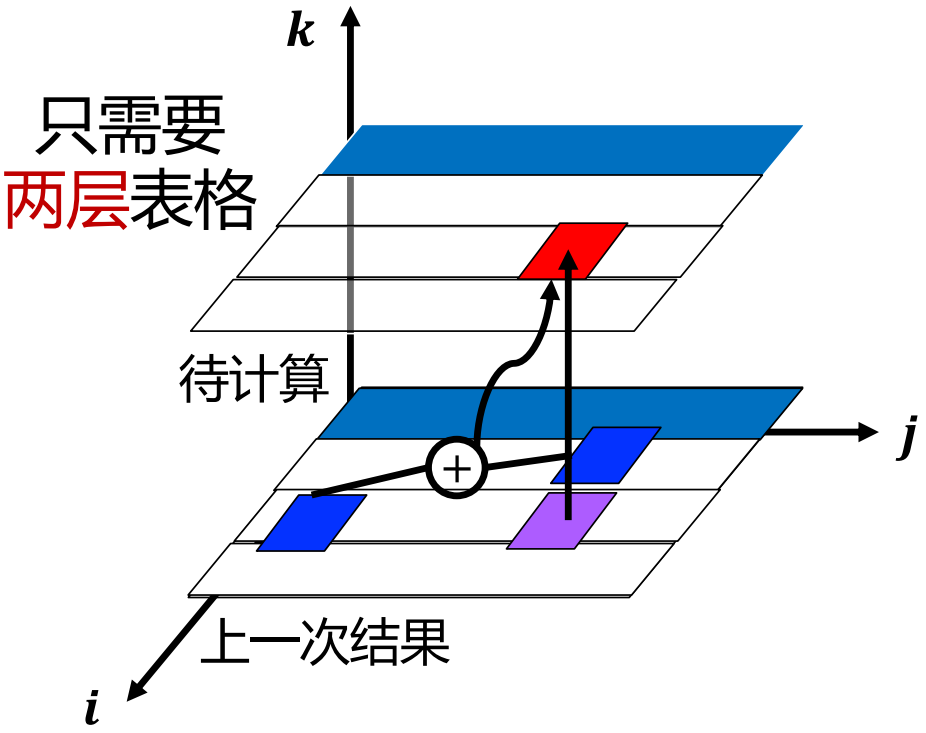
求解当前层只依赖上一层

初始化的表格：$k = 0$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	1200	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j] = \min\{D[k-1,i,j], D[k-1,i,k] + D[k-1,k,j]\}$



问题：是否能只需要一层表格？

动态规划：自底向上计算（确定计算顺序）

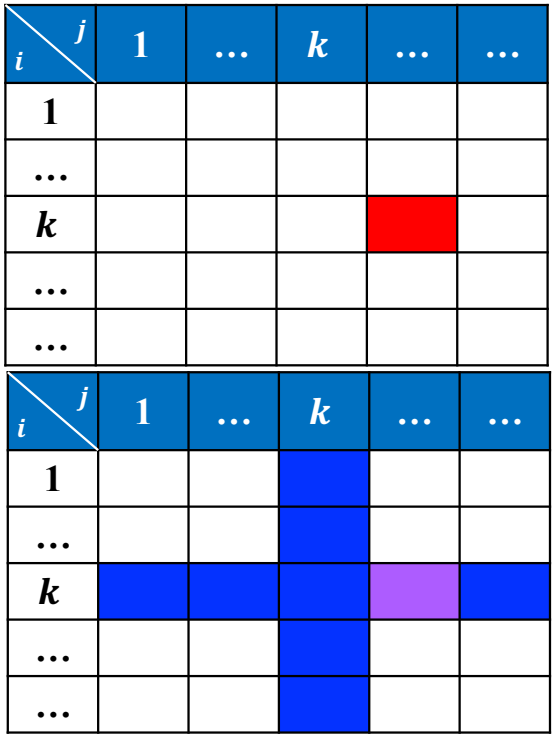
- 递推公式
  - $D[k,i,j] = \min\{D[k-1,i,j], D[k-1,i,k] + D[k-1,k,j]\}$
- 若$k = i$

$$
\begin{aligned}
&D[k-1,i,k] + D[k-1,k,j] \\
&= D[k-1,k,k] + D[k-1,k,j] \\
&= 0 + D[k-1,i,j] = D[k-1,i,j]
\end{aligned}
$$

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j] = \min\{D[k-1,i,j], D[k-1,i,k] + D[k-1,k,j]\}$
- 若$k = i$
  - $D[k,i,j] = D[k-1,i,j]$

$$D[k-1,i,k] + D[k-1,k,j]$$
$$= D[k-1,k,k] + D[k-1,k,j]$$
$$= 0 + D[k-1,i,j] = D[k-1,i,j]$$



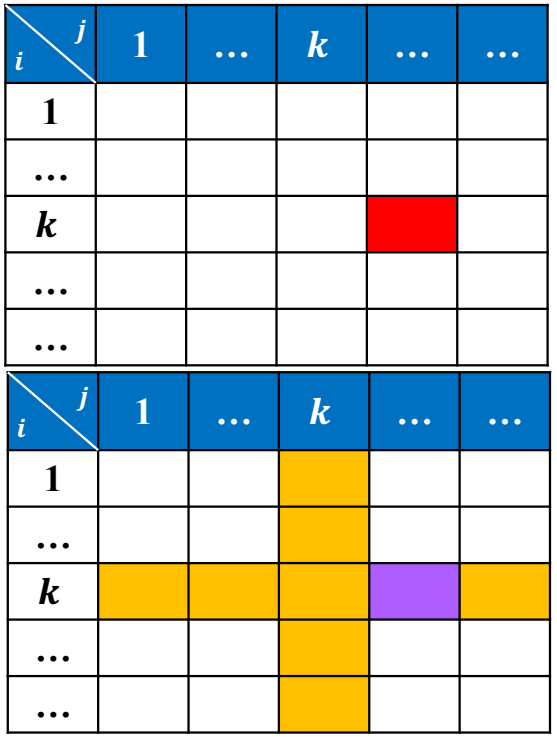
动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j] = \min\{D[k-1,i,j], D[k-1,i,k] + D[k-1,k,j]\}$
- 若$k = i$
  - $D[k,i,j] = D[k-1,i,j]$
  - 值相同，可以直接覆盖

$D[k-1,i,k] + D[k-1,k,j]$

$= D[k-1,k,k] + D[k-1,k,j]$

$= 0 + D[k-1,i,j] = D[k-1,i,j]$

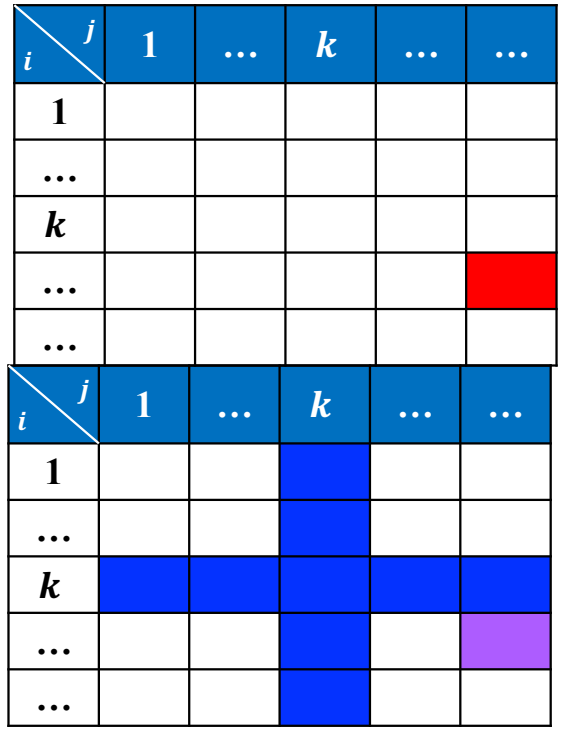


动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j] = \min\{D[k-1,i,j], D[k-1,i,k] + D[k-1,k,j]\}$
- 若$k = i$ 或 $k = j$
  - $D[k,i,j] = D[k-1,i,j]$
  - 值相同，可以直接覆盖

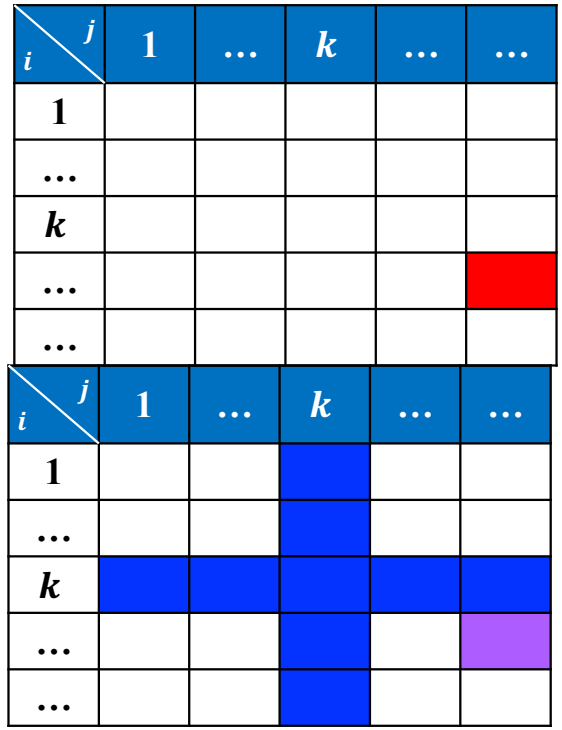
动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j] = \min\{D[k-1,i,j], D[k-1,i,k] + D[k-1,k,j]\}$
- 若$k = i$ 或 $k = j$
  - $D[k,i,j] = D[k-1,i,j]$
  - 值相同，可以直接覆盖
- 若$k \neq i$且$k \neq j$
  - $D[k-1,i,j]$ 和 $D[k-1,i,k], D[k-1,k,j]$ 不是相同子问题



动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j] = \min\{D[k-1,i,j], D[k-1,i,k] + D[k-1,k,j]\}$
- 若$k = i$ 或 $k = j$
  - $D[k,i,j] = D[k-1,i,j]$
  - 值相同，可以直接覆盖
- 若$k \neq i$且$k \neq j$
  - $D[k-1,i,j]$ 和 $D[k-1,i,k], D[k-1,k,j]$ 不是相同子问题
  - 求出 $D[k,i,j]$ 后，$D[k-1,i,j]$ 不再被使用，可直接覆盖



动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j] = \min\{D[k-1,i,j], D[k-1,i,k] + D[k-1,k,j]\}$
- 若$k = i$ 或 $k = j$
  - $D[k,i,j] = D[k-1,i,j]$
  - 值相同，可以直接覆盖
- 若$k \neq i$且$k \neq j$
  - $D[k-1,i,j]$ 和 $D[k-1,i,k], D[k-1,k,j]$ 不是相同子问题
  - 求出 $D[k,i,j]$ 后，$D[k-1,i,j]$ 不再被使用，可直接覆盖

求出新值可直接在原位置覆盖，只需存储一层表格

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

动态规划：最优方案追踪

- 递推公式
  - $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
- 追踪数组Rec，记录经过的中间点
  - $D_k[i,j] = D_{k-1}[i,j]$：0 表示没有中间点

Rec

i \ j	1	…	j	…	V
1					
…					
i			0		
…					
V					

动态规划：最优方案追踪

- 递推公式
  - $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
- 追踪数组Rec，记录经过的中间点
  - $D_k[i,j] = D_{k-1}[i,j]$：0 表示没有中间点
  - $D_k[i,j] = D_{k-1}[i,k] + D_{k-1}[k,j]$：$k$ 表示经过中间点k

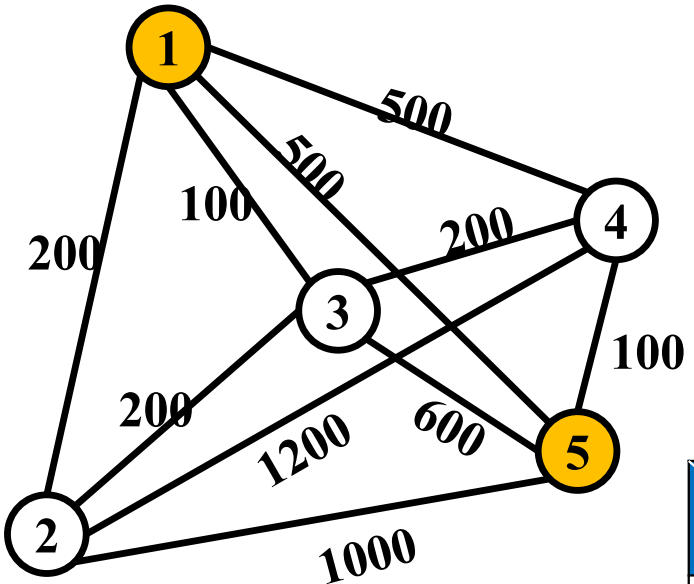
松弛时使用的点

Rec

i \ j	1	…	j	…	V
1					
…					
i			k		
…					
V					

动态规划：最优方案追踪

- 根据数组Rec，输出最短路径



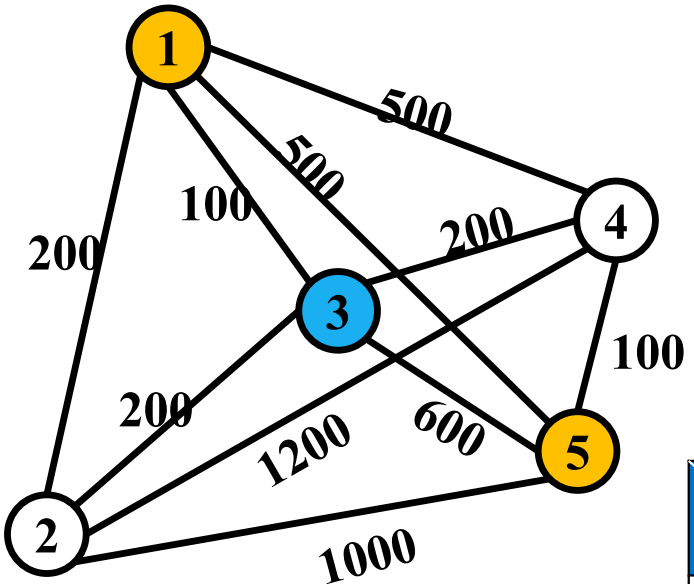
Rec

i \ j	1	2	3	4	5
1			0		3
2					
3				0	4
4					0
5					

1-3-5

动态规划：最优方案追踪

- 根据数组Rec，输出最短路径



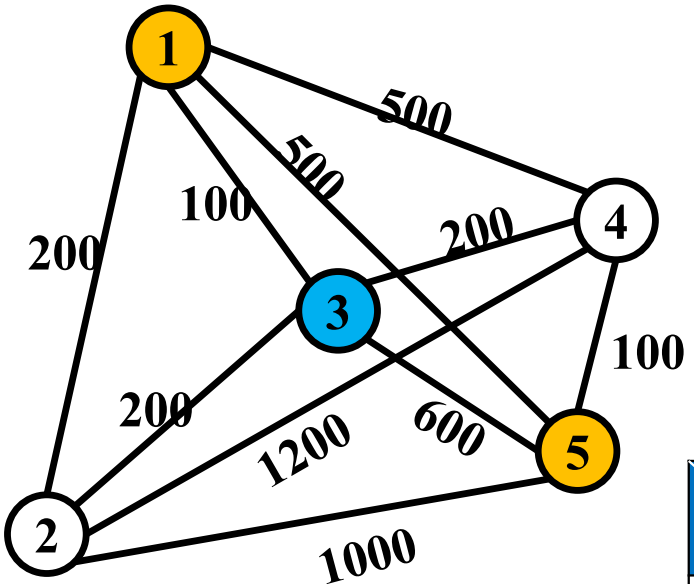
Rec

i \ j	1	2	3	4	5
1			0		3
2					
3				0	4
4					0
5					

1-3-5

动态规划：最优方案追踪

- 根据数组Rec，输出最短路径



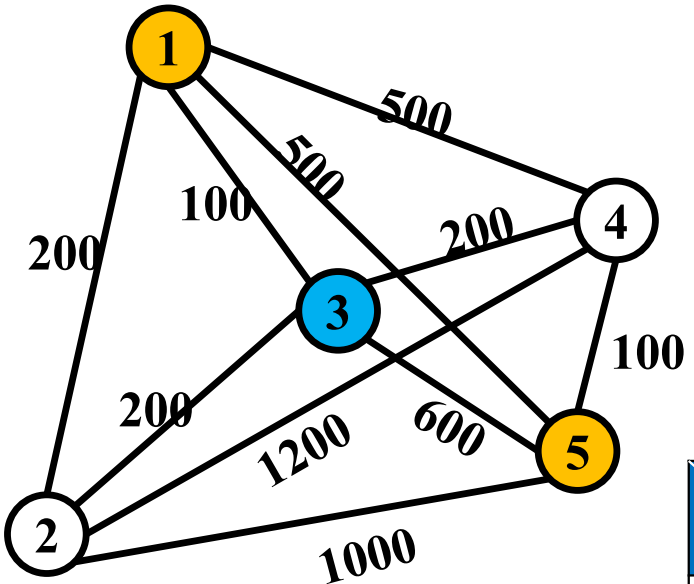
Rec

i \ j	1	2	3	4	5
1			0		3
2					
3				0	4
4					0
5					

1-3-5

动态规划：最优方案追踪

- 根据数组Rec，输出最短路径



Rec

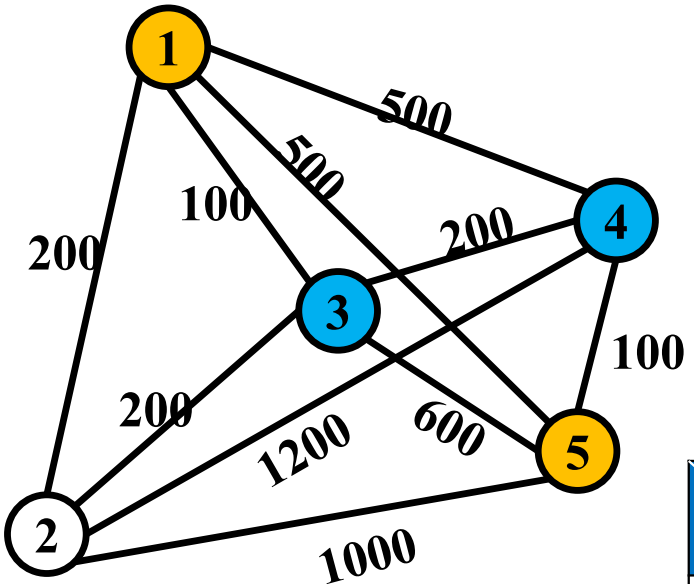
i \ j	1	2	3	4	5
1			0		3
2					
3				0	4
4					0
5					

1-3-5

1-3-4-5

动态规划：最优方案追踪

- 根据数组Rec，输出最短路径



Rec

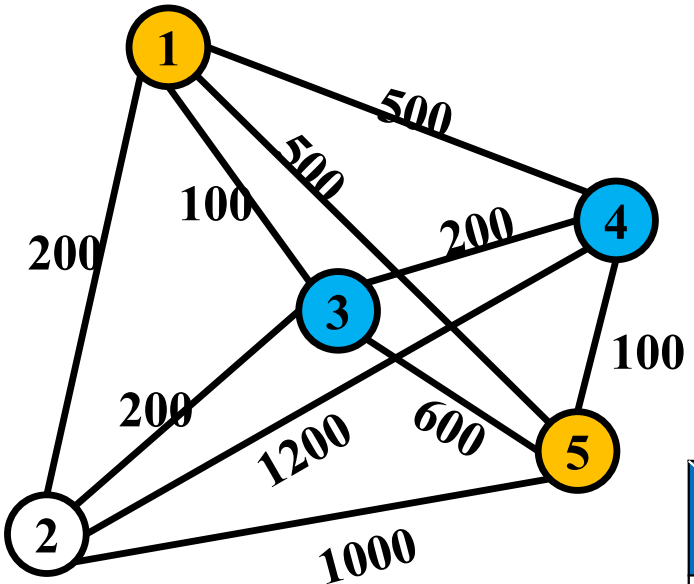
i \ j	1	2	3	4	5
1			0		3
2					
3				0	4
4					0
5					

1-3-5

1-3-4-5

动态规划：最优方案追踪

- 根据数组Rec，输出最短路径



Rec

i \ j	1	2	3	4	5
1			0		3
2					
3				0	4
4					0
5					

1-3-5

1-3-4-5

1 到 5 的最短路径

提纲

问题定义

算法思想

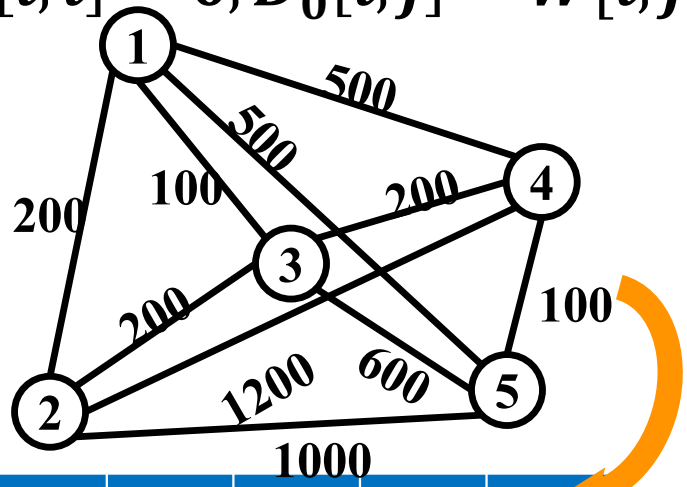
算法设计

算法实例

算法分析

算法实例

- $D_0[i,i]=0, D_0[i,j]=W[i,j]$



D

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	1200	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

$k=0$

Rec

所有点对都没有经过其他点

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

D

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	1200	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

$k = 1$

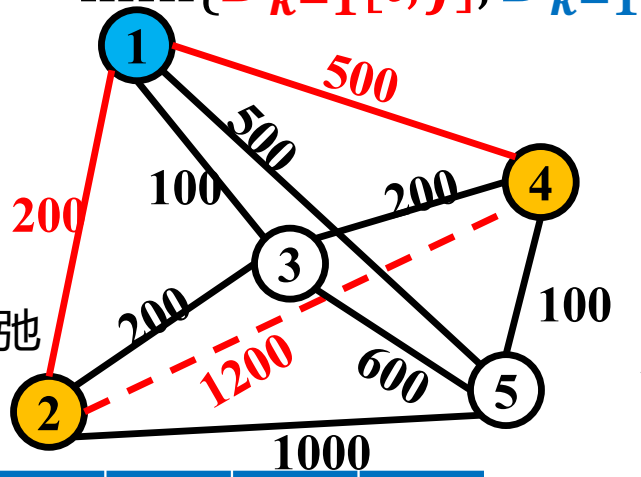
Rec

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

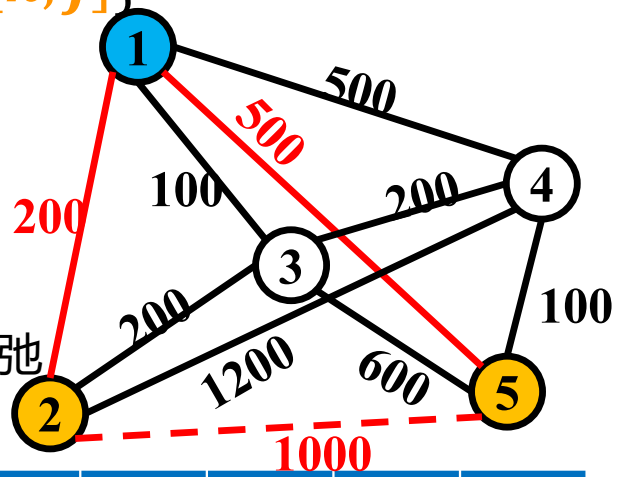
算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

从2到4路径得到松弛



从2到5路径得到松弛



D

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

$k = 1$

Rec

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

$k = 1$

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	1000	600	100	0

$k = 1$

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	700	600	100	0

$k = 1$

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	700	600	100	0

$k = 2$

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	700	600	100	0

$k = 2$

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	700	600	100	0

$k = 2$

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	700	600	100	0

$k = 2$

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

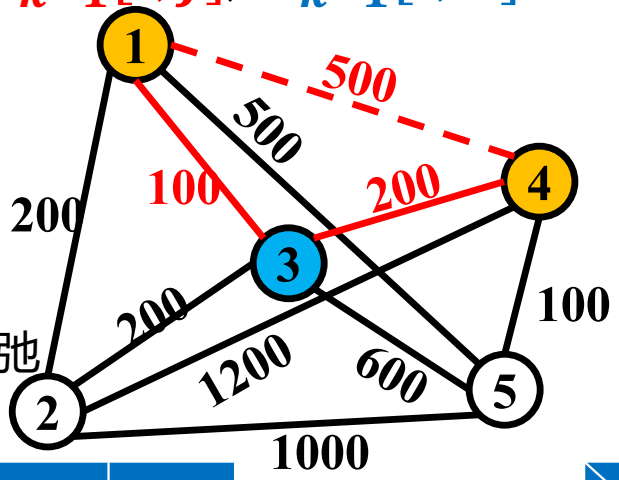
i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	700	600	100	0

$k = 2$

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$



从1到4路径得到松弛

i \ j	1	2	3	4	5
1	0	200	100	300	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	700	600	100	0

$k = 3$

i \ j	1	2	3	4	5
1	0	0	0	3	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	500
2	200	0	200	400	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	700	600	100	0

$k = 3$

i \ j	1	2	3	4	5
1	0	0	0	3	0
2	0	0	0	3	1
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	500
2	200	0	200	400	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	700	600	100	0

$k = 3$

i \ j	1	2	3	4	5
1	0	0	0	3	0
2	0	0	0	3	1
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	500
2	200	0	200	400	700
3	100	200	0	200	600
4	300	400	200	0	100
5	500	700	600	100	0

$k = 3$

i \ j	1	2	3	4	5
1	0	0	0	3	0
2	0	0	0	3	1
3	0	0	0	0	0
4	3	3	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	500
2	200	0	200	400	700
3	100	200	0	200	600
4	300	400	200	0	100
5	500	700	600	100	0

$k = 3$

i \ j	1	2	3	4	5
1	0	0	0	3	0
2	0	0	0	3	1
3	0	0	0	0	0
4	3	3	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	700
3	100	200	0	200	600
4	300	400	200	0	100
5	500	700	600	100	0

$k = 4$

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	1
3	0	0	0	0	0
4	3	3	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	600
4	300	400	200	0	100
5	500	700	600	100	0

$k = 4$

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	0
4	3	3	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	500	700	600	100	0

$k = 4$

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	500	700	600	100	0

$k = 4$

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

$k = 4$

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

$k = 5$

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

$k = 5$

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

$k = 5$

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

$k = 5$

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

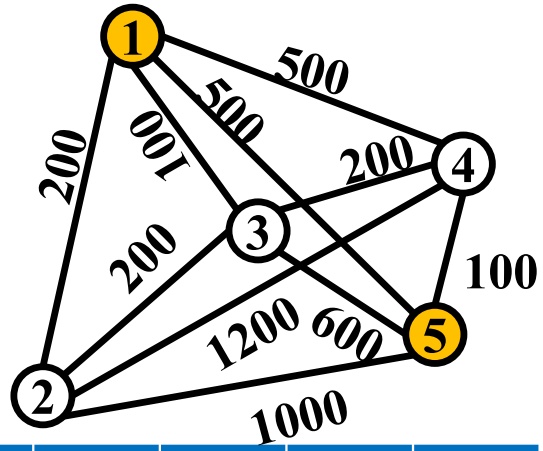
i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

$k = 5$

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
- 查询从1到5的最短路



D

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

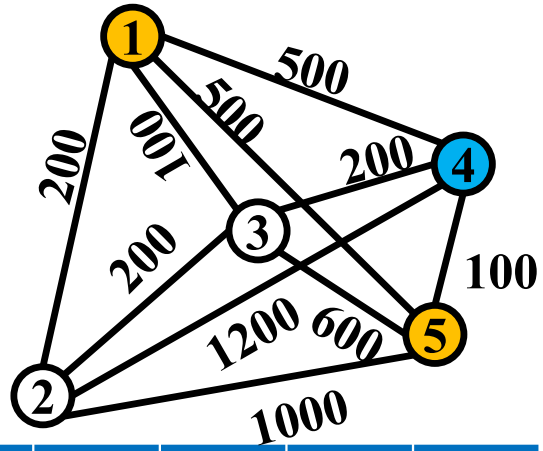
$k = 5$

Rec

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
- 查询从1到5的最短路



D

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

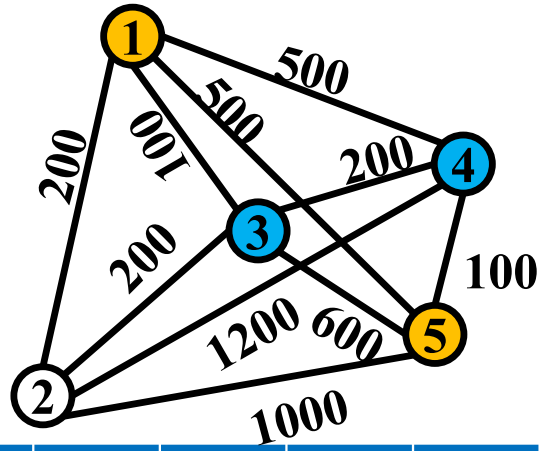
$k = 5$

Rec

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
- 查询从1到5的最短路



D

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

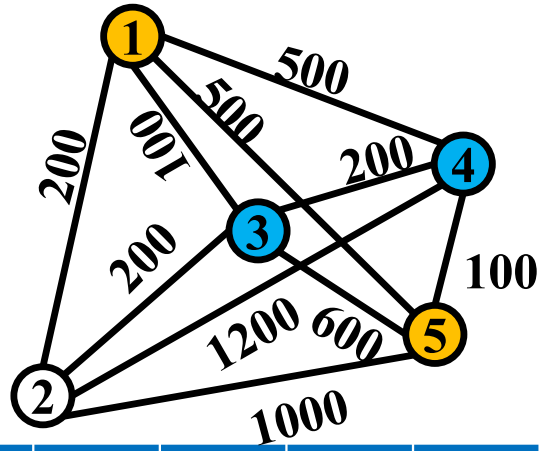
$k = 5$

Rec

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
- 查询从1到5的最短路



D

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

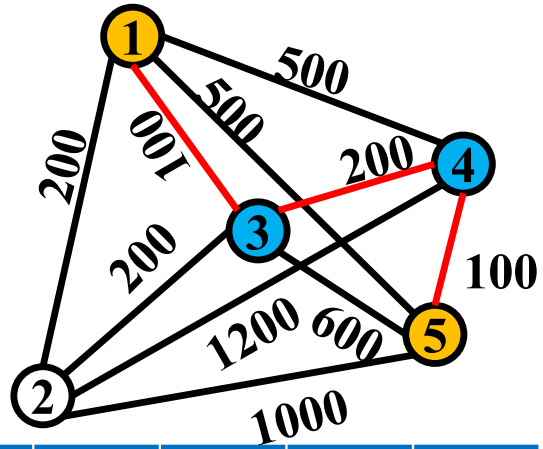
$k = 5$

Rec

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
- 查询从1到5的最短路



D

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

$k = 5$

Rec

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

提纲

问题定义

算法思想

算法设计

算法实例

算法分析

伪代码

- **All-Pairs-Shortest-Paths(G)**

```
输入: 图G =< V, E, W >
输出: 任意两点最短路径
新建二维数组D[1..|V|, 1..|V|], Rec[1..|V|, 1..|V|]
for i ← 1 to |V| do
    for j ← 1 to |V| do
        Rec[i, j] ← 0
        if i = j then
            D[i, j] ← 0
        end
        else
            D[i, j] ← W[i, j]
        end
    end
end
```

初始化

伪代码

- **All-Pairs-Shortest-Paths(G)**

输入: 图$G =< V, E, W >$
输出: 任意两点最短路径
新建二维数组$D[1..|V|, 1..|V|], Rec[1..|V|, 1..|V|]$
for $i \leftarrow 1$ *to* $|V|$ **do**
  for $j \leftarrow 1$ *to* $|V|$ **do**
    $Rec[i, j] \leftarrow 0$
    if $i = j$ **then**
      $D[i, j] \leftarrow 0$
    end
    else
      $D[i, j] \leftarrow W[i, j]$
    end
  end
end

起终点相同，距离为0

起终点不同，距离为边权

伪代码

- **All-Pairs-Shortest-Paths(*G*)**

```
for k ← 1 to |V| do
    for i ← 1 to |V| do
        for j ← 1 to |V| do
            if D[i, j] > D[i, k] + D[k, j] then
                D[i, j] ← D[i, k] + D[k, j]
                Rec[i, j] ← k
            end
        end
    end
end
return D, Rec
```

按照*k*增大的顺序

伪代码

- **All-Pairs-Shortest-Paths(*G*)**

```
for k ← 1 to |V| do
    for i ← 1 to |V| do
        for j ← 1 to |V| do
            if D[i,j] > D[i,k] + D[k,j] then
                D[i,j] ← D[i,k] + D[k,j]
                Rec[i,j] ← k
            end
        end
    end
end
return D, Rec
```

松弛操作

伪代码

- **Find-Path(Rec, u, v)**

```
输入: 备忘数组Rec,起点u,终点v
输出: 最短路径(逆序)
if Rec[u, v] = 0 then
    print v
    return
end
k ← Rec[u, v]
Find-Path(Rec, u, k)
Find-Path(Rec, k, v)
```

没有中间点，直接输出

伪代码

- **Find-Path(Rec, u, v)**

输入: 备忘数组Rec,起点u,终点v
输出: 最短路径(逆序)
if $Rec[u, v] = 0$ **then**
    print v
    return
end
$k \leftarrow Rec[u, v]$
Find-Path(Rec, u, k)
Find-Path(Rec, k, v)

有中间点，递归查找

时间复杂度

- **All-Pairs-Shortest-Paths(G)**

```
for k ← 1 to |V| do
    for i ← 1 to |V| do
        for j ← 1 to |V| do
            if D[i,j] > D[i,k] + D[k,j] then
                D[i,j] ← D[i,k] + D[k,j]
                Rec[i,j] ← k
            end
        end
    end
end
return D, Rec
```

$O(|V|)$ $O(|V|^2)$ $O(|V|^3)$

时间复杂度：$O(|V|^3)$

算法小结

- 该算法由Floyd和Warshall于1962年分别提出
- 也被称为Floyd-Warshall算法

Robert Floyd
1936-2001

Stephen Warshall
1935-2006

算法小结

- 直观思路：使用Dijkstra算法依次求解所有点

```
输入: 图G
输出: 任意两点最短路径
for i ← 1 to |V| do
    Paths[i] ← Dijkstra − PriQueue(G, i)
end
return Paths
```

$O(|E|\log|V|)$

$O(|V||E|\log|V|)$

回顾

```
//执行单源最短路径算法
while 优先队列Q非空 do
    v ← Q.ExtractMin()
    for u ∈ G.adj[v] do
        if dist[v] + w(v, u) < dist[u] then
            dist[u] ← dist[v] + w(v, u)
            pred[u] ← v
            Q.DecreaseKey((u, dist[u]))
        end
    end
    color[v] ← BLACK
end
```

时间复杂度$O(|E| \cdot \log|V|)$

算法小结

- 直观思路：使用**Dijkstra**算法依次求解所有点

```
输入: 图G
输出: 任意两点最短路径
for i ← 1 to |V| do
  |  Paths[i] ← Dijkstra − PriQueue(G, i)
end
return Paths
```

$O(|E|\log|V|)$ → $O(|V||E|\log|V|)$

- **Floyd-Warshall**算法时间复杂度：$O(|V|^3)$

算法小结

- 直观思路：使用Dijkstra算法依次求解所有点

```
输入: 图G
输出: 任意两点最短路径
for i ← 1 to |V| do
    Paths[i] ← Dijkstra − PriQueue(G, i)
end
return Paths
```

$O(|E|\log|V|)$ → $O(|V||E|\log|V|)$

针对稠密图 $|E| = O(|V|^2)$

$O(|V|^3\log|V|)$

- Floyd-Warshall算法时间复杂度：$O(|V|^3)$

算法小结

- 直观思路：使用Dijkstra算法依次求解所有点

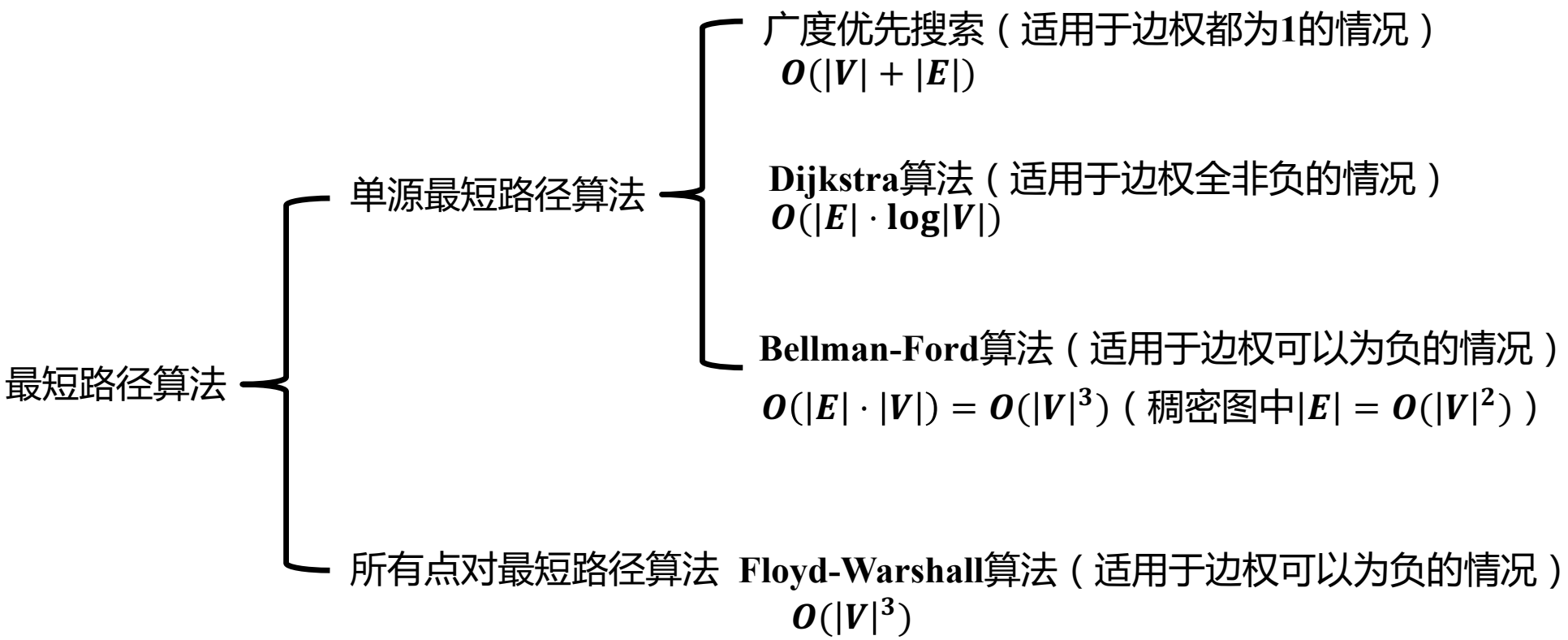
```
输入: 图G
输出: 任意两点最短路径
for i ← 1 to |V| do
    Paths[i] ← Dijkstra − PriQueue(G, i)
end
return Paths
```

$O(|E|\log|V|)$ → $O(|V||E|\log|V|)$

针对稠密图 $|E| = O(|V|^2)$

- Floyd-Warshall算法时间复杂度：$O(|V|^3)$ 优于 $O(|V|^3\log|V|)$

最短路径算法小结


最短路径算法
单源最短路径算法
广度优先搜索（适用于边权都为1的情况）
$O(|V|+|E|)$
Dijkstra算法（适用于边权全非负的情况）
$O(|E|\cdot \log|V|)$
Bellman-Ford算法（适用于边权可以为负的情况）
$O(|E|\cdot|V|)=O(|V|^3)$（稠密图中$|E|=O(|V|^2)$）
所有点对最短路径算法
Floyd-Warshall算法（适用于边权可以为负的情况）
$O(|V|^3)$

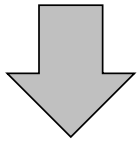
传递闭包

- 给定图 $G=(V,E)$，其中 $V=\{1,2,\ldots,n\}$。图 G 的传递闭包是一个图 $G^*=(V,E^*)$，其中

$$E^*=\{(i,j)|G\text{中有一条从 } i \text{ 到 } j \text{ 的路径}\}$$

- 直接使用Floyd-Warshall算法
- 对递推式进行转化

$$\boldsymbol{D_k[i,j]=\min\{D_{k-1}[i,j],D_{k-1}[i,k]+D_{k-1}[k,j]\}}$$



$$\boldsymbol{D_k[i,j]=D_{k-1}[i,j]\vee(D_{k-1}[i,k]\wedge D_{k-1}[k,j])}$$

有向无环图中的单源最短路径

- 有向无环图
  - 可以有负权边
  - 没有源点可达的负环
- 算法
  - 先求得图 G的拓扑排序
  - 按照拓扑排序的顶点顺序，对每个节点的关联边进行松驰