



HITWH
SE

第八章

图算法



8.1 网络流算法

8.2 单源最短路径问题



HITWH
SE

参考资料

Introduction to Algorithms

第24、26章



8.1 网络流算法

- 基本概念与问题定义
- Ford-Fulkerson方法

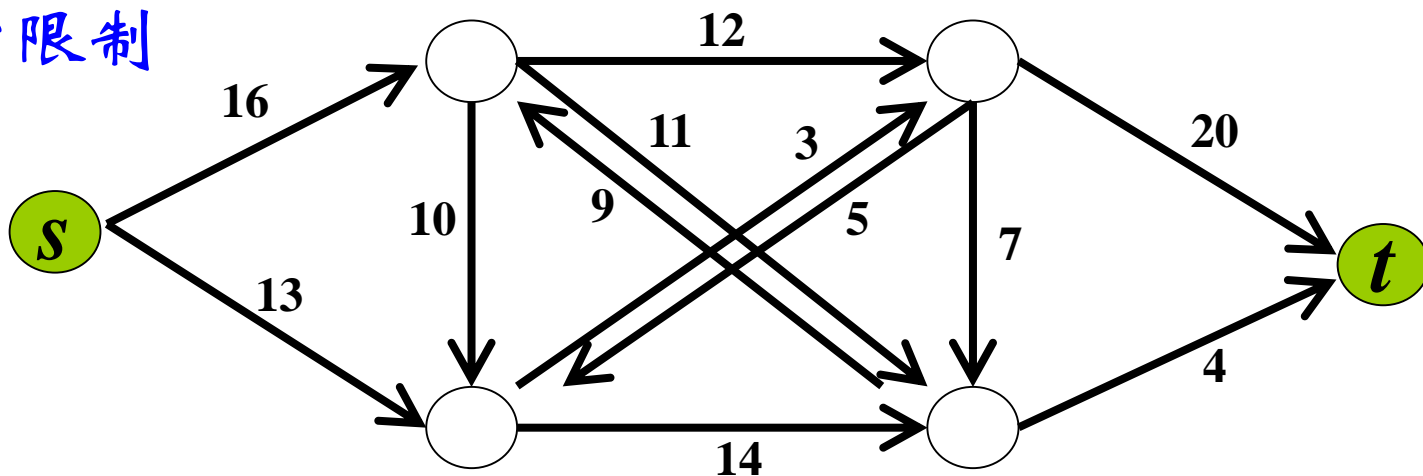


HITWH
SE

8.1.1 基本概念与问题定义



- 很多实际问题可以建模为流网络
 - 装配线上物件的流动
 - 电网中电流的流动
 - 通信网络中信息的流动
 - 道路交通网络中的运输活动
 -
- 一个源节点 s 、一个汇点 t ，由源节点流向汇点
 - 流量守恒
 - 容量限制





• 流网络

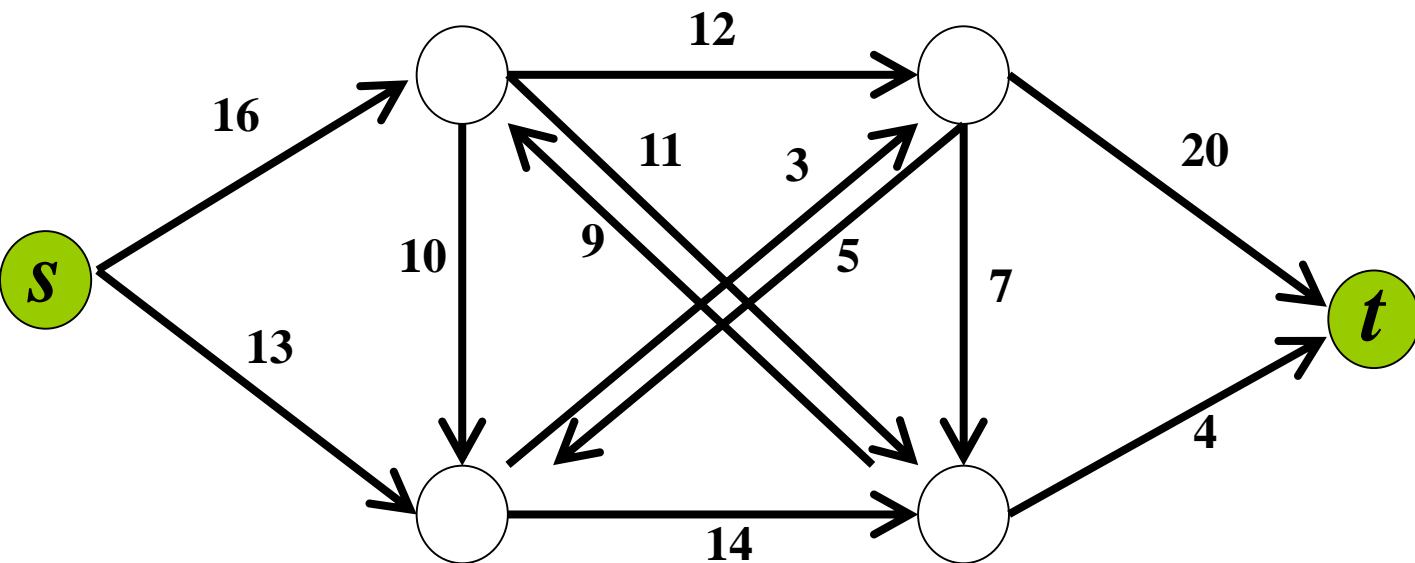
是一个无自环的有向图 $G=(V, E)$,

(1) 图中的每条边 $(u, v) \in E$ 有一个非负的容量值 $c(u, v) \geq 0$ 。

if $(u, v) \notin E$ $c(u, v) = 0$;

(2) 有两个特殊结点 $s, t \in V$, s 称为源结点(source), t 称为汇点(sink)

(3) For $\forall v \in V$, 存在一个 s 到 t 经过 v 的路径 $s \Rightarrow v \Rightarrow t$.



流网络是连通的

除源结点外, 每个结点
都至少有一条进入的边,
所以 $|E| \geq |V| - 1$



- 流(Flow)

设 $G(V, E)$ 是一个流网络, c 是容量函数, s 源结点, t 是汇点。

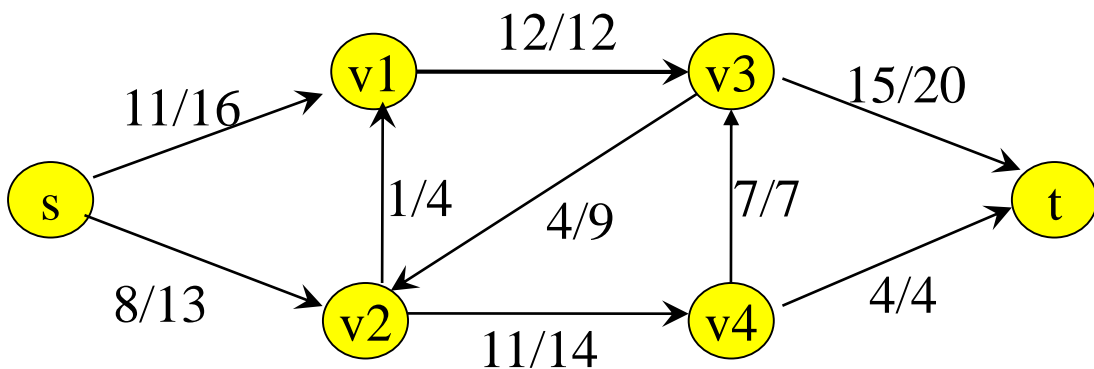
G 中的流是一个实值函数 $f: V \times V \rightarrow R$, 满足下列性质:

- (1) 容量约束: $\forall u, v \in V, 0 \leq f(u, v) \leq c(u, v)$;
- (2) 流量守恒: $\forall u \in V - \{s, t\}$, 有

$$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$$

称 $f(u, v)$ 为从结点 u 到结点 v 的流

当 $(u, v) \notin E$ 时, 从结点 u 到 v 之间没有流, 因此 $f(u, v) = 0$.

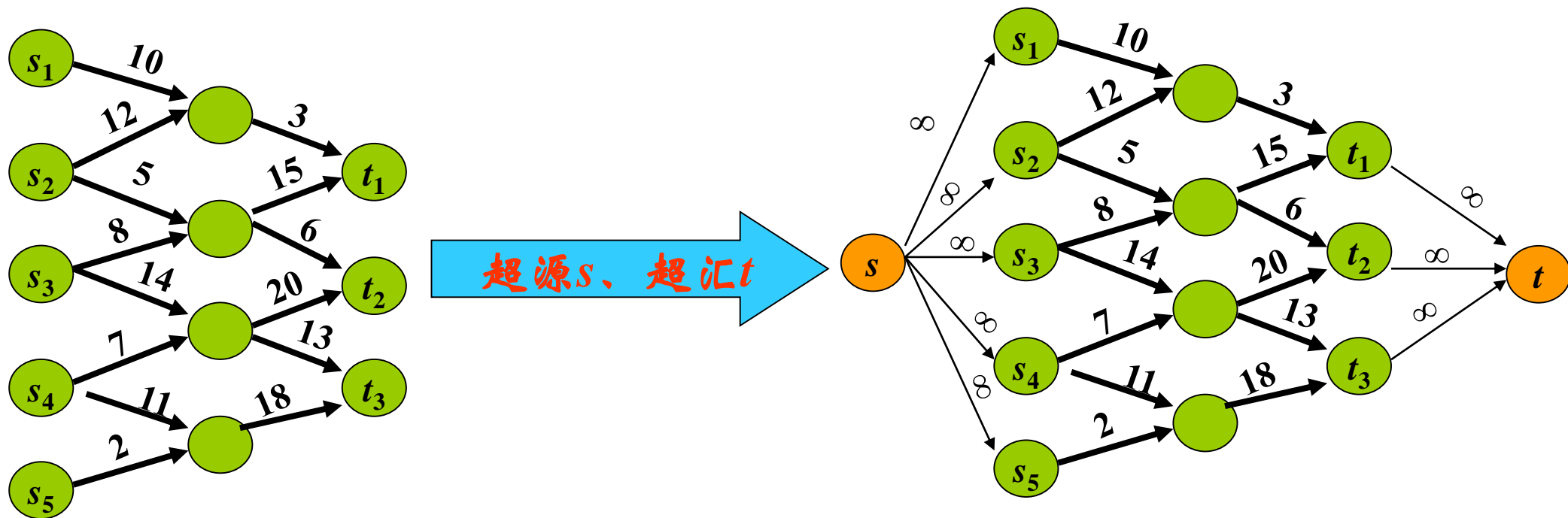


一个流 f 的值 $|f|$ 定义为:

$$\sum_{v \in V} f(s, v)$$



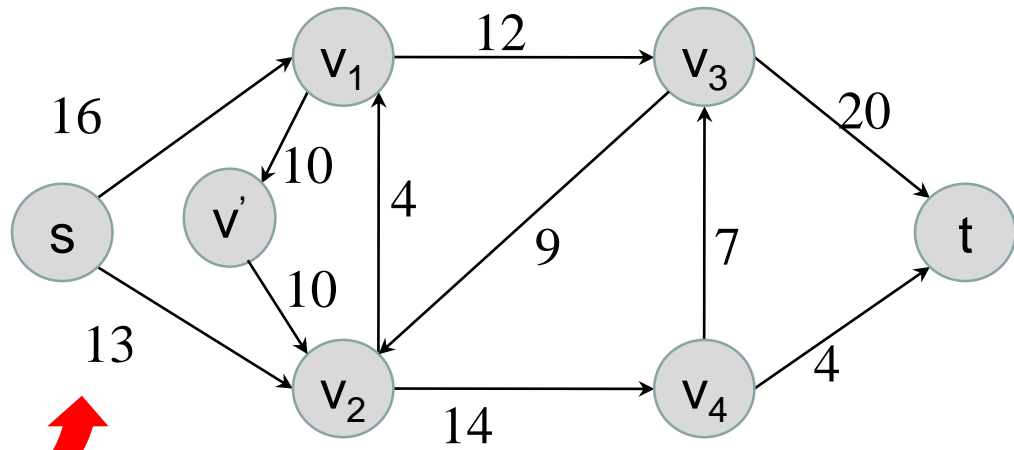
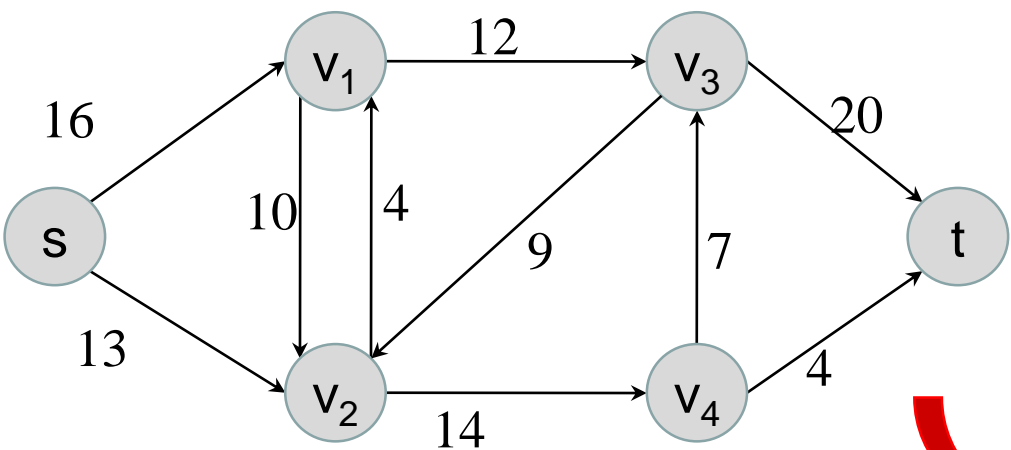
• 多源多汇的网络



只需讨论单源单汇的网络流



- 单源结点、单汇点流网络
- 假设：流网络中无反向边
 - 给定有向图 $G=(V, E)$ ，如果边 $(u, v) \in E$ ，则边 $(v, u) \notin E$





- 问题定义

- 输入: 流网络 $G=(V, E)$

- 输出: 具有最大流值的流 f





- 循环递进

- 初始：网络上的流为0
- 找出一条从 s 到 t 的路径 p 和正数 a ，使得 p 上的每一条边 (u,v) 的流量增加 a 之后仍能够满足容量约束： $f(u,v)+a \leq c(u,v)$
// 将 p 上的每条边的流量增加 a ，得到一个更大的流
- 重复执行第二步，直到找不到满足约束条件的路径。

关键在于：

1. 如何找路径 p ，以便得到更大的流？
2. 如何判断循环结束的条件？

即：当循环结束时，所得到的流一定是最大流么？

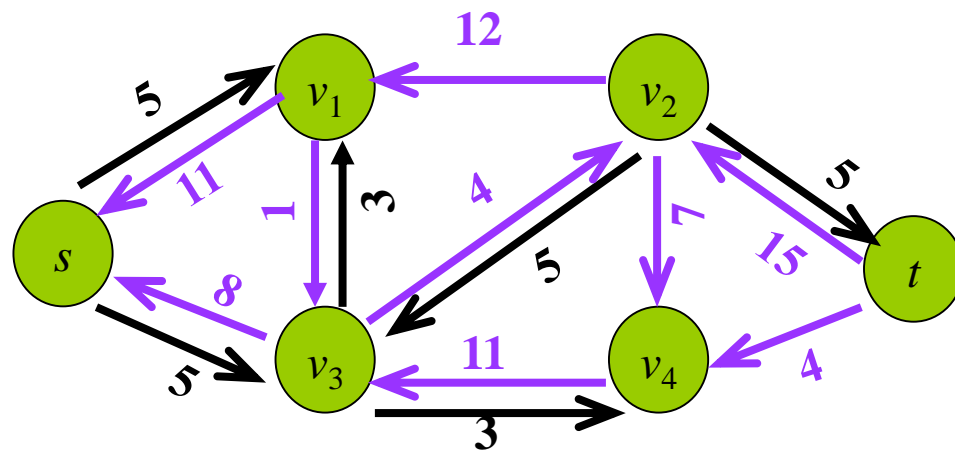
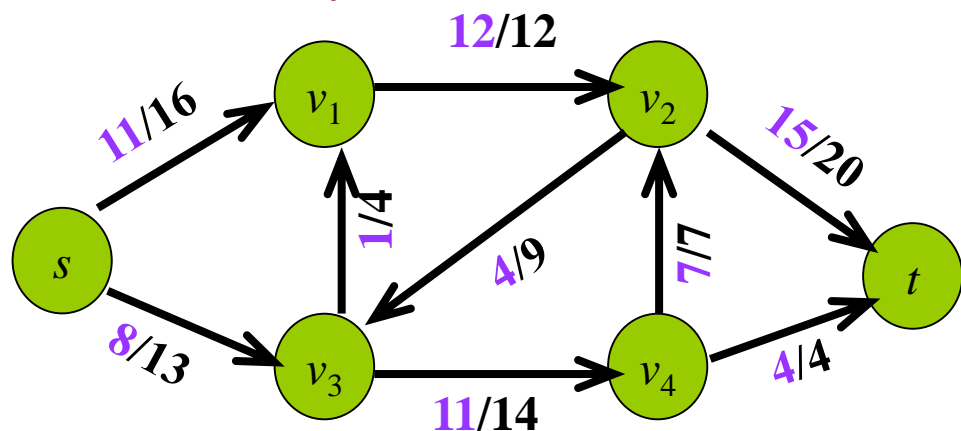


8.1.2 Ford-Fulkerson方法

- 如何找路径 p , 以便得到更大的流?
- 如何判断循环结束的条件?



- 在一个关联的**剩余网络**中寻找一条**增广路径**
- 剩余网络(Residual network)
 - 给定流网络 $G(V, E)$ 和一个流 f , 则由 f 诱导的 G 的剩余网络为 $G_f = (V, E_f)$, 其中 E_f 为
 - 对于 G 中每条边 (u, v) , 若 $c(u, v) - f(u, v) > 0$, 则 $(u, v) \in E_f$, 且 $c_f(u, v) = c(u, v) - f(u, v)$ (称 $c_f(u, v)$ 为剩余容量residual capacity)
 - 对于 G 中每条边 (u, v) , 若 $f(u, v) > 0$, 在 G_f 中构造边 (v, u) , 且 $c_f(v, u) = f(u, v)$



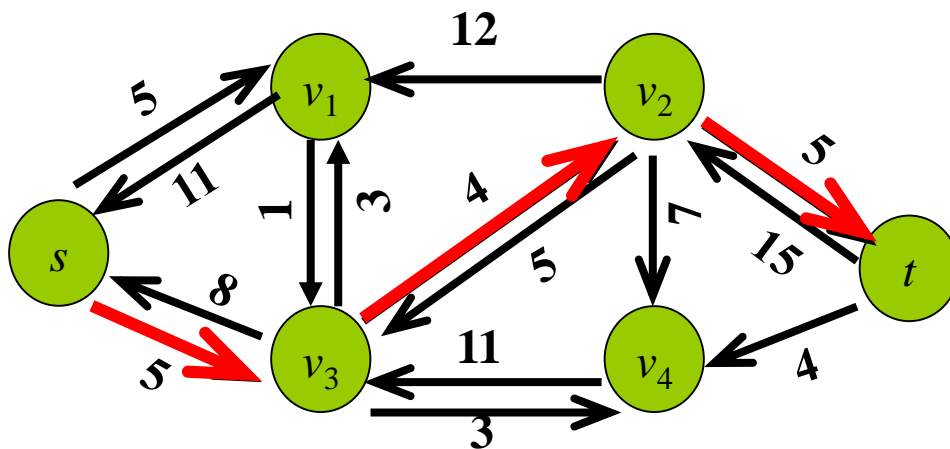
E_f 中的边要么是 E 中原有的边, 要么是其**反向边**, 因此 $|E_f| \leq 2|E|$



- 剩余网络类似于一个容量为 c_f 的流网络，但包含反向边
- 也可以把流网络 G 看成是一个当前流 $f=0$ 的剩余网络
- 可以在剩余网络中定义一个流



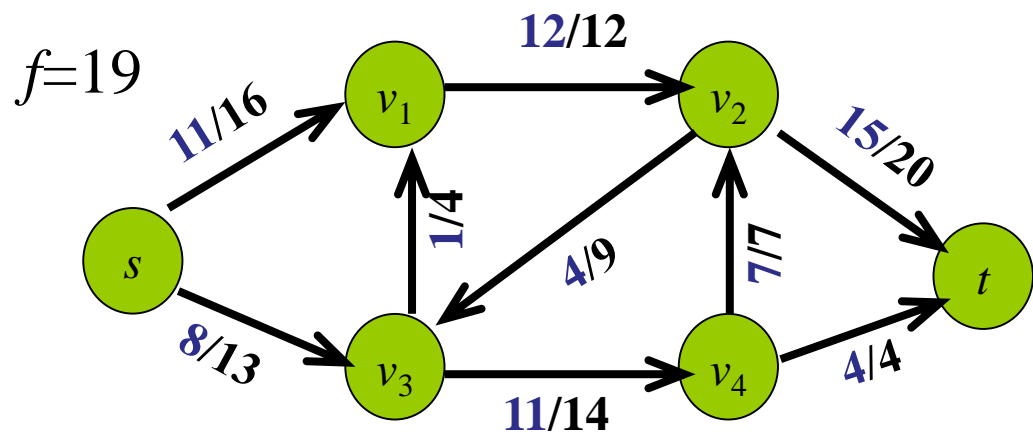
- 增广路径
 - 剩余网络中的一条由源结点 s 到汇点 t 的一条路径 p
- 增广路径 p 的剩余容量
 - $c_f(p) = \min\{c_f(u, v) : (u, v) \text{ 属于路径 } p\}$
 - 表示了该路径能够增加的流的最大值



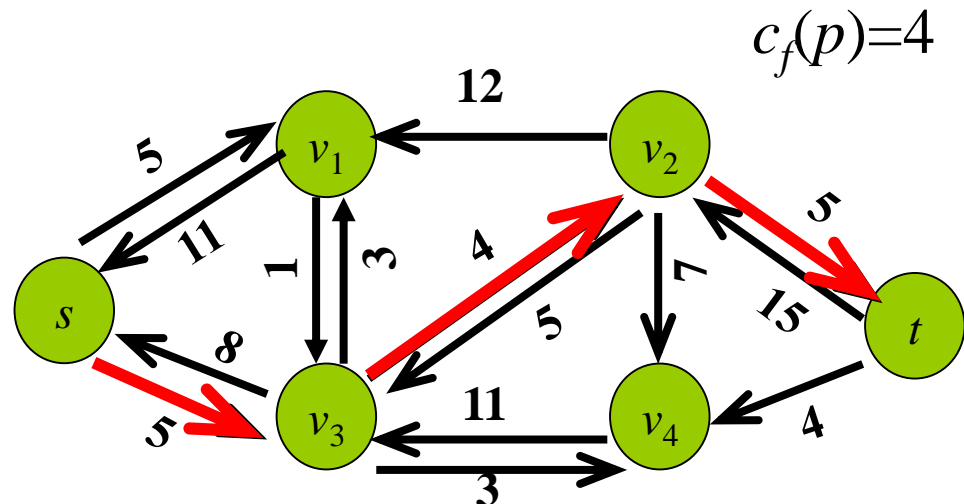
图中红色标注的路径为一条增广路径，其剩余容量为4



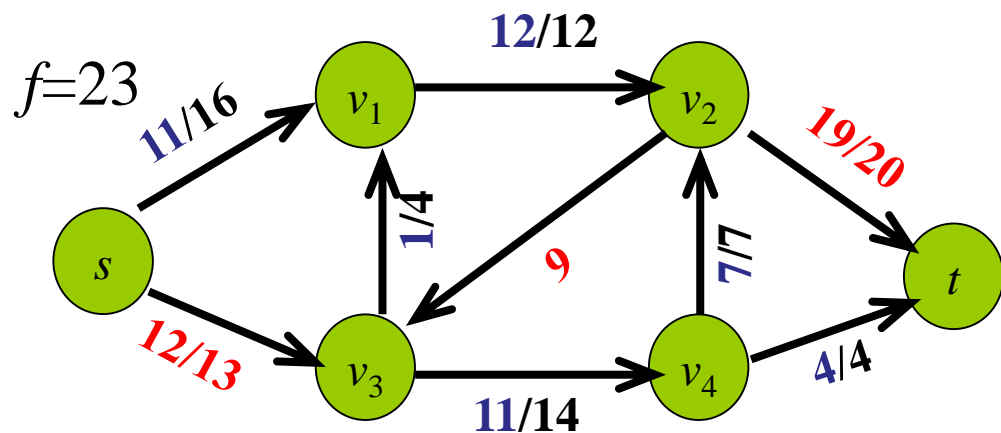
- 在剩余网络中寻找增广路径



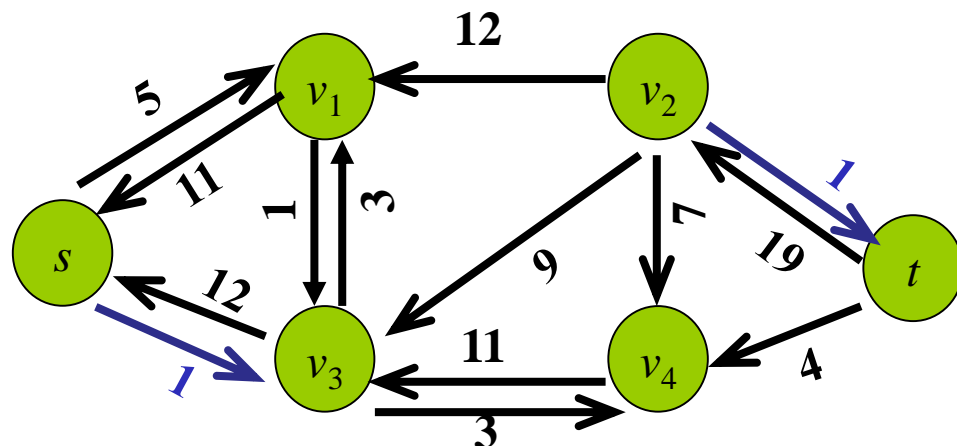
(a) 流网络 G 及流 f



(b) 由 (a) 诱导的剩余网络



(c) 由增广路径得到的更大流



(d) 由 (c) 诱导的剩余网络



- FF算法的核心是：通过增广路径不断增加路径上的流量，直到找到一个最大流为止

问题：

如何判断算法结束时，确实找到了一个最大流？

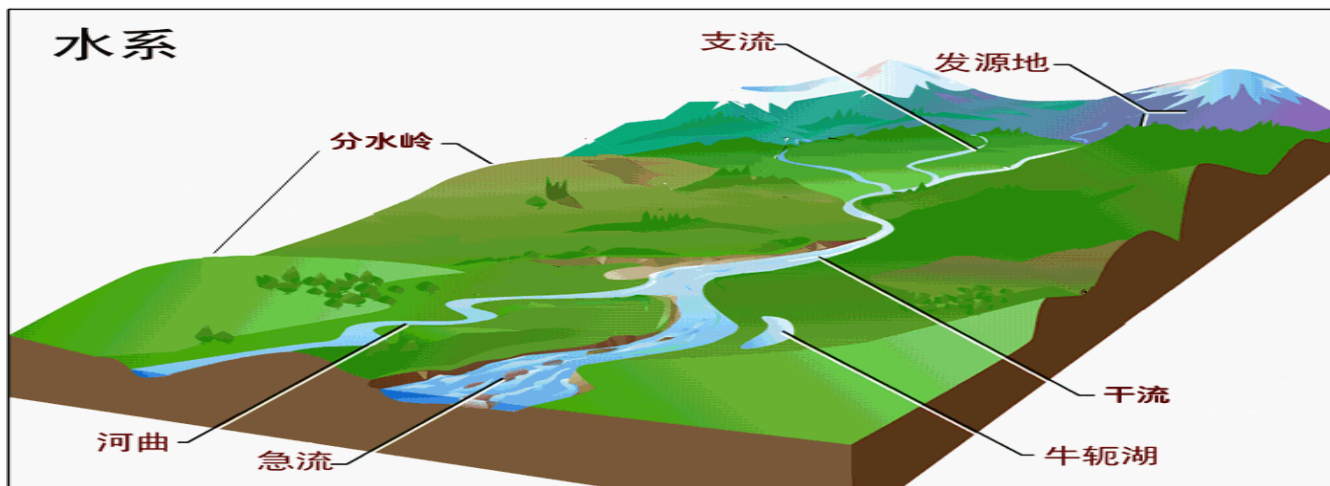


HITWH
SE

如何判断是否已获得最大流?

河水的**最大流量**取决于

干流中河道**狭窄处**的通行能力



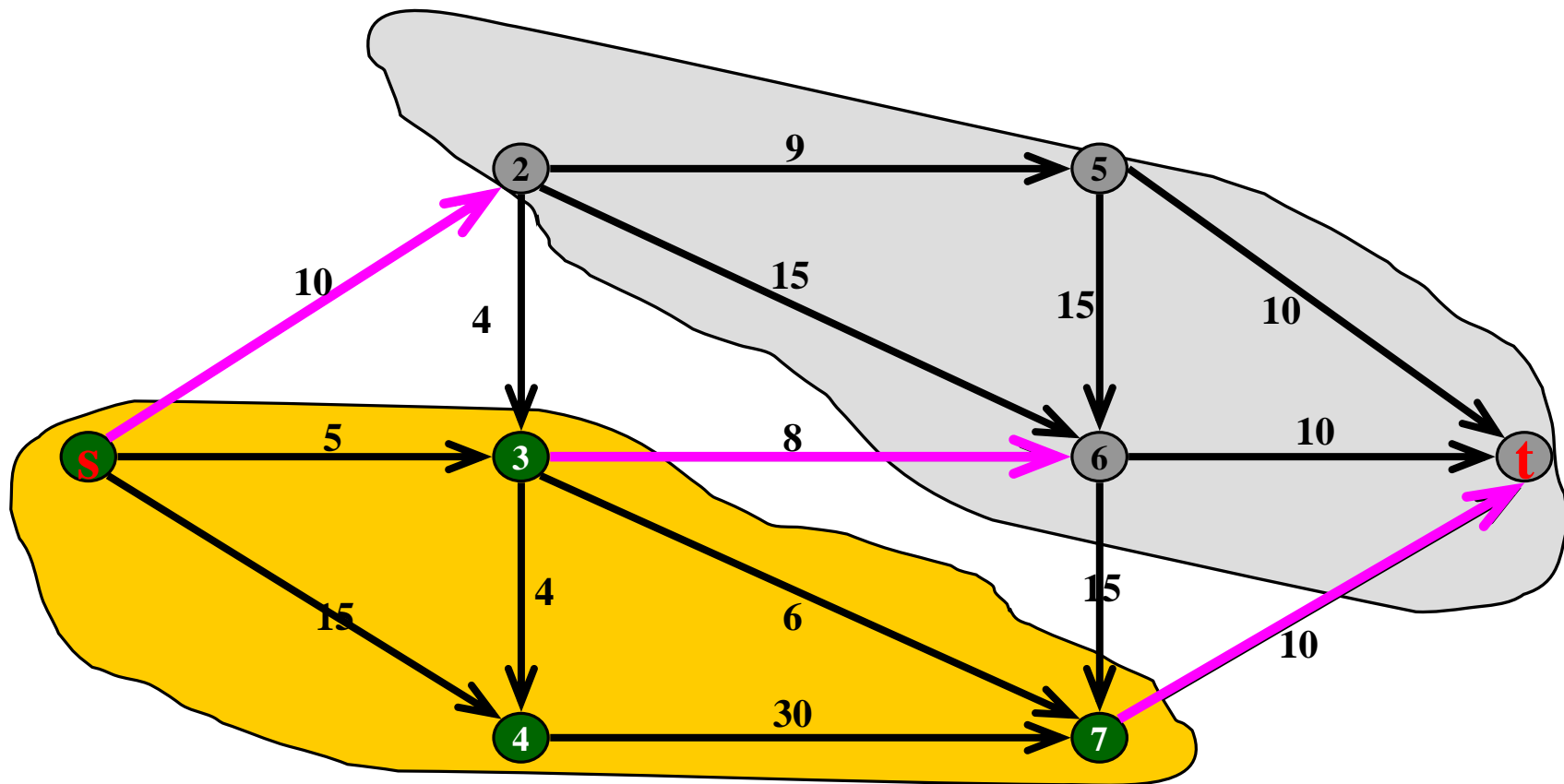
这种观察能否用于最大流问题呢?



HITWH
SE

如何判断是否已获得最大流?

从 s 流到 t 的最大流量不会超过 $10+8+10=28$





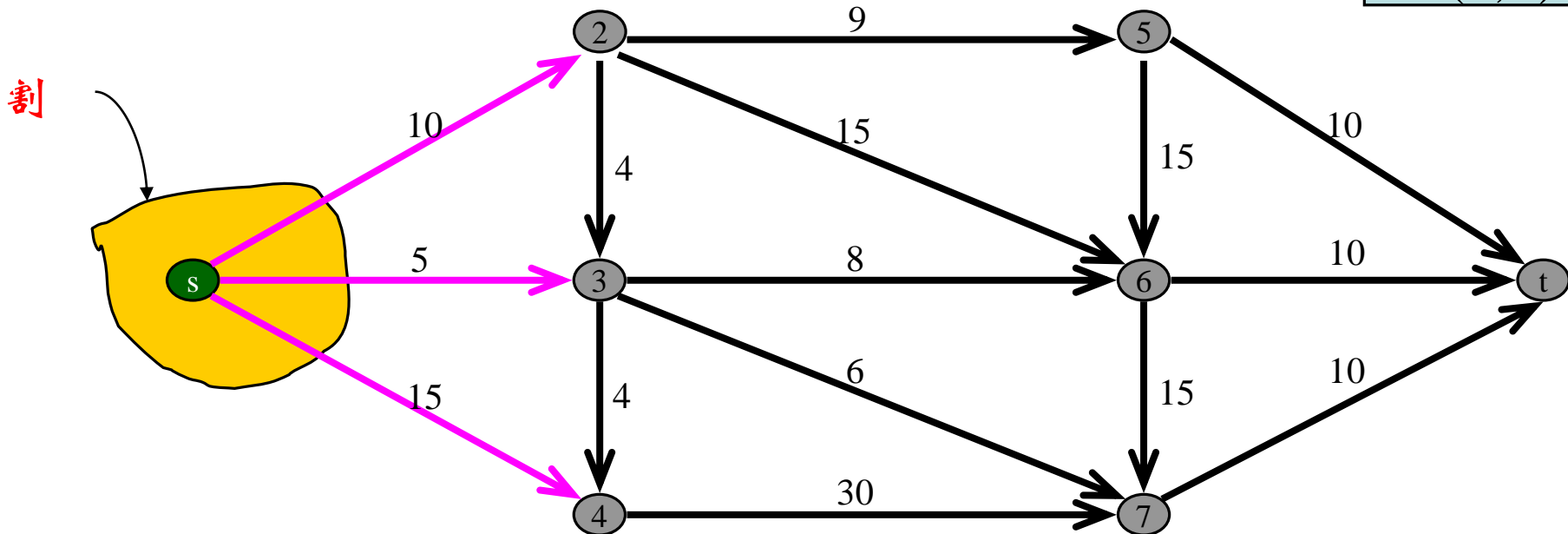
给定流网络 $G = (V, E)$, 其源为 s , 汇为 t ,

G 的一个割 (cut) 是 V 的 2-集合划分 (S, T) , $T = V - S$, 且 $s \in S$, $t \in T$

割的容量定义为

$$c(S, T) = \sum_{u \in S, v \in T} c(u, v)$$

$$c(S, T) = 30$$

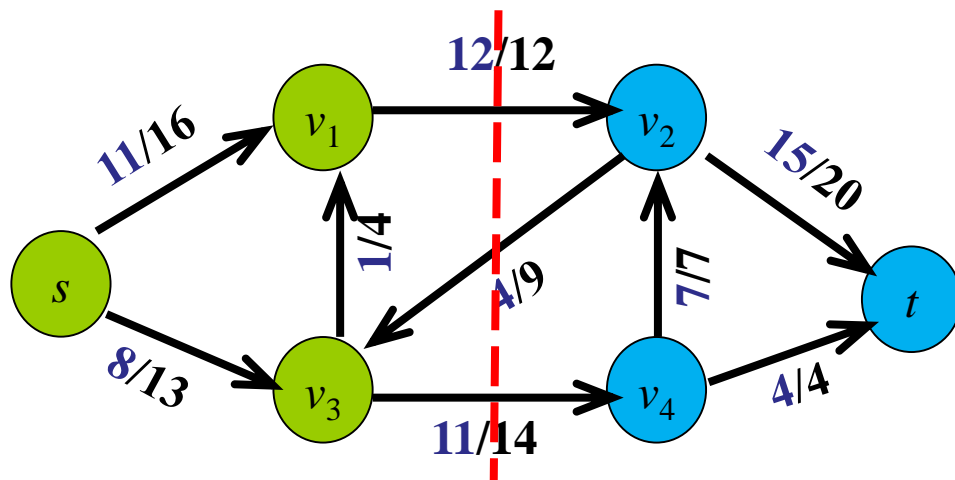




引理1. 设 f 为流网络 G 的一个流, 该流网络的源结点为 s , 汇点为 t , 设 (S, T) 为流网络 G 的任意一个割, 则横跨割 (S, T) 的净流量为 $|f|$.

横跨割 (S, T) 的净流量定义为:

$$f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u)$$



$f=19$

流网络 G 及流 f



推论1. 流网络 G 中任意流的值不能超过 G 的任意割的容量.

由引理知:

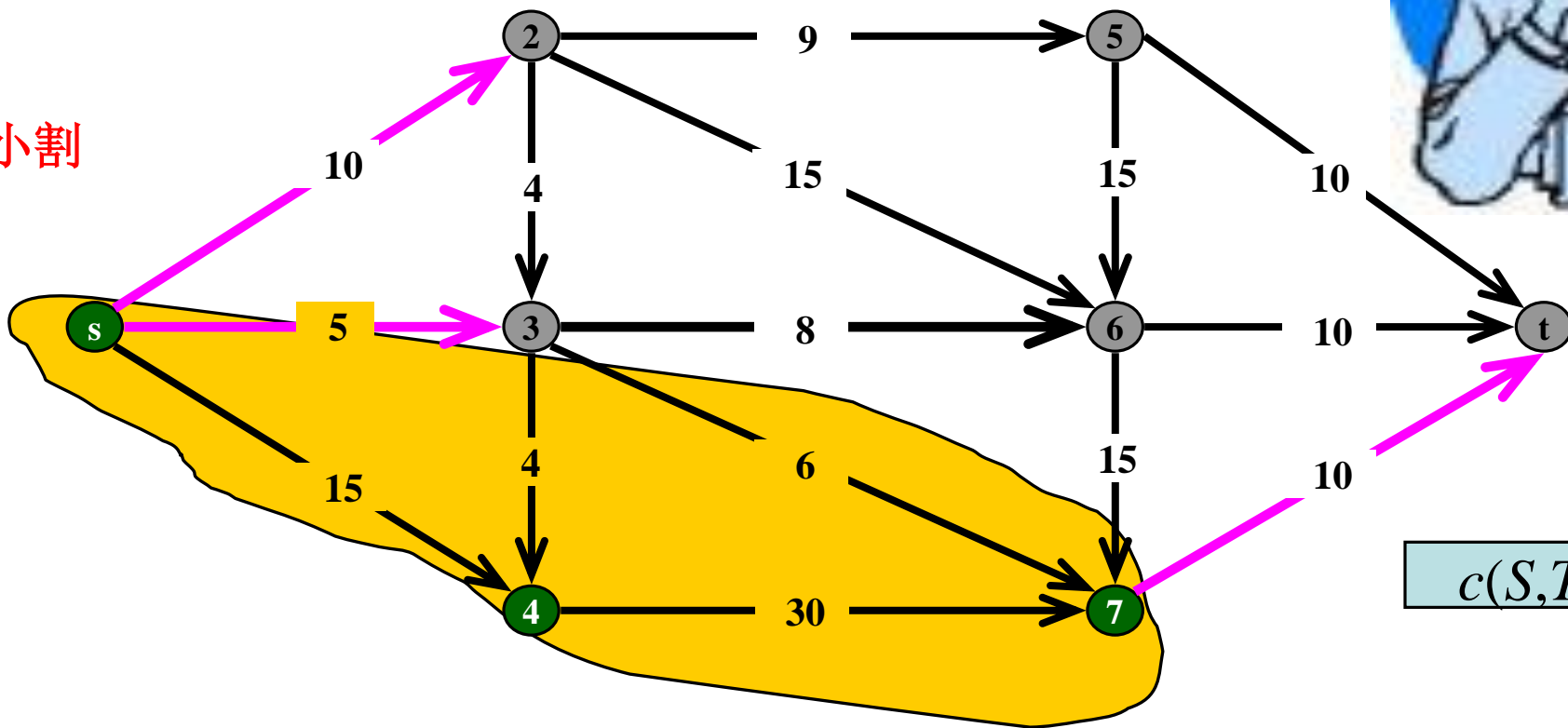
$$\begin{aligned} |f| &= f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u) \\ &\leq \sum_{u \in S} \sum_{v \in T} f(u, v) \\ &\leq \sum_{u \in S} \sum_{v \in T} c(u, v) = c(S, T) \end{aligned}$$



- 一个流网络的最小割
是指：整个网络中容量最小的割



最小割



$$c(S, T) = 25$$



最大流最小割定理:

设 f 为流网络 $G(V, E)$ 一个流, 该流网络的源结点为 s , 汇点为 t , 则下面命题等价:

1. f 是 G 的最大流.
2. 剩余网络 G_f 不包含增广路径.
3. 对于 G 的某个划分 (S, T) , $|f| = c(S, T)$.

一个最大流的值实际上等于一个最小割的容量

Max-Min关系: 对偶关系

最大流与最小割

最大匹配与最小覆盖

.....



- 对同一问题从不同角度考虑，有两种对立的描述
— 例如，平面中矩形面积与周长的关系

正方形： 周长一定，面积最大的矩形

面积一定，周长最小的矩形

Max问题

Min问题



HITW
SE

利用Max-Min关系求解最大流问题

1. 初始化一个可行流 f

— 0-流：所有边的流量均等于0的流

2. 不断将 f 增大，直到 f 不能继续增大为止

3. 找出一个割 (S, T) 使得 $|f| = c(S, T)$

— 由此断言 f 是最大流，而 (S, T) 是最小割

Max-Min关系提供了高效求解最大流-最小割问题的机制！



算法 Ford-Fulkerson(G, s, t)

Input 流网络 G , 源 s , 汇 t

Output G 中从 s 到 t 的最大流

1. For $\forall (u, v) \in E[G]$ do
2. $f(u, v) \leftarrow 0$
3. $f(v, u) \leftarrow 0$
4. While G_f 存在增广路径 p do
5. $c_f(p) = \min\{c_f(u, v) \mid (u, v) \text{ 是 } p \text{ 上的边}\}$
6. For p 上的每条边 (u, v) do
7. If (u, v) 是流网络中的边 Then
8. $f(u, v) \leftarrow f(u, v) + c_f(p)$
9. Else
10. $f(v, u) \leftarrow f(v, u) - c_f(p)$



Ford-Fulkerson 算法分析

- 正确性分析

1. 算法输出的一定是最大流

- 由最大流-最小割定理可得

2. 算法可终止性

- 假设整数容量，每次流量增加1



Ford-Fulkerson 算法分析

算法 Ford-Fulkerson(G, s, t)

Input 流网络 G , 源 s , 汇 t

Output G 中从 s 到 t 的最大流

1. For $\forall (u, v) \in E[G]$ do
2. $f(u, v) \leftarrow 0$
3. $f(v, u) \leftarrow 0$
4. While G_f 存在增广路径 p do
5. $c_f(p) = \min\{c_f(u, v) \mid (u, v) \text{ 是 } p \text{ 上的边}\}$
6. For p 上的每条边 (u, v) do
7. If (u, v) 是流网络中的边 Then
8. $f(u, v) \leftarrow f(u, v) + c_f(p)$
9. Else
10. $f(v, u) \leftarrow f(v, u) - c_f(p)$

1-3步: $O(E)$

4-8步: 循环次数最多为 $|f^*|$

第4步在 G_f 中找路径
(深度或宽度优先)
代价 $O(E)$

总的复杂度 $O(|f^*|E)$

如何改进 Ford-Fulkerson 算法?



加速增广路径的寻找

- 最短增广路径算法：始终沿着 G_f 中具有最少边的路径进行增广
- 最大增广路径算法：始终沿着 G_f 中具有最大容量的路径进行增广



- 利用宽度优先在剩余网络 G_f 中寻找增广路径
 - 从源结点 s 到汇点 t 的一条最短路径
 - 每条边的权重为单位距离
 - $\delta_f(u, v)$ = 剩余网络 G_f 中从结点 u 到结点 v 的最短路径距离



引理2: 如果Edmonds-Karp算法运行在流网络 $G(V, E)$ 上, 该网络的源结点为 s , 汇点为 t , 则对于所有的结点 $v \in V - \{s, t\}$, 剩余网络 G_f 中的最短路径距离 $\delta_f(s, v)$ 随着每次流量的递增而单调递增。

证明: 反证法, 假设存在一个 $v \in V - \{s, t\}$, 使得 $\delta_f(s, v)$ 减小。

令 f 是第一次发生最小距离减小之前的流, f' 是第一次发生最小距离减小之后的流。

令 v 为 f 增长为 f' 的过程中, 在最小距离减小的顶点中具有最小 $\delta_{f'}(s, v)$, $\delta_{f'}(s, v) < \delta_f(s, v)$ 。

令路径 $p: s \rightsquigarrow u \rightarrow v$ 是 $G_{f'}$ 中从 s 到 v 的最短路径, 最后一条边为 (u, v) 。

于是 $\delta_{f'}(s, v) = \delta_{f'}(s, u) + 1$, 而且 $\delta_f(s, u) \leq \delta_{f'}(s, u)$. (v 的选择方式)

考查边 (u, v) 是否在 G_f 中, 如果在:

$\delta_f(s, v) \leq \delta_f(s, u) + 1 \leq \delta_{f'}(s, u) + 1 = \delta_{f'}(s, v)$. 矛盾!



Edmonds-Karp 算法

证明：反证法，假设存在一个 $v \in V - \{s, t\}$ ，使得 $\delta_f(s, v)$ 减小。令 f 是第一次发生最小距离减小之前的流， f' 是第一次发生最小距离减小之后的流。令 v 为 f 增长为 f' 的过程中，在最小距离减小的顶点中具有最小 $\delta_{f'}(s, v)$ ， $\delta_{f'}(s, v) < \delta_f(s, v)$ 。

令路径 $p: s \rightsquigarrow u \rightarrow v$ 是 G_f 中从 s 到 v 的最短路径，最后一条边为 (u, v) 。于是有 $\delta_{f'}(s, v) = \delta_{f'}(s, u) + 1$ ，而且 $\delta_f(s, u) \leq \delta_{f'}(s, u)$ 。考查边 (u, v) 是否在 G_f 中，如果在： $\delta_f(s, v) \leq \delta_f(s, u) + 1 \leq \delta_{f'}(s, u) + 1 = \delta_{f'}(s, v)$ 。矛盾！

若不在，则边 (u, v) 的出现必然因为 f' 在 f 中增大了 (v, u) 上的流。

Edmonds-Karp 算法总是在最短路径上增大流，因此 (v, u) 必为 G_f 中从 s 到 u 的某条最短路径的最后一条边。

于是有 $\delta_f(s, v) = \delta_f(s, u) - 1 \leq \delta_{f'}(s, u) - 1 = \delta_{f'}(s, v) - 2$ 。

再次矛盾！

因此不存在一个 $v \in V - \{s, t\}$ ，使得 $\delta_f(s, v)$ 减小，引理2成立。



Edmonds-Karp算法运行时间: $O(VE^2)$

定理1: 如果Edmonds-Karp算法运行在流网络 $G(V,E)$ 上, 该网络的源结点为 s , 汇点为 t , 则算法所执行的流量递增操作的总次数为 $O(VE)$ 。

证明要点: 考查增广路径上剩余容量最小的边 (u,v) ——关键边。

$\delta_f(s,v) = \delta_f(s,u) + 1$, 流 f 增大后关键边 (u,v) 将消失。

(u,v) 再次出现时, 是因为某次增大流 f' 时边 (v,u) 出现在增广路径上。

于是有 $\delta_{f'}(s,u) = \delta_{f'}(s,v) + 1$. 同时, $\delta_{f'}(s,v) \geq \delta_f(s,v)$ (引理2).

因此 $\delta_{f'}(s,u) - \delta_f(s,u) \geq 2$.

边 (u,v) 成为关键边, 而后消失再出现, s 到 u 的最短距离至少增长2.

s 到 u 的最短距离最大为 $|V| - 1$, 因此每条边 (u,v) 成为关键边的次数不超过 $(|V|+1)/2$. 共有 $|E|$ 条边, 所有边成为关键边次数之和不超过 $(|V|+1)|E|/2$.

流量递增操作次数不大于所有边成为关键边次数之和(一次递增可能有多个关键边). 因此流量递增次数为 $O(VE)$.



8.2 单源最短路径问题

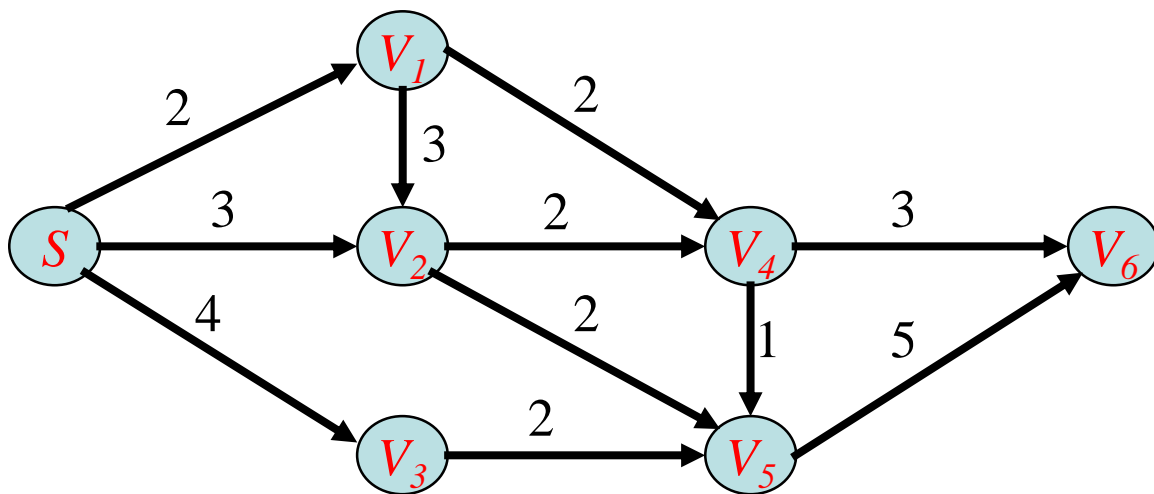
- 单源最短路径问题
- Bellman-Ford 算法
- DAG 中的算法
- Dijkstra 算法



- 问题定义

输入：给定一个带权的有向连通图 $G(V, E)$ 和权重函数 $W:E \rightarrow \{\text{实数}\}$ ，源点 s

输出：对于 $\forall v \in V$ ，由 s 到 v 的权值最小路径，即最短路径

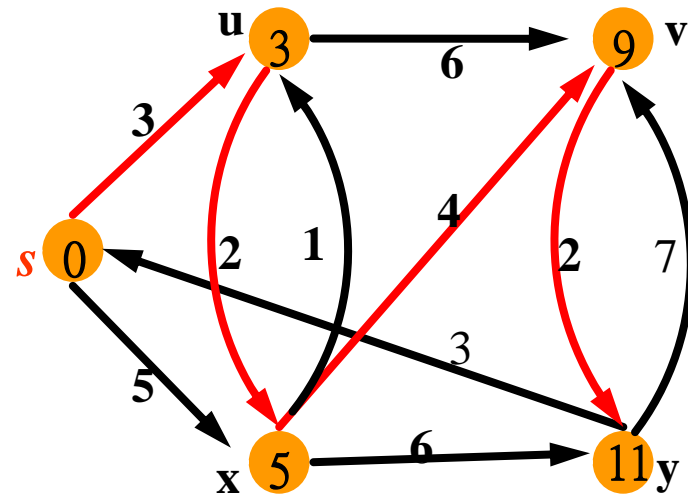
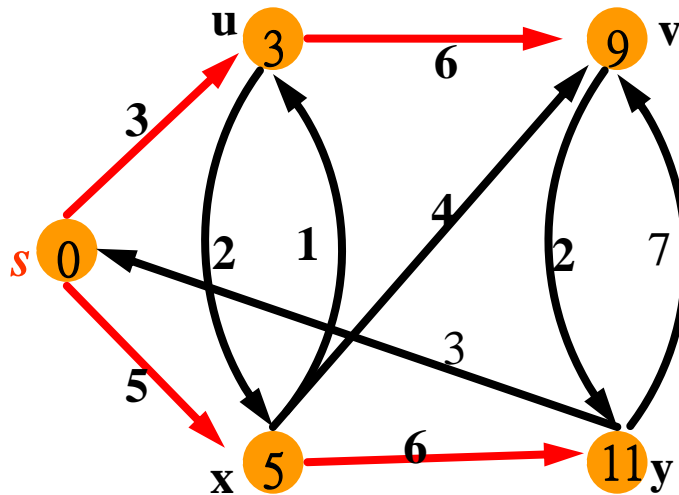
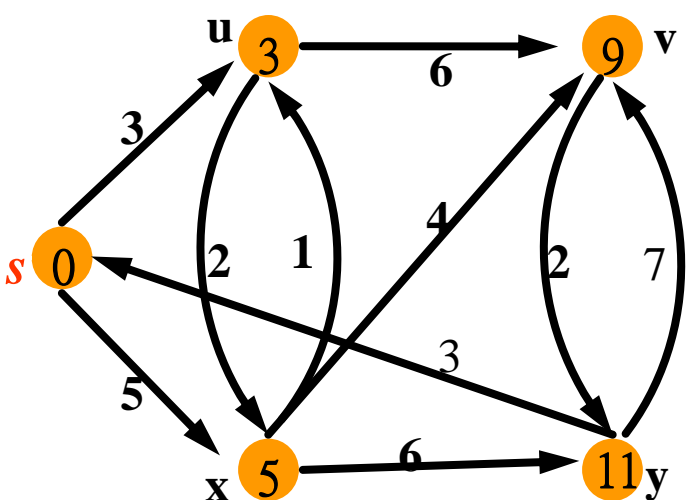


可以应用在很多的领域：路由选择、社交网络、智慧交通等



- 最短路径树

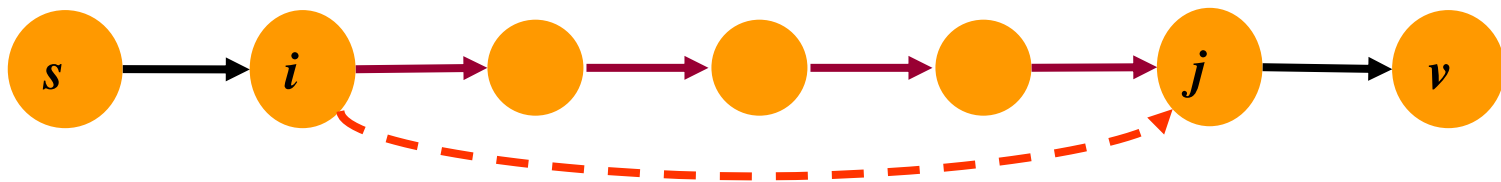
- 一棵有根结点的树，包括了从源结点 s 到每个可以从 s 到达的结点的一条最短路径。
- 最短路径不是唯一的，最短路径树也不是唯一的





- 优化子结构：最短路径包含最短子路径

设 s 到 v 的最短路径 P 经过顶点 i 和 j ，则 P 上从 i 到 j 的部分必然是 i 到 j 的最短路径



证明：如果某条子路径不是最短子路径
则必然存在最短子路径
用最短子路径替换当前子路径
当前路径不是最短路径
矛盾！

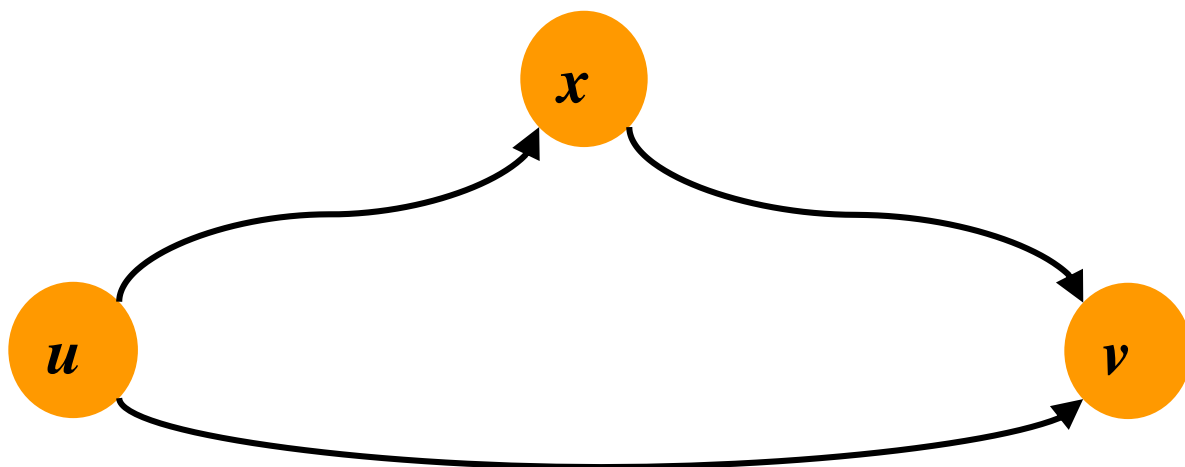
说明：该问题可以用贪心或动态规划方法来解



- 最短路径权重满足三角不等式性质

— 令 $\delta(u, v)$ 表示从 u 到 v 的最短路径的权重

$$\delta(u, v) \leq \delta(u, x) + \delta(x, v)$$

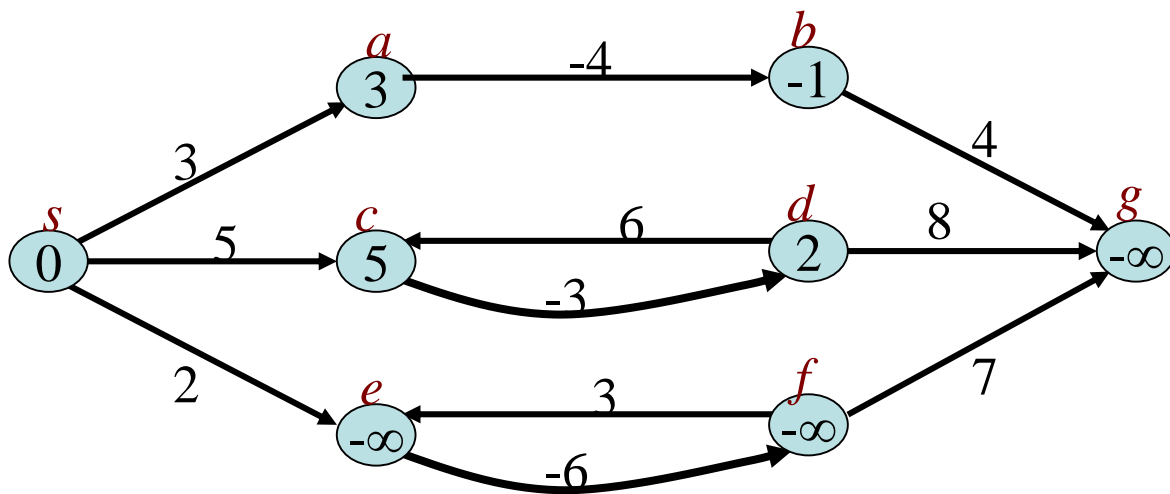


最短路径不比任何其他路径长



- 边权值为负值的问题

- 在某些问题实例中，某些边的权值可能为负值
- 如果 $G = (V, E)$ 不包含可由 s 到达的负权值环，则对于任意 $v \in V$ ， s 到 v 的最短路径权重是有精确定义的。
- 如果 $G = (V, E)$ 包含可由 s 到达的负权值环，则对于任意 $v \in V$ ， s 到 v 的最短路径权重无定义。





- 环问题(最短路径不能包含非0环)
 - 一个最短路径如果包含负值环, 它的权值只能 $-\infty$
 - 一个最短路径如果包含正值环, 去掉这个环后得到一个更短的路径, 于是它不是最短路径
 - 一个最短路径可以包含0环, 去掉该环后, 路径权值相同
 - 我们假定: 最短路径不包含环.
 - 无环路径至多包括 $|V|$ 个节点, 即至多包含 $|V|-1$ 条边
 - 我们只关心边数至多为 $|V|-1$ 的最短路径



- 松弛(Relaxation)

- 令 $\delta(s,v)$ 表示从 s 到 v 的最短路径的权值
- 对所有 v , 维护 $\delta(s,v)$ 的一个上界 $d[v]$ (对 $\delta(s,v)$ 的一个估计)

算法 $\text{Relax}(u,v,w)$

Input 顶点 u 和 v , 图的加权函数 w

Output 松弛后的 $d[v]$

1. if $(d[v] > d[u] + w(u,v))$ then
2. $d[v] = d[u] + w(u,v);$
3. $\pi[v] = u;$ // 将 v 的前驱结点置为 u



- 针对一般情况下的单源最短路径问题

— 边的权值可以为负

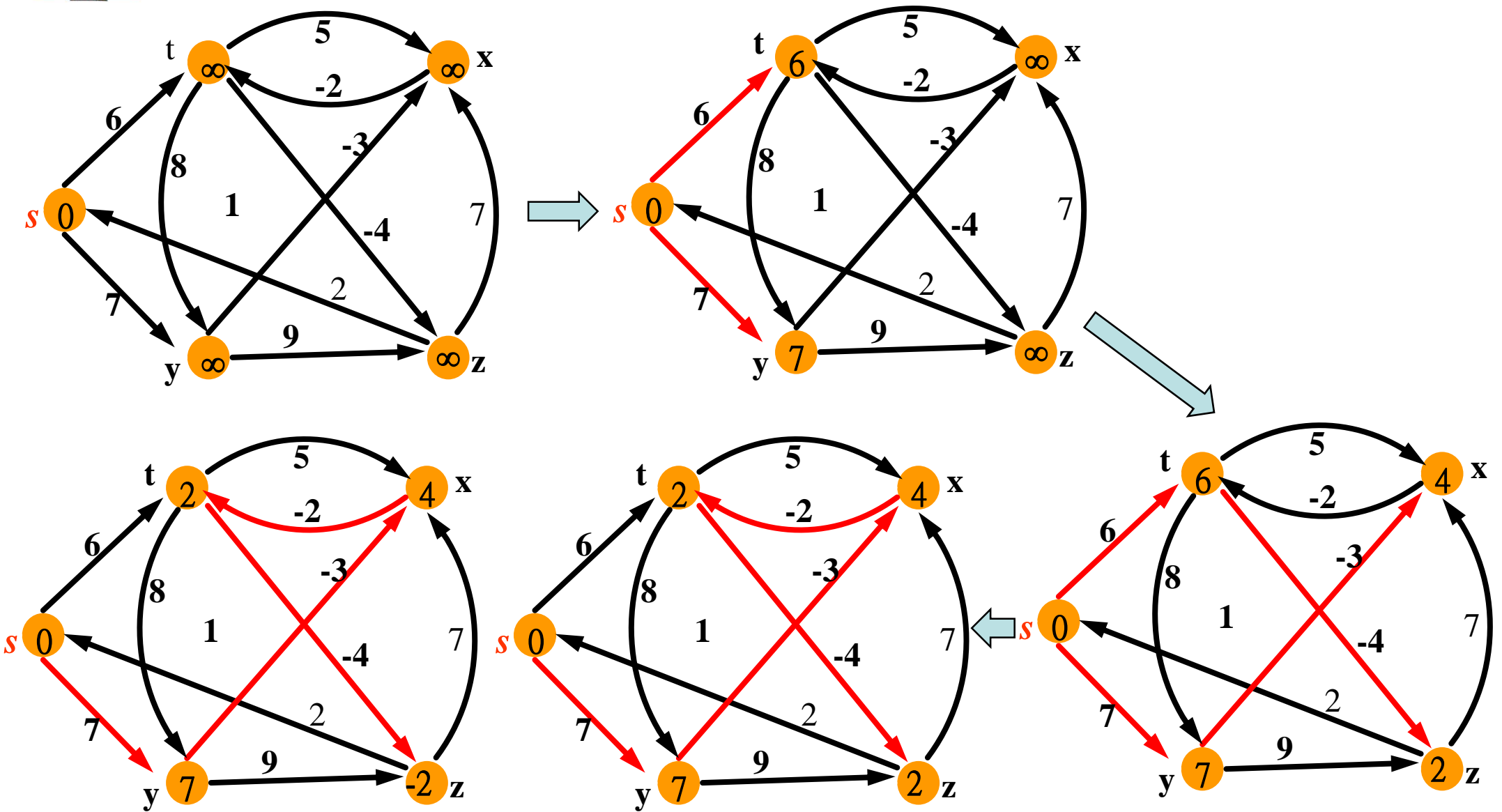
输入：带权有向图 $G(V, E)$ ，及权重函数 $W: E \rightarrow R$ ，源结点 s

输出：是否存在一个从 s 可以到达的权值为负值的环路，

若不存在环路，则给出对应的最短路径及权重



Bellman-Ford算法主要思想：每轮对所有边进行一次松弛
 $(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$.





算法 Bellman-Ford(G, w, s)

Input 图 $G=(V, E)$, 边加权函数 w , 源顶点 s

Output s 到其所有可达顶点的最短路径

1. For $\forall v \in V$ do
2. $d[v] \leftarrow \infty$;
3. $\pi[v] \leftarrow \text{null}$;
4. $d[s] \leftarrow 0$;
5. For $i \leftarrow 1$ to $|V|-1$ do
6. For $\forall (u, v) \in E$ do
7. Relax(u, v, w);
8. For $\forall (u, v) \in E$ do
9. If $d[v] > d[u] + w(u, v)$ then
10. return FALSE
11. Return True

初始化

求解

执行 $|V|-1$ 遍，松弛每条边

检查解的合理性，是否有负环

Relax(u, v, w): if ($d[v] > d[u] + w(u, v)$) then $d[v] = d[u] + w(u, v)$



Bellman-Ford 算法分析



算法 Bellman-Ford(G, w, s)

Input 图 $G=(V, E)$, 边加权函数 w , 源顶点 s

Output s 到其所有可达顶点的最短路径

时间复杂性?

$O(VE)$

1. For $\forall v \in V$ do
2. $d[v] \leftarrow \infty$;
3. $\pi[v] \leftarrow \text{null}$;
4. $d[s] \leftarrow 0$;
5. For $i \leftarrow 1$ to $|V|-1$ do
6. For $\forall (u, v) \in E$ do
7. Relax(u, v, w);
8. For $\forall (u, v) \in E$ do
9. If $d[v] > d[u] + w(u, v)$ then
10. return FALSE
11. Return True

为什么算法运行 $|V|-1$ 遍就足够了?

Relax(u, v, w): if ($d[v] > d[u] + w(u, v)$) then $d[v] = d[u] + w(u, v)$



- 正确性证明

引理1. $G(V, E)$ 为一个带权的源结点为 s 的有向图，其权值函数为 $W: E \rightarrow R$ 。

假设图 G 不包含从源结点 s 可以到达的权值为负值的环路。那么在算法的第2-4行for循环执行了 $|V|-1$ 次之后，对于所有从源结点 s 可以到达的结点 v ，都有 $d(v) = \delta(s, v)$ 。

证明：设 $P = (v_0, \dots, v_k)$ 是从 s 到 v 的最短路径，其中 $v_0 = s$ 且 $v_k = v$ 。

由于 P 是简单路径，故 $k < |V|-1$ 。

最初， $d[v_0] = d[s] = 0 = \delta(s, s)$ 且以后不再变化。

一遍之后， $d[v_1] = \delta(s, v_1)$ 是 s 到 v_1 的最短路径，且 $d[v_1]$ 以后不再变化。

设 $k-1$ 边后有 $d[v_{k-1}] = \delta(s, v_{k-1})$ ，则第 k 遍循环过程中，

如果 $(d[v_k] > d[v_{k-1}] + w(u, v))$ ，则 $d[v_k] = d[v_{k-1}] + w(u, v)$

$d[v_k] = \delta(s, v_k)$ ，因为每条最短 $(k-1)$ -路径均会被松弛过程检查和扩展。

$d[v] = \delta(s, v)$ 在 $|V|-1$ 遍后成立。



• 正确性证明

定理. $G(V, E)$ 为一个带权的源结点为 s 的有向图, 其权值函数为 $W: E \rightarrow R$.

- (1). 如果图 G 不包含从源结点 s 可以到达的权值为负值的环路, 那么算法返回 True 值, 且对于所有结点 $v \in V$, $d[v] = \delta(s, v)$ 成立;
- (2). 如果图 G 包含一条从源结点 s 可以到达的权值为负值的环路, 算法返回 FALSE.

证明: (1). 若 G 中没有负环.

$|V|-1$ 遍后, 我们有 $d[v] = \delta(s, v)$ 对任意顶点 v 成立.

由三角不等式,

$d[v] = \delta(s, v) \leq \delta(s, u) + w(u, v) = d[u] + w(u, v)$, 对任意 $(u, v) \in E$ 成立.



• 正确性证明

定理. $G(V, E)$ 为一个带权的源结点为 s 的有向图, 其权值函数为 $W: E \rightarrow R$.

- (1). 如果图 G 不包含从源结点 s 可以到达的权值为负值的环路, 那么算法返回 True 值, 且对于所有结点 $v \in V$, $d[v] = \delta(s, v)$ 成立;
- (2). 如果图 G 包含一条从源结点 s 可以到达的权值为负值的环路, 算法返回 FALSE.

证明: (2). 反证法. 设 G 中从 s 可以到达负环 (v_0, v_1, \dots, v_k) 其中 $v_k = v_0$.

$$\sum_{i=1}^k w(v_{i-1}, v_i) < 0$$

但, 算法返回 TRUE, 也就是说: 没有负环被检测到, 且检测负环 (v_0, v_1, \dots, v_k) 上的任意一条边时, 均有

$$d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i) \quad \text{for } i = 1, 2, \dots, k. \quad \text{三角不等式}$$

$$\sum_{i=1}^k d[v_i] \leq \sum_{i=1}^k d[v_{i-1}] + \sum_{i=1}^k w(v_{i-1}, v_i).$$

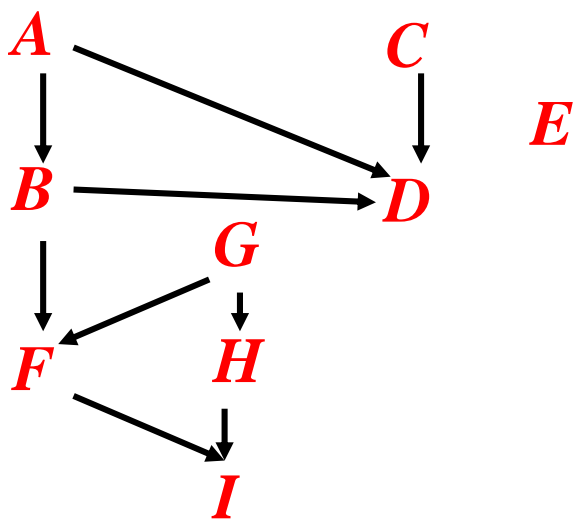
$$\text{Since } \sum_{i=1}^k d[v_i] = \sum_{i=1}^k d[v_{i-1}], \quad \sum_{i=1}^k w(v_{i-1}, v_i) \geq 0 \quad \text{矛盾!}$$



- 在有向无环图(Directed Acyclic Graph- DAG)中如何高效地解决单源最短路径问题
 - Bellman-Ford 算法的时间复杂度是 $O(VE)$.
 - 在DAG中能否更快?
 - 分析: Bellman-Ford算法执行 $|V|-1$ 遍
 - 每遍均需扫描所有边一遍
 - 对许多边的扫描均是无用的
 - 事实上,
 - 无需扫描不影响结果的边
 - 对于已经找到的最短路径, 其上的边无需再扫描



- Main Idea: 利用拓扑排序
 - DAG中每条路径均是拓扑序顶点序列的子序列
 - 能够容易识别从s可达的顶点，避免无用边扫描
 - 按照拓扑序处理顶点，将始终是前向地处理每条路径，避免重复扫描已知最短路径上的边
 - 仅需要一遍扫描





算法 DAG-Shortest-Paths(G, w, s)

Input 无环有向图 $G=(V, E)$, 边加权函数 w , 源顶点 s

Output s 到其所有可达顶点的最短路径

1. 将 V 中顶点进行拓扑排序
2. For $\forall v \in V$ do
3. $d[v] \leftarrow \infty$;
4. $\pi[v] \leftarrow \text{null}$;
5. $d[s] \leftarrow 0$;
6. For each $u \in V$ (按拓扑序考虑) do
7. For $\forall v \in \text{Adj}[u]$ do
8. Relax(u, v, w);

作业：分析该算法的正确性和性能



- Dijkstra 算法假设 $w(u, v) \geq 0$ 对 $\forall (u, v) \in E$ 成立
- 始终维护顶点集 S 使得
 - $\forall v \in S, d[v] = \delta(s, v)$, 即 s 到 v 的最短路径已经找到.
 - 初始值: $S = \emptyset, d[s] = 0$ 且 $d[v] = +\infty$
- 算法运行过程中
 - (a) 选择 $u \in V - S$ 使得
$$d[u] = \min \{d[x] \mid x \in V - S\}.$$
 令 $S = S \cup \{u\}$

此时 $d[u] = \delta(s, u)$! 为什么?
 - (b) 对于 u 的每个相邻顶点 v 执行 $\text{RELAX}(u, v, w)$
- 重复上述步骤(a)和 (b) 直到 $S = V$.
- 该算法类似于 Prim 算法, 属于贪心算法

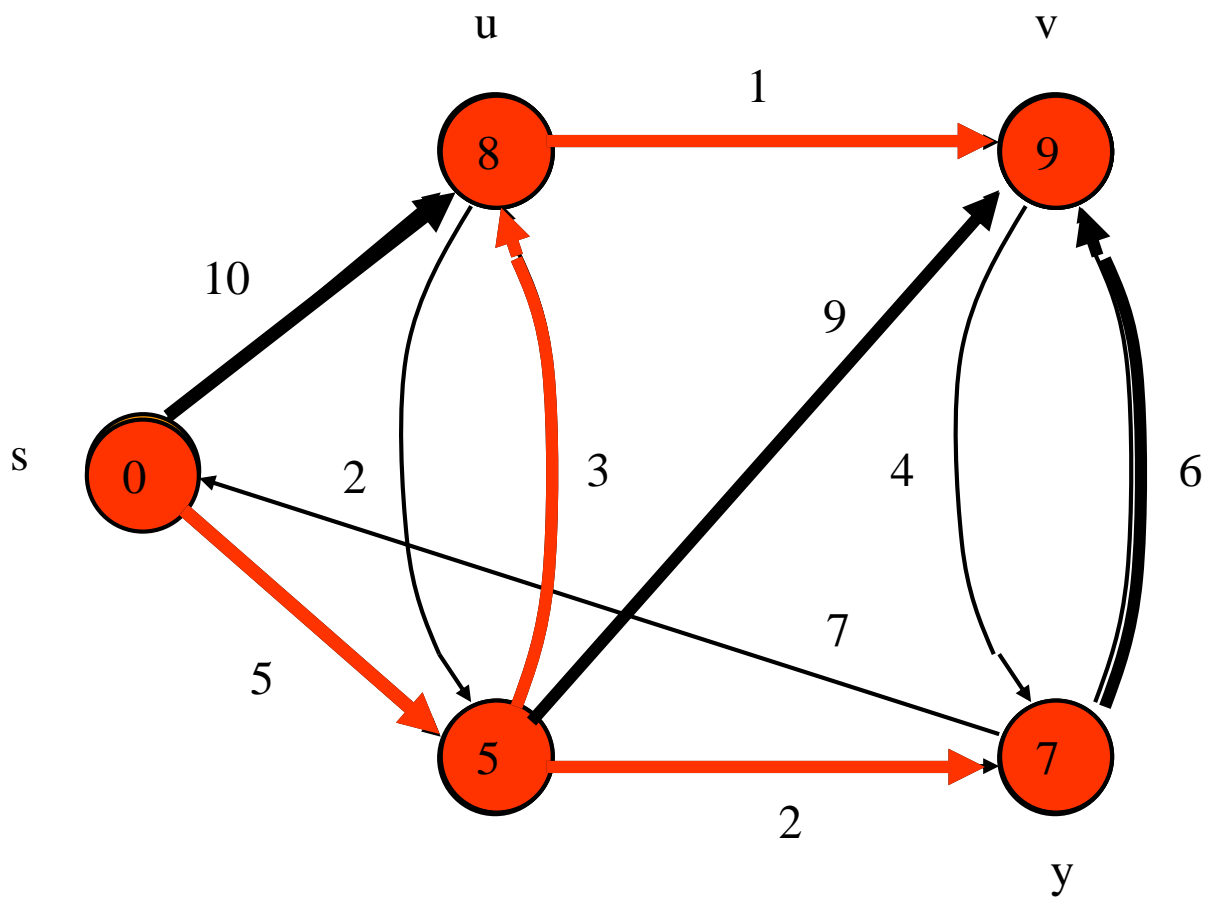


算法 Dijkstra(G, w, s)

Input 图 $G=(V, E)$, 边加权函数 w , 源顶点 s

Output s 到其所有可达顶点的最短路径

1. For $\forall v \in V$ do
2. $d[v] \leftarrow \infty$;
3. $\pi[v] \leftarrow \text{null}$;
4. $d[s] \leftarrow 0$;
5. $S \leftarrow \emptyset$
6. $Q \leftarrow V$
7. while $Q \neq \emptyset$ do
8. $u \leftarrow \text{EXTRACT-MIN}(Q)$;
9. $S \leftarrow S \cup \{u\}$
10. For $\forall v \in \text{Adj}[u]$ do
11. Relax(u, v, w);





第一步：假设 $\text{EXTRACT-MIN}(Q)=x$.

- sx 是仅含一条边的最短路径
 - 为什么？
 - 因为 sx 是从 s 出发的最短的边。
- 它也是 s 到 x 的最短路径

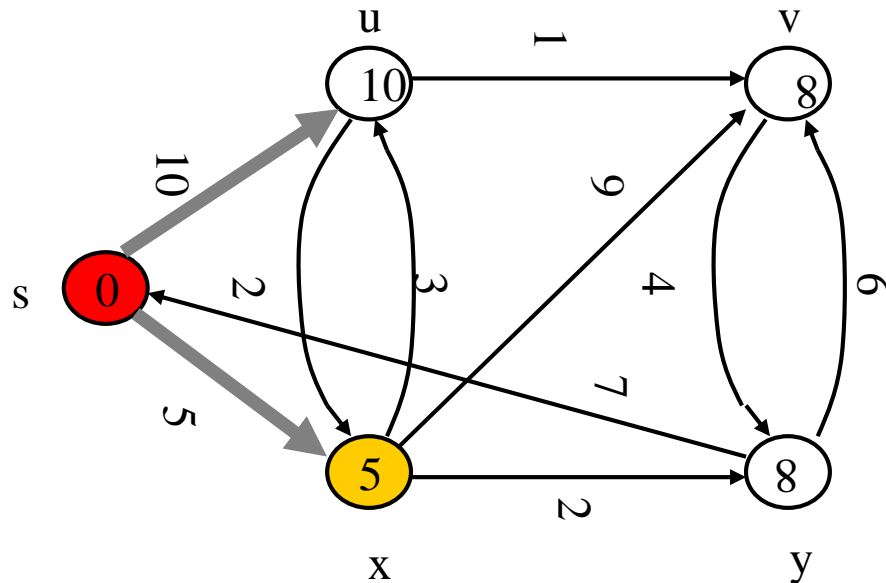
证明：

(1) 设 $P: s \rightarrow u \dots \rightarrow x$ 是 s 到 x 的最短路径，则 $w(s, u) \geq w(s, x)$.

(2) 由于图中没有负权值边，路径 P 的总权值至少为

$$w(s, u) \geq w(s, x).$$

(3) 故，边 sx 是 s 到 x 的最短路径。





第二步: $S=\{s, x\}$ $d[y] = \min_{v \in V-S} d[v]$

- 论断: $d[y]$ 是从 s 到 y 的最短路径代价, 即

- 要么 sy 是最短路径
- 要么 $s \rightarrow x \rightarrow y$ 是最短路径。

- 为什么?

- 如果 sy 是最短路径, 论断成立
- 考察 $s \rightarrow x \rightarrow y$ 是从 s 到 y 的最短路径的情况

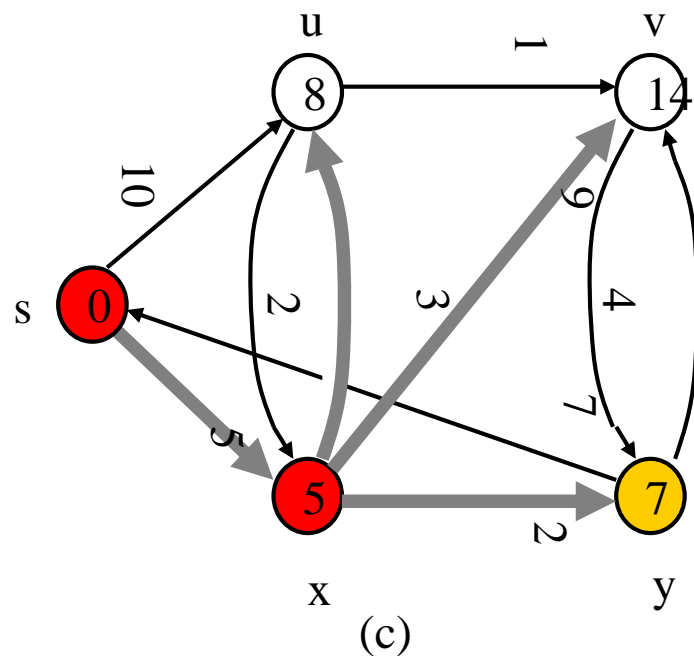
证明: (反证法) 设 $s \rightarrow x \rightarrow y$ 不是从 s 到 y 的最短路径

(1) 设 $P_1: s \rightarrow y' \rightarrow \dots \rightarrow y$ 是 s 到 y 的最短路径, 其中 $y' \notin S$. (注意此时, 我们已经考察了 $y'=x$ 和 $y'=s$ 的情形).

(2) 因此, $w(P_1) < w(s \rightarrow x \rightarrow y)$.

(3) 由于 $w(uv) \geq 0$ 对任意边成立, 故 $w(sy') < w(P_1) < w(s \rightarrow x \rightarrow y)$.

进而 $d[y'] < d[y]$, 这样算法第二步不可能选中 y , 矛盾!





后续步骤: 设 S 是算法维护的集合, 令 $d[y] = \min_{v \in V-S} d[v]$

- **定理:** $d[y]$ 是从 s 到 y 的最短路径代价 (正确性分析中最难的部分)

证明: (归纳法+反证)

归纳假设: 设对 $\forall v \in S$, $d[v]$ 是从 s 到 v 的最短路径的代价, 往证本次操作完成后 $d[y]$ 将是 s 到 y 的最短路径的代价

若不然, $d[y]$ **不是** 从 s 到 y 的最短路径的代价。设 $P_1: s \rightarrow \dots \rightarrow y' \rightarrow \dots \rightarrow y$ 是从 s 到 y 的最短路径, 其中 $y' \notin S$ 是 P_1 上第一个不属于 S 的顶点. 这意味着 $y \neq y'$ 且 $w(P_1) < d[y]$.

因此, $w(s \rightarrow \dots \rightarrow y') < w(P_1)$. (每条边的权值均非负)

进而 $w(s \rightarrow \dots \rightarrow y') < w(P_1) < d[y]$.

据此, $d[y'] = w(s \rightarrow \dots \rightarrow y') < w(P_1) < d[y]$.

因此, 算法在本次操作中不会选中 y , **矛盾!**



Dijkstra算法的时间复杂度

- 时间复杂度依赖于优先队列 Q 的实现
- 模型1: 利用数组存储 Q
 - **EXTRACT-MIN(Q)** —需要 $O(|V|)$ 时间.
 - 总共需要执行 $|V|$ 次 EXTRACT -MIN(Q).
 - $|V|$ 次 EXTRACT -MIN(Q) 操作的总时间为 $O(|V|^2)$.
 - **RELAX(u, v, w)** —需要 $O(1)$ 时间.
 - 总共需要执行 $|E|$ 次 RELAX(u, v, w) 操作.
 - $|E|$ 次 RELAX(u, v, w) 操作的总时间为 $O(|E|)$.
 - 总时间开销为 $O(|V|^2 + |E|) = O(|V|^2)$
- 模型2: Q 用斐波那契堆实现.
 - **EXTRACT-MIN(Q)** —平摊代价 $O(\log |V|)$.
 - **DECREASE-Key** —平摊代价 $O(1)$.
 - 总时间开销为 $O(|V| \log |V| + |E|)$.