算法设计与分析

第八章 图论算法

提纲

8.1 最短路径问题

8.2 网络流问题

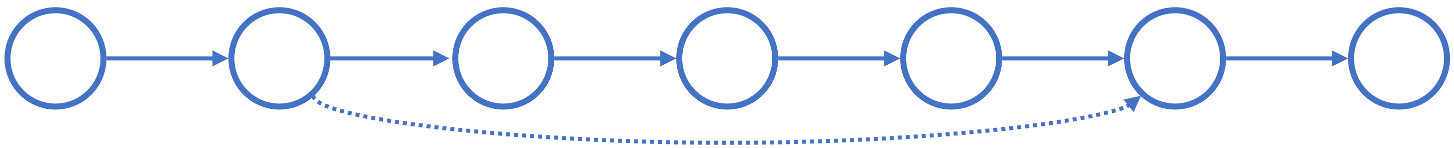
8.3 匹配问题

单源最短路径

问题：给定一个边加权的有向图**G**，找到从给定源结点**u**到图中所有其他结点**v**的最短路径。

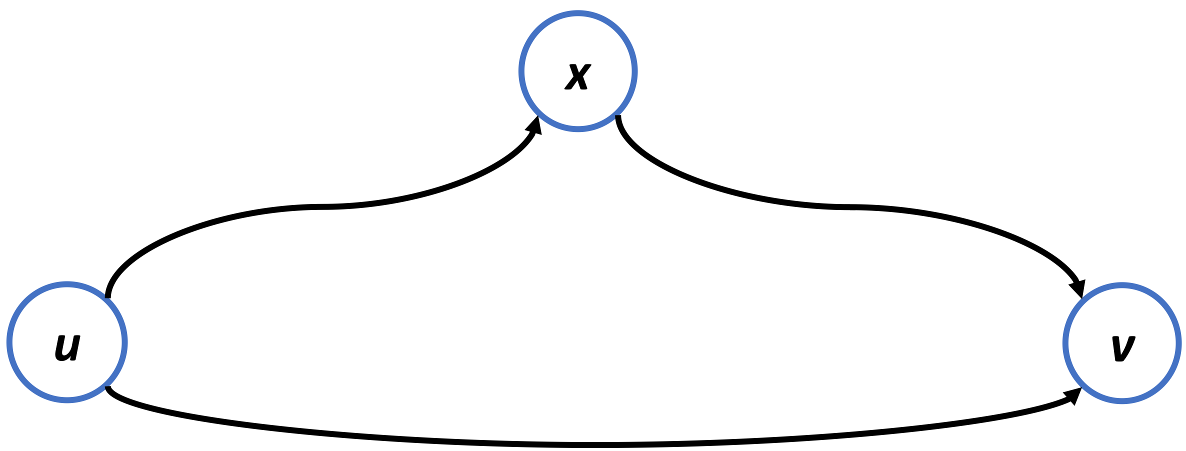
最短路径的性质

- 优化子结构: 最短路径包含最短子路径



- 三角不等式性质， 令$\delta(u,v)$ 是从u到v最短路径的权重

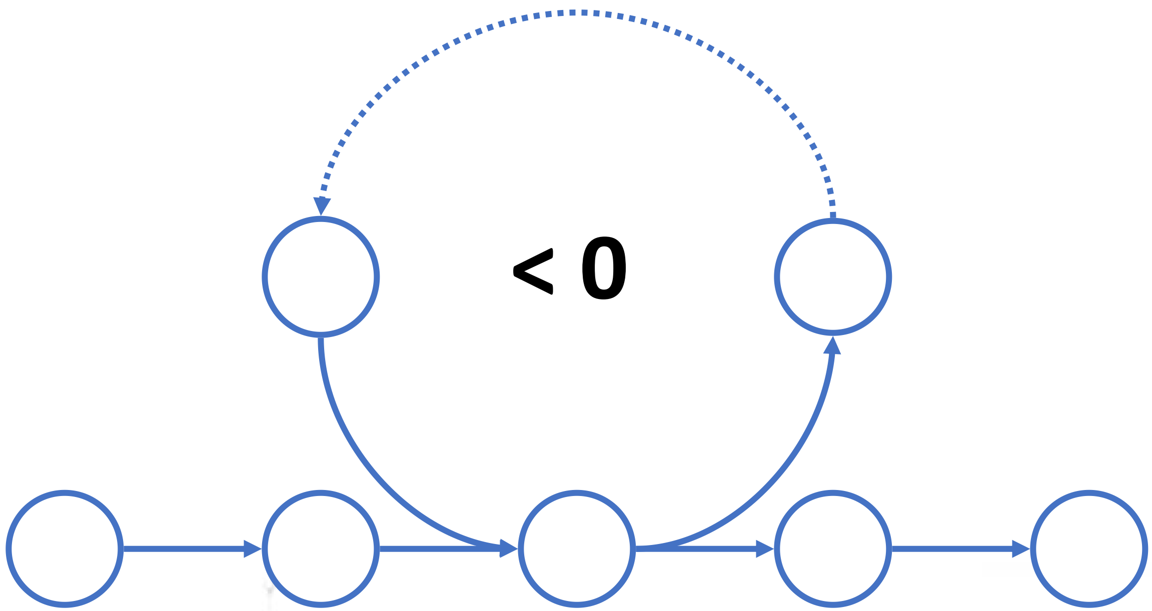
则： $\delta(u,v) \le \delta(u,x) + \delta(x,v)$



这条路径不会比另外两条之和长

最短路径的性质

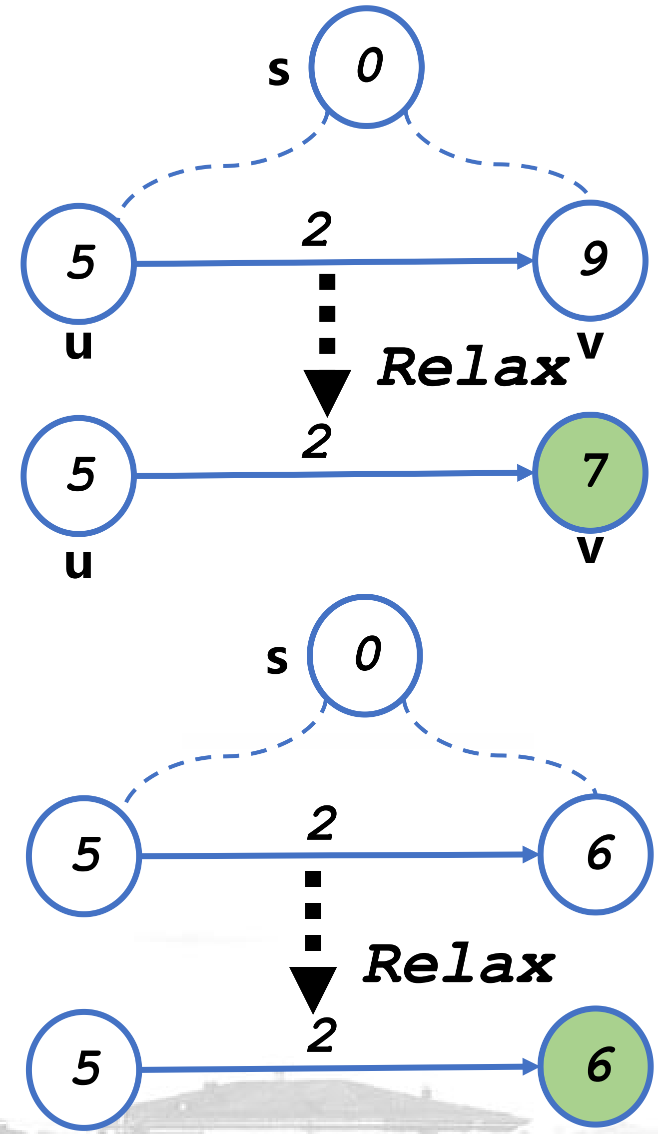
- 如果图中包含负圈，某些最短路径可能不存在:



松弛技术

- 最短路径算法的核心技术是松弛
- w(u,v) 表示结点 u 到结点 v 的边的权值， v.d 表示从起点 s 到结点 v 的路径权值, v.π表示 s 到结点 v 的当前路径上v的前序结点。

```
Relax(u,v,w(u,v))
{
    If (v.d > u.d+w(u,v)) Then
        v.d=u.d+w(u,v);
        v.π=u
}
```



Bellman-Ford 算法

Input：有向加权图G=(V,E), 边权重矩阵W, 源点s

Output：判断s到每个结点v是否有最短路径，如有最短路径权重存储在v.d中，路径信息存储在v.π中

过程：BellmanFord(G,w,s)

```
For each v ∈ G.V Do
    v.d ← ∞;
s.d ← 0;
For i←1 To |G.V|-1Do
    For each edge (u,v) ∈ G.E Do
        Relax(u,v, w(u,v));
For each edge (u,v) ∈ G.E Do
    IF (v.d > u.d + w(u,v)) Then
        Return FALSE;
Return True
```

初始化 d

对每条边松弛 |V|-1 遍

从u到v，最多经过V-1条边

检验结果

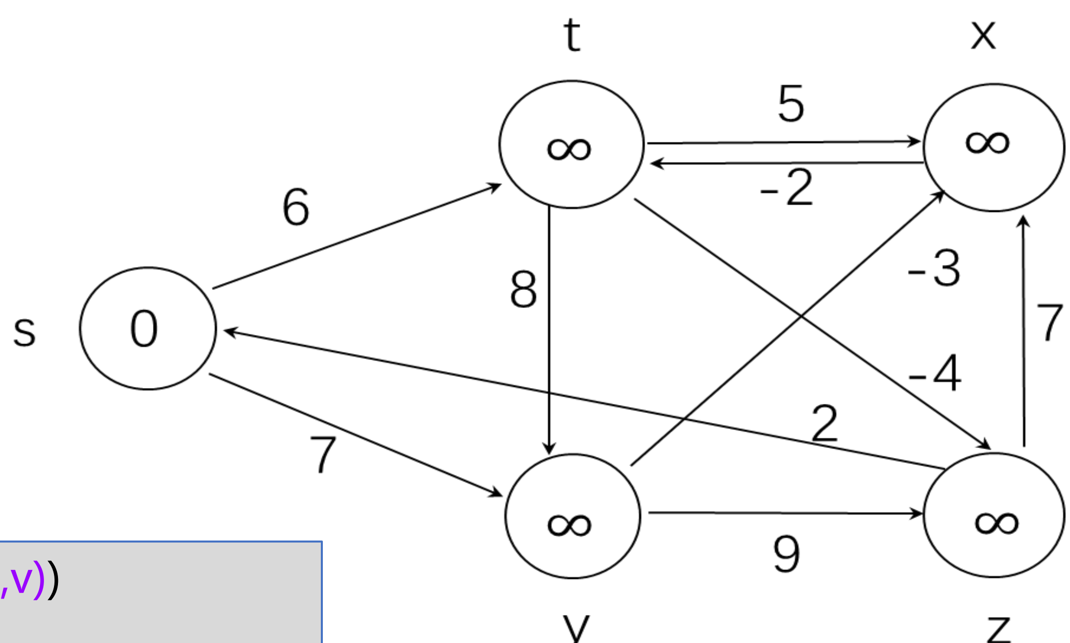
```
Relax(u,v,w(u,v))
{
    If (v.d > u.d+w(u,v))
Then
        v.d=u.d+w(u,v);
        v.π=u
}
```

时间复杂度：O(VE)

Bellman-Ford 算法

- 源点是s， s到各个结点v的当前路径权重d被标记在各自结点内，加粗的边指示了前趋边，每一趟按照如下顺序对边进行松弛：

(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)

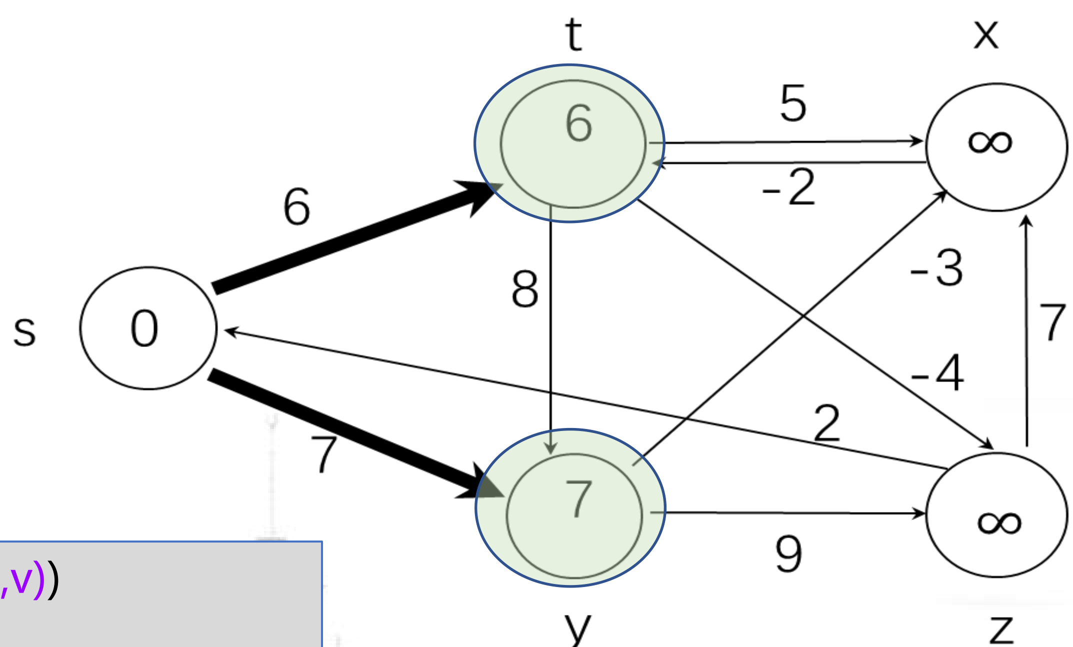



```
Relax(u,v,w(u,v))
{
    If (v.d > u.d+w(u,v)) Then
            v.d=u.d+w(u,v);
            v.π=u
}
```

Bellman-Ford 算法

- 源点是s， s到各个结点v的当前路径权重d被标记在各自结点内，加粗的边指示了前趋边，每一趟按照如下顺序对边进行松弛：

(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)

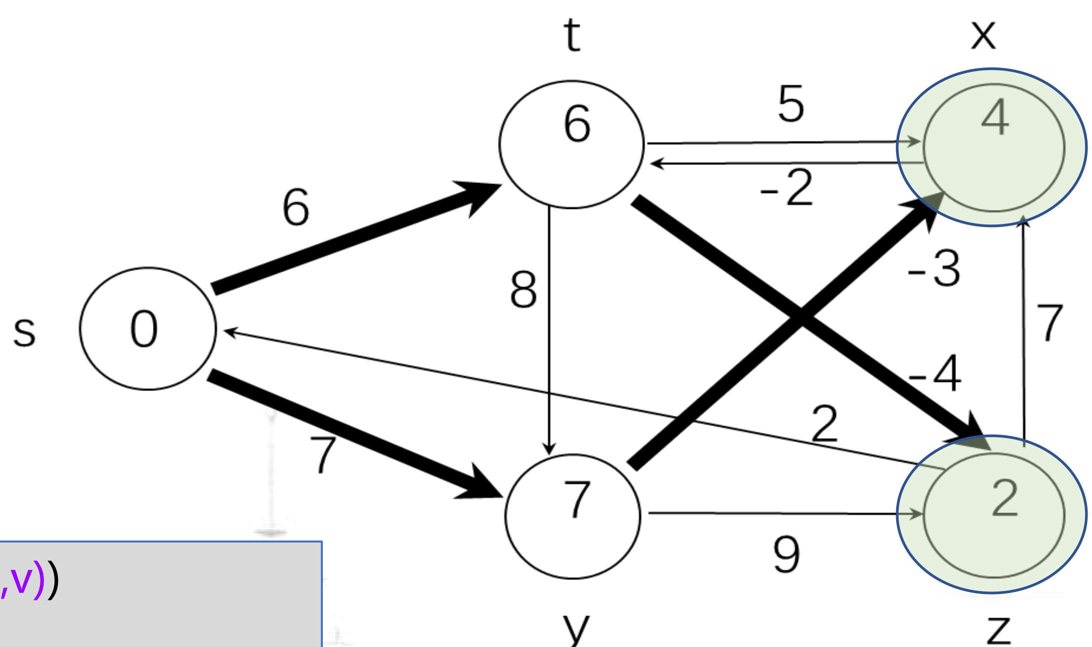



```
Relax(u,v,w(u,v))
{
    If (v.d > u.d+w(u,v)) Then
            v.d=u.d+w(u,v);
            v.π=u
}
```

Bellman-Ford 算法

- 源点是s， s到各个结点v的当前路径权重d被标记在各自结点内，加粗的边指示了前趋边，每一趟按照如下顺序对边进行松弛：

(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)

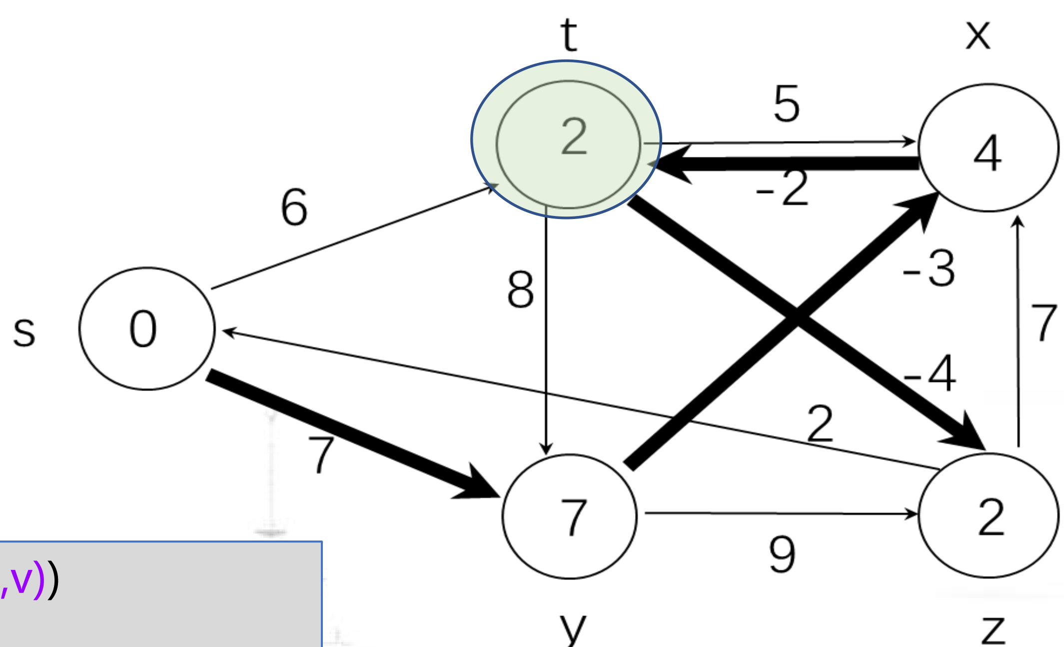



```
Relax(u,v,w(u,v))
{
    If (v.d > u.d+w(u,v)) Then
                v.d=u.d+w(u,v);
                v.π=u
}
```

Bellman-Ford 算法

- 源点是s，s到各个结点v的当前路径权重d被标记在各自结点内，加粗的边指示了前趋边，每一趟按照如下顺序对边进行松弛：

(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)

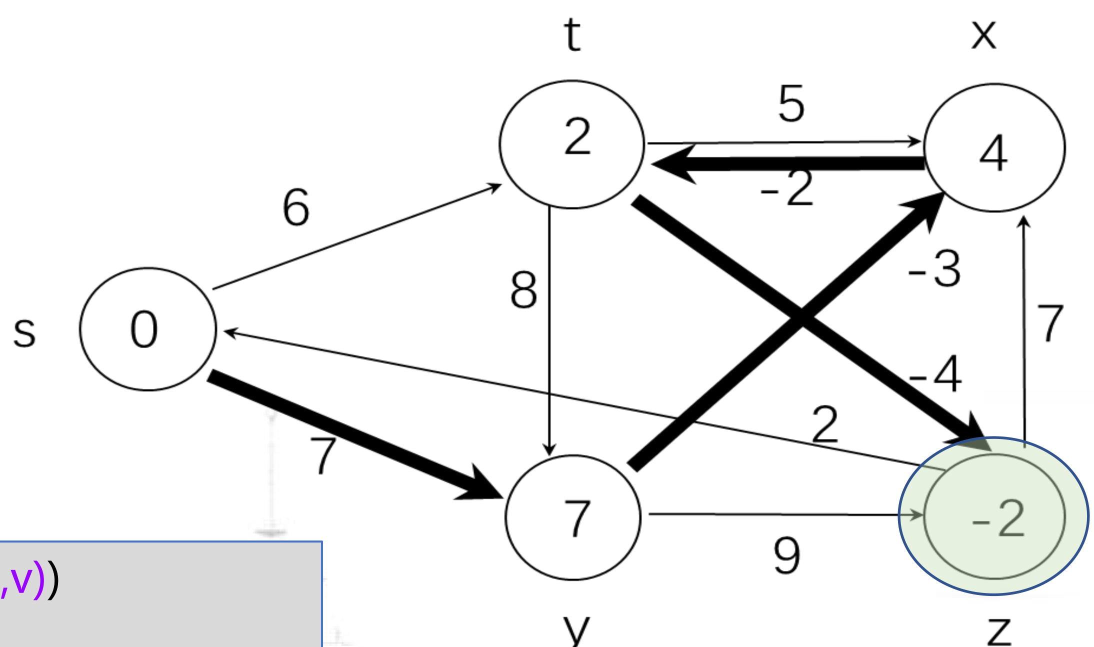



```
Relax(u,v,w(u,v))
{
    If (v.d > u.d+w(u,v)) Then
            v.d=u.d+w(u,v);
            v.π=u
}
```

Bellman-Ford 算法

- 源点是s，s到各个结点v的当前路径权重d被标记在各自结点内，加粗的边指示了前趋边，每一趟按照如下顺序对边进行松弛：

(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)




```
Relax(u,v,w(u,v))
{
    If (v.d > u.d+w(u,v)) Then
            v.d=u.d+w(u,v);
            v.π=u
}
```

有向无环图中最短路径

- 问题: 寻找加权有向无环图（Directed Acyclic Graph，DAG）中的单源最短路径
  - Bellman-Ford 时间是 O(VE)
  - 能否做的好一点呢?
- 思路: 使用拓扑排序
  - 如果沿着最短路径，则可以一遍完成
  - DAG中的每条路径都是通过拓扑排序得到的结点序列的一个子序列，那么，如果按照这个顺序处理，我们将沿着路径进行处理

Dijkstra算法

- 如果图中没有负边，Dijkstra效率远高于Bellman-Ford
- 类似Best-First搜索
  - 从队列中取结点
- 类似Prim算法
  - 使用以v.d为键的优先队列

Dijkstra算法

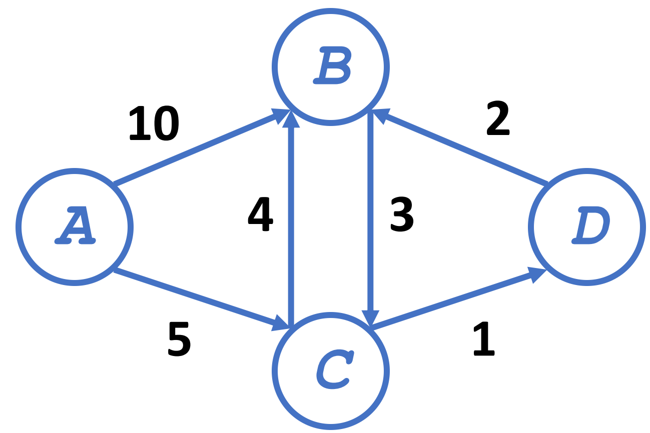
Input：有向加权图G=(V,E), 边权重矩阵W, 源点s

Output：最短路径权重存储在v.d中，路径信息存储在v.π中

过程：Dijkstra(G,w,s)

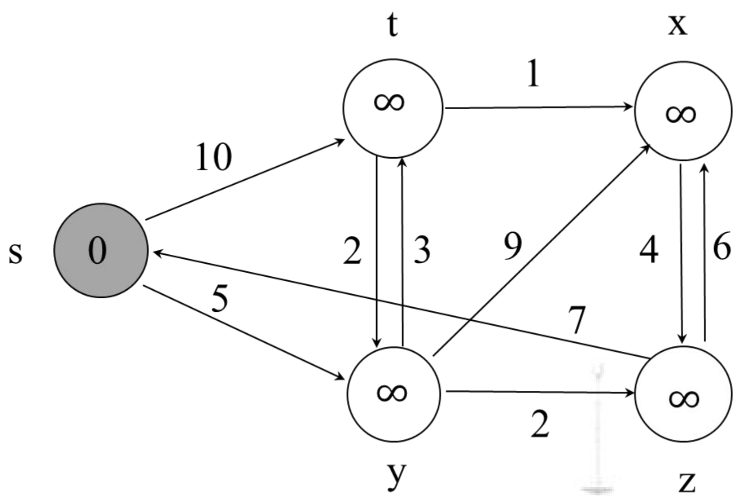
```
For each v ∈ G.V Do
    v.d ← ∞;
s.d ← 0; S ← ∅; Q ← V;
While (Q ≠ ∅) Do
    u ← ExtractMin(Q);
    S ← S U {u};
    For each v ∈ u->Adj[] Do
      If (v.d > u.d+w(u,v)) Then
        v.d ← u.d+w(u,v);
        v.π=u
```

松弛步骤



Dijkstra算法

- Dijkstra算法的执行过程：源点为s。当前最短路径被标记在结点内，加粗的边为前趋边。蓝色结点是集合S中的元素；白色顶点是优先队列Q中的元素；灰色顶点是当前具有最小d的结点，它被选作下一次迭代第5行的结点u。



Input：有向加权图G=(V,E), 边权重矩阵W, 源点s

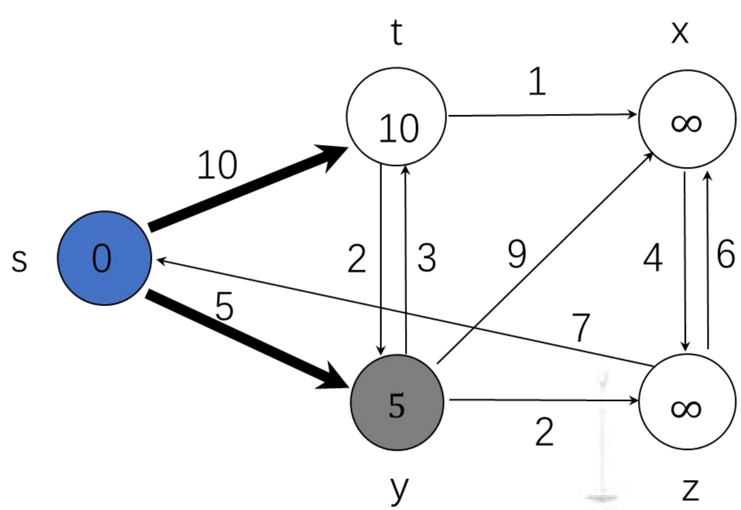
Output：最短路径权重存储在v.d中，路径信息存储在v.π中

过程：Dijkstra(G,w,s)

```
1.  For each v ∈ G.V Do
2.      v.d ← ∞;
3.  s.d ← 0; S ← ∅; Q ← V;
4.  While (Q ≠ ∅) Do
5.      u ← ExtractMin(Q);
6.      S ← S U {u};
7.      For each v ∈ u->Adj[] Do
8.          If (v.d > u.d+w(u,v)) Then
9.              v.d ← u.d+w(u,v);
10.             v.π=u
```

Dijkstra算法

- Dijkstra算法的执行过程：源点为s。当前最短路径被标记在结点内，加粗的边为前趋边。蓝色结点是集合S中的元素；白色顶点是优先队列Q中的元素；灰色顶点是当前具有最小d的结点，它被选作下一次迭代第5行的结点u。



Input：有向加权图G=(V,E), 边权重矩阵W, 源点s

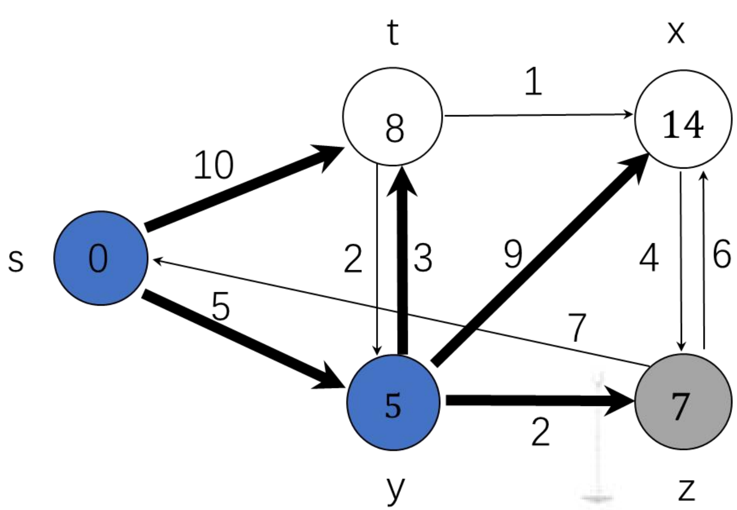
Output：最短路径权重存储在v.d中，路径信息存储在v.π中

过程：Dijkstra(G,w,s)

```
1.  For each v ∈ G.V Do
2.      v.d ← ∞;
3.  s.d ← 0; S ← ∅; Q ← V;
4.  While (Q ≠ ∅) Do
5.      u ← ExtractMin(Q);
6.      S ← S U {u};
7.      For each v ∈ u->Adj[] Do
8.          If (v.d > u.d+w(u,v)) Then
9.              v.d ← u.d+w(u,v);
10.             v.π=u
```

Dijkstra算法

- Dijkstra算法的执行过程：源点为s。当前最短路径被标记在结点内，加粗的边为前趋边。蓝色结点是集合S中的元素；白色顶点是优先队列Q中的元素；灰色顶点是当前具有最小d的结点，它被选作下一次迭代第5行的结点u。



Input：有向加权图G=(V,E), 边权重矩阵W, 源点s

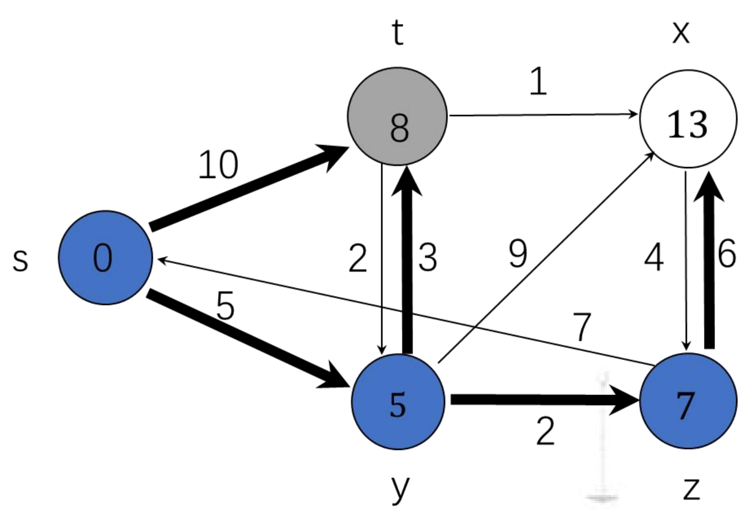
Output：最短路径权重存储在v.d中，路径信息存储在v.π中

过程：Dijkstra(G,w,s)

```
1.  For each v ∈ G.V Do
2.      v.d ← ∞;
3.  s.d ← 0; S ← ∅; Q ← V;
4.  While (Q ≠ ∅) Do
5.      u ← ExtractMin(Q);
6.      S ← S U {u};
7.      For each v ∈ u->Adj[] Do
8.          If (v.d > u.d+w(u,v)) Then
9.              v.d ← u.d+w(u,v);
10.             v.π=u
```

Dijkstra算法

- Dijkstra算法的执行过程：源点为s。当前最短路径被标记在结点内，加粗的边为前趋边。蓝色结点是集合S中的元素；白色顶点是优先队列Q中的元素；灰色顶点是当前具有最小d的结点，它被选作下一次迭代第5行的结点u。



Input：有向加权图G=(V,E), 边权重矩阵W, 源点s

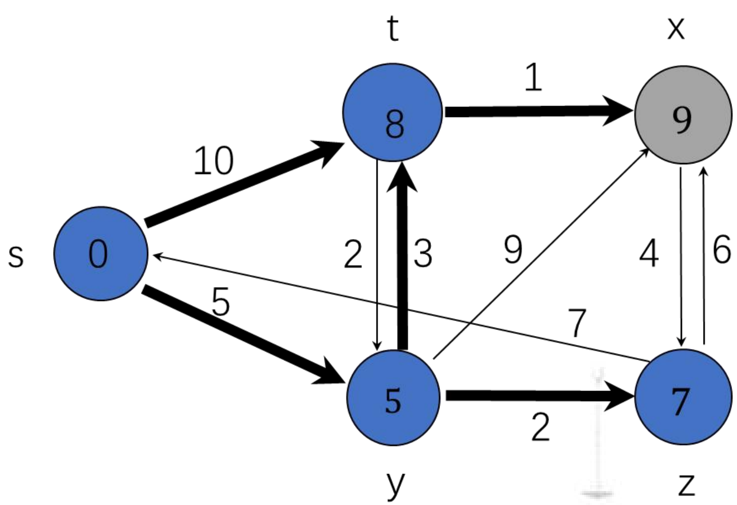
Output：最短路径权重存储在v.d中，路径信息存储在v.π中

过程：Dijkstra(G,w,s)

```
1.  For each v ∈ G.V Do
2.      v.d ← ∞;
3.  s.d ← 0; S ← ∅; Q ← V;
4.  While (Q ≠ ∅) Do
5.      u ← ExtractMin(Q);
6.      S ← S U {u};
7.      For each v ∈ u->Adj[] Do
8.          If (v.d > u.d+w(u,v)) Then
9.              v.d ← u.d+w(u,v);
10.             v.π=u
```

Dijkstra算法

- Dijkstra算法的执行过程：源点为s。当前最短路径被标记在结点内，加粗的边为前趋边。蓝色结点是集合S中的元素；白色顶点是优先队列Q中的元素；灰色顶点是当前具有最小d的结点，它被选作下一次迭代第5行的结点u。



Input：有向加权图G=(V,E), 边权重矩阵W, 源点s

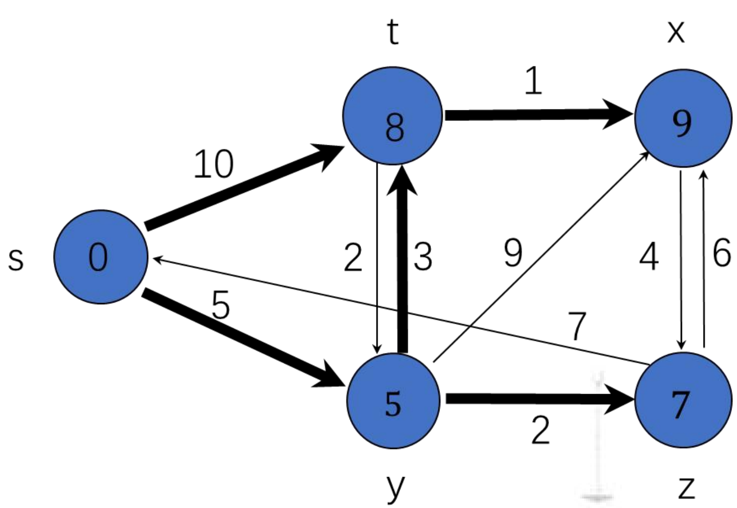
Output：最短路径权重存储在v.d中，路径信息存储在v.π中

过程：Dijkstra(G,w,s)

1. For each $v \in G.V$ Do
2. $\quad v.d \leftarrow \infty$;
3. $s.d \leftarrow 0$; $S \leftarrow \varnothing$; $Q \leftarrow V$;
4. While $(Q \neq \varnothing)$ Do
5. $\quad u \leftarrow$ ExtractMin(Q);
6. $\quad S \leftarrow S \cup \{u\}$;
7. $\quad$For each $v \in$ u->Adj[] Do
8. $\quad\quad$If $(v.d > u.d+w(u,v))$ Then
9. $\quad\quad\quad v.d \leftarrow u.d+w(u,v)$;
10. $\quad\quad\quad v.\pi = u$

Dijkstra算法

- Dijkstra算法的执行过程：源点为s。当前最短路径被标记在结点内，加粗的边为前趋边。蓝色结点是集合S中的元素；白色顶点是优先队列Q中的元素；灰色顶点是当前具有最小d的结点，它被选作下一次迭代第5行的结点u。



Input：有向加权图G=(V,E), 边权重矩阵W, 源点s

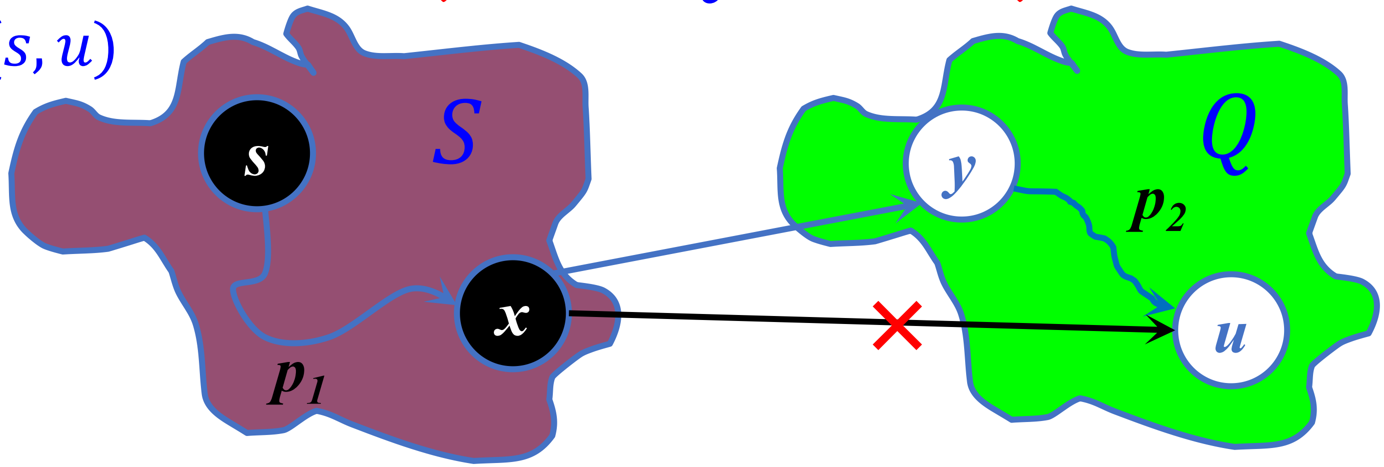
Output：最短路径权重存储在v.d中，路径信息存储在v.π中

过程：Dijkstra(G,w,s)

```
1.  For each v ∈ G.V Do
2.      v.d ← ∞;
3.  s.d ← 0; S ← ∅; Q ← V;
4.  While (Q ≠ ∅) Do
5.      u ← ExtractMin(Q);
6.      S ← S ∪ {u};
7.      For each v ∈ u->Adj[] Do
8.          If (v.d > u.d+w(u,v)) Then
9.              v.d ← u.d+w(u,v);
10.             v.π=u
```

Dijkstra 算法的正确性

正确性：我们必须证明，当u从Q中取出时，它已经收敛了，即 $d[u] = \delta(s, u)$



证明：$\delta(s, u)$是s到u最短路径的值, $d[u]$是s到u的当前路径的值，对$\forall v$我们有：$d[v] \geq \delta(s, v)$

- u是从Q中选的出第一个结点。假设$d[u] \neq \delta(s, u)$，则s到u的真实最短路径P，不能由S中的x直接到u。即存在比$d[u]$更短的路径，故$d[u] > \delta(s, u)$
- 那么在Q中，必存在一个不是u的结点y，y是$s \rightarrow u$真实最短路径P上的第一个属于集合Q的结点（此时路径$P = s \xrightarrow{p_1} x \rightarrow y \xrightarrow{p_2} u$），故：

$$d[y] = \delta(s, y);\ d[u] > \delta(s, u) = \delta(s, y) + \delta(y, u)$$
$$= d[y] + \delta(y, u) \geq d[y]$$

- 但是如果 $d[u] > d[y]$，则不会选择u。矛盾。

多源最短路径

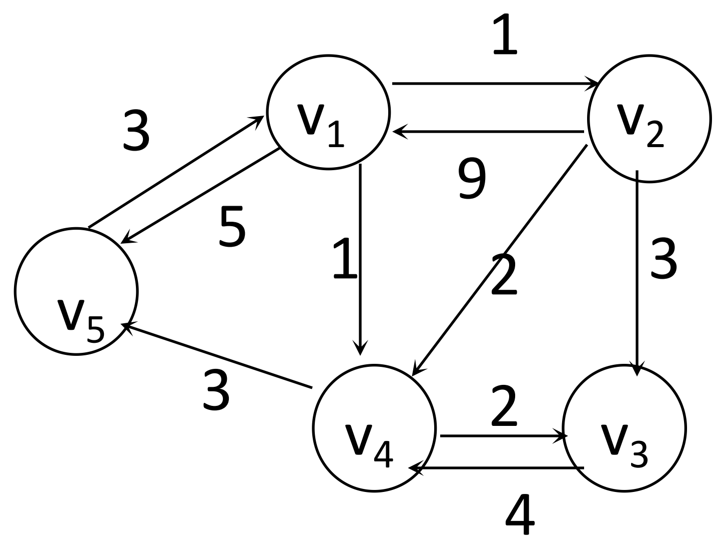
问题: 找到图中每一对结点间最短路径,图可能包含负边

但是不包含负圈

动态规划:Floyd-Warshall算法

图和权矩阵

	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	∞
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	∞	∞	∞	0



多源最短路径的动态规划算法

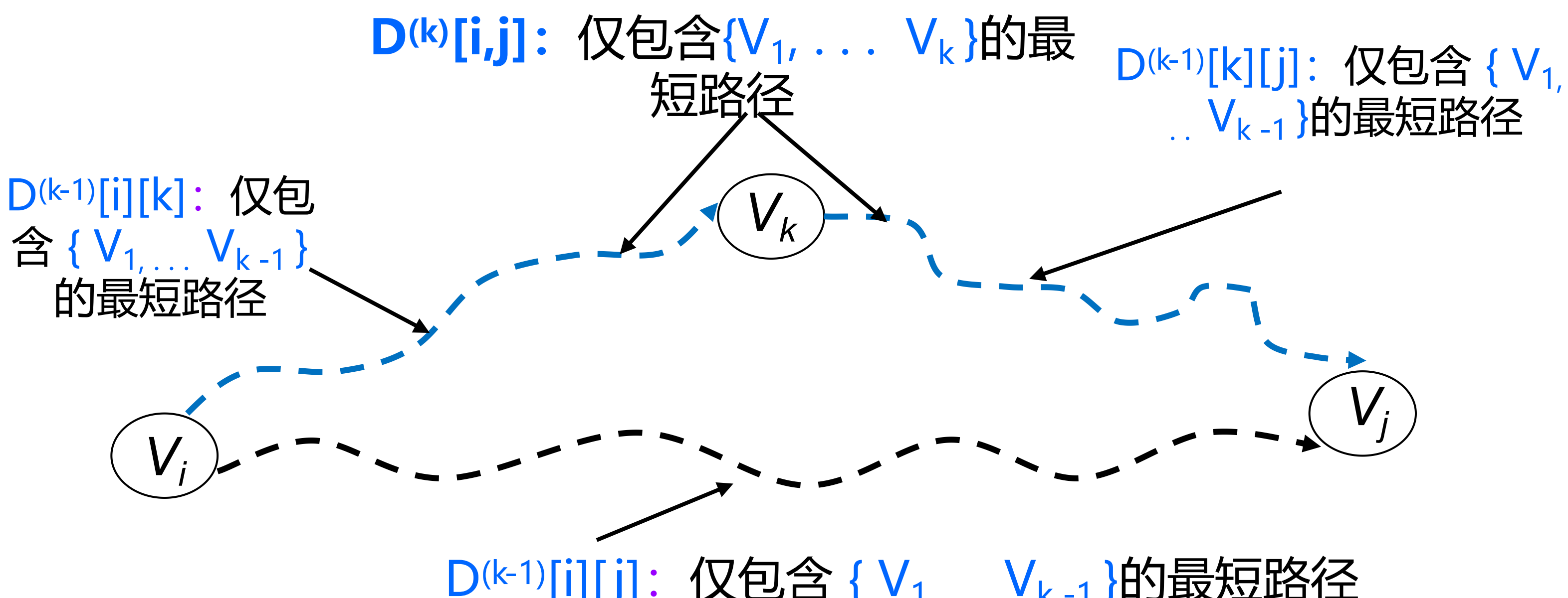
- 分析优化解的结构
- 递归地定义最优解的代价
- 自底向上地计算优化解的代价并保存，同时获取构造最优解的信息
- 根据构造最优解的信息构造优化解

子问题

- 如何定义更小的问题?
  - 一种方法是将路径限制在仅包含一个有限集合中的结点
  - 开始这个集合是空的
  - 这个集合可以一直增长到包含所有结点

子问题

- 令 $D^{(k)}[i,j]$=从v_i 到 v_j仅包含 $\{v_1,v_2,\ldots,v_k\}$ 的路径
  - $D^{(0)}=W$
  - $D^{(n)}=D$ 目标矩阵
- 如何从$D^{(k-1)}$计算 $D^{(k)}$?
  - $D^{(k)}[i,j]= D^{(k-1)}[i,j]$
  - 或者$D^{(k)}[i,j]= D^{(k-1)}[i,k]+ D^{(k-1)}[k,j]$



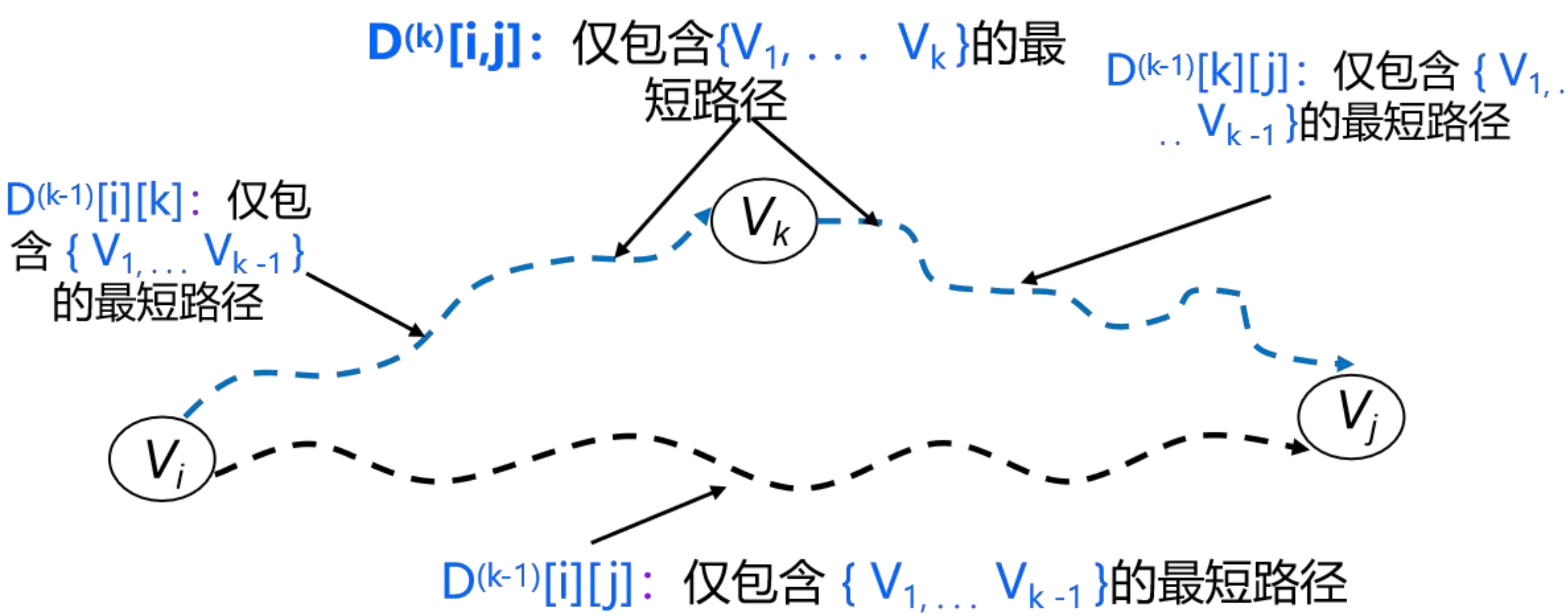
多源最短路径的动态规划算法

- 分析优化解的结构
- 递归地定义最优解的代价
- 自底向上地计算优化解的代价并保存，同时获取构造最优解的信息
- 根据构造最优解的信息构造优化解

递归定义有化解的代价

根据优化子结构：

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



多源最短路径的动态规划算法

- 分析优化解的结构
- 递归地定义最优解的代价
- 自底向上地计算优化解的代价并保存，同时获取构造最优解的信息
- 根据构造最优解的信息构造优化解

Floyd-Warshall算法

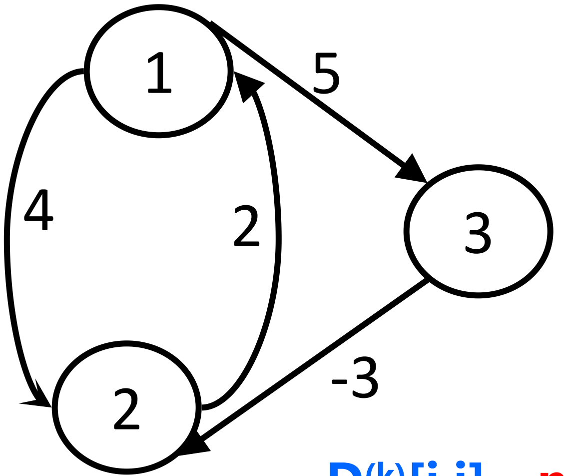
Input： 有向加权边权重矩阵W,结点个数n

Output： 最短路径权重矩阵D^n，路径信息存储矩阵P

过程：Floyd-Warshall（W,n)

1. $D^0 \leftarrow W$ //初始化D
2. $P \leftarrow 0$ // 初始化 P
3. For $k \leftarrow 1$ To n Do
4. For $i \leftarrow 1$ To n Do
5. For $j \leftarrow 1$ To n Do
6. IF $(D^{k-1}[i, j] > D^{k-1}[i, k] + D^{k-1}[k, j])$ Then
7. $D^k[i, j] \leftarrow D^{k-1}[i, k] + D^{k-1}[k, j]$
8. $P[i, j] \leftarrow k$;
9. ELSE $D^k[i, j] \leftarrow D^{k-1}[i, j]$
10. Return D^n,P

示例



W =

	1	2	3
1	0	4	5
2	2	0	∞
3	∞	-3	0

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

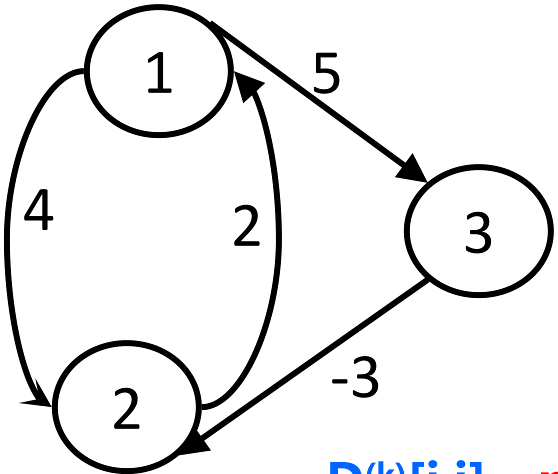
D^0 =

	1	2	3
1	0	4	5
2	2	0	∞
3	∞	-3	0

P =

	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0

示例



$D^0 =$

	1	2	3
1	0	4	5
2	2	0	∞
3	∞	-3	0

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

$$D^1[2,3] = \min(\, D^0[2,3], D^0[2,1] + D^0[1,3] \,) = \min\,(\propto, 7) = 7$$

$$D^1[3,2] = \min(\, D^0[3,2], D^0[3,1] + D^0[1,2] \,) = \min\,(-3, \propto) = -3$$

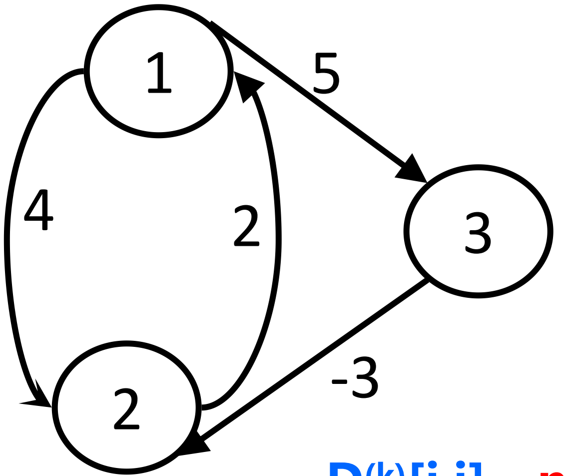
$D^1 =$

	1	2	3
1	0	4	5
2	2	0	7
3	∞	-3	0

P =

	1	2	3
1	0	0	0
2	0	0	1
3	0	0	0

示例



$D^1 =$

	1	2	3
1	0	4	5
2	2	0	7
3	∞	-3	0

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

$$D^2[1,3] = \min(\, D^1[1,3], D^1[1,2] + D^1[2,3]\,) = \min(5, 4+7) = 5$$

$$D^2[3,1] = \min(\, D^1[3,1], D^1[3,2] + D^1[2,1]\,) = \min(\infty, -3+2) = -1$$

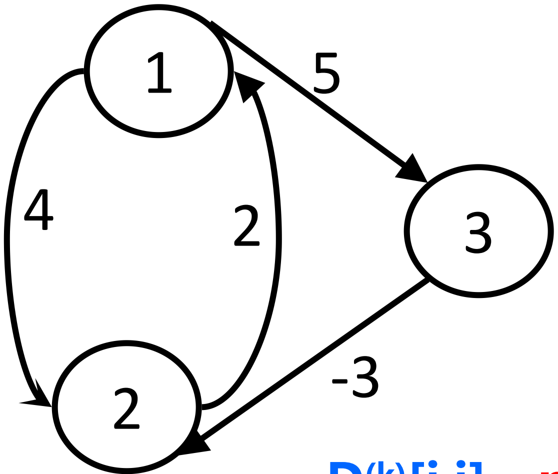
$D^2 =$

	1	2	3
1	0	4	5
2	2	0	7
3	-1	-3	0

$P =$

	1	2	3
1	0	0	0
2	0	0	1
3	2	0	0

示例



$D^2 =$

	1	2	3
1	0	4	5
2	2	0	7
3	-1	-3	0

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

$$D^3[1,2] = \min(D^2[1,2], D^2[1,3] + D^2[3,2]) = \min(4, 5+(-3)) = 2$$

$$D^3[2,1] = \min(D^2[2,1], D^2[2,3] + D^2[3,1]) = \min(2, 7+(-1)) = 2$$

$D^3 =$

	1	2	3
1	0	2	5
2	2	0	7
3	-1	-3	0

$P =$

	1	2	3
1	0	3	0
2	0	0	1
3	2	0	0

Floyd-Warshall算法:使用2个D矩阵

Input：有向加权边权重矩阵W,结点个数n

Output：最短路径权重矩阵D，路径信息存储矩阵P

过程：Floyd-Warshall_2D（W,n）

1. D $\leftarrow$ W
2. P $\leftarrow$ 0
3. For k $\leftarrow$ 1 to n Do// Computing D’ from D
4. For i $\leftarrow$ 1 to n Do
5. For j $\leftarrow$ 1 to n Do
6. If (D[i, j] > D[i, k] + D[k, j]) Then
7. D'[i, j] $\leftarrow$ D[i, k] + D[k, j]
8. P[i, j] $\leftarrow$ k;
9. Else D'[i, j] $\leftarrow$ D[i, j]
10. Move D' to D
11. Return D,P

Floyd-Warshall算法:使用1个D矩阵

Input：有向加权边权重矩阵W,结点个数n

Output：最短路径权重矩阵D，路径信息存储矩阵P

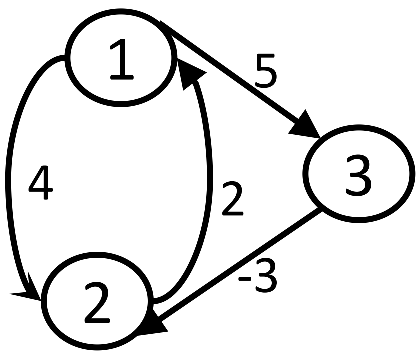
过程：Floyd-Warshall_1D（W,n）

```
D ← W
P ← 0
For k ← 1 To n Do
    For i ← 1 To n Do
        For j ← 1 To n Do
            If (D[ i, j ] > D[ i, k ] + D[ k, j ] )  Then
                D[ i, j ] ← D[ i, k ] + D[ k, j ]
                P[ i, j ] ← k;
Return D, P
```

多源最短路径的动态规划算法

- 分析优化解的结构
- 递归地定义最优解的代价
- 自底向上地计算优化解的代价并保存，同时获取构造最优解的信息
- 根据构造最优解的信息构造优化解

打印出从q到r的路径



P =

	1	2	3
1	0	3	0
2	0	0	1
3	2	0	0

path(q, r)

1. If (P[q, r]!=0) Then
2. path(q, P[q, r])
3. println(“→v” + P[q, r])
4. path(P[q, r], r)
5. return;//no intermediate nodes
6. Else return

提纲

8.1 最短路径问题

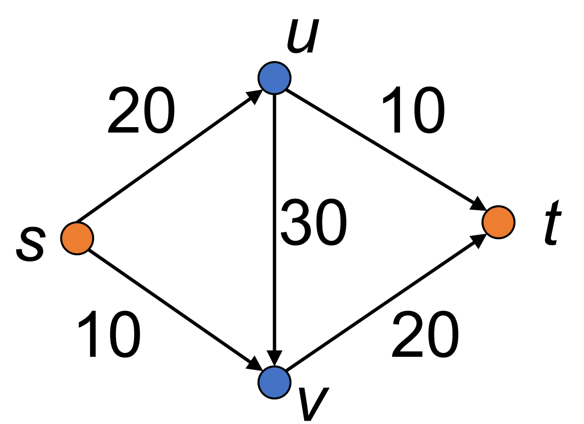
8.2 网络流问题

8.3 匹配问题

网络

有向图 $G = (V,E)$ 满足

- 每个有向边 e 有一个非负容量c_e
- 有一个源结点 s没有入边
- 有一个目标点 t 没有出边

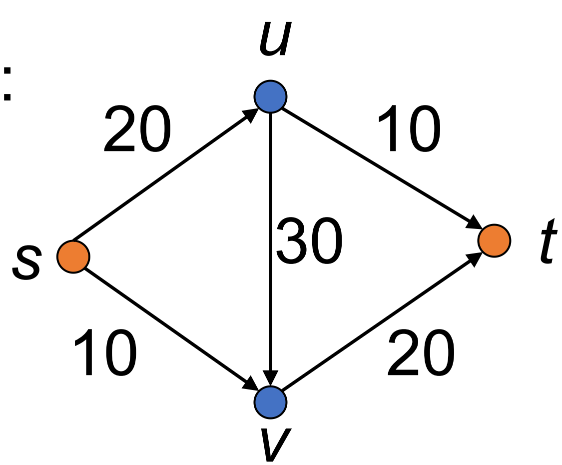


流

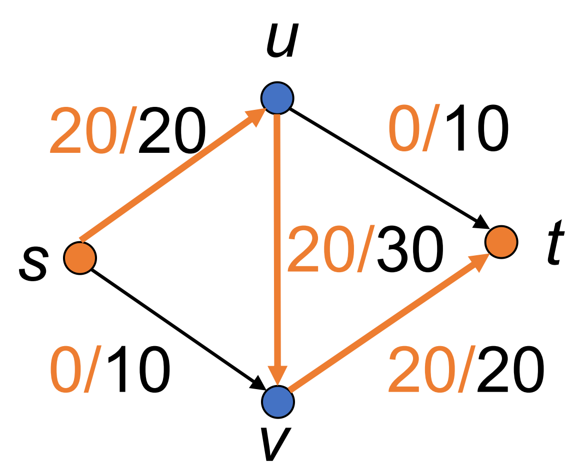
$G = (V,E)$ 中的s-t 流是一个从E 到 R^+的函数 f满足:

- 容量条件: 对每个e, $0 \le f(e) \le c_e$
- 保存条件: 对每个中间结点v, $\sum_{e \text{ in } v} f(e) = \sum_{e \text{ out } v} f(e)$
- 源和汇满足: $\sum_{e \text{ in } t} f(e) = \sum_{e \text{ out } s} f(e)$

网络:



流:

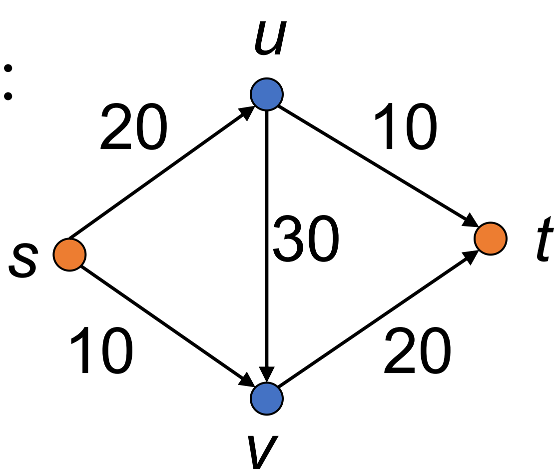


相关定义

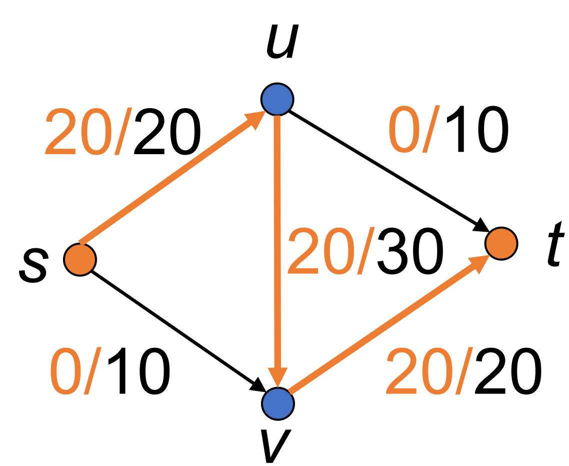
给定图$G = (V,E)$中 s-t 流 f 和任意结点集合 S:

- $f^{in}(S) = \sum_{e \text{ in } S} f(e)$
- $f^{out}(S) = \sum_{e \text{ out } S} f(e)$
- 满足: $f^{in}(t) = f^{out}(s)$，如 $f^{in}(u,v) = f^{out}(u,v) = 20$

网络:



流:

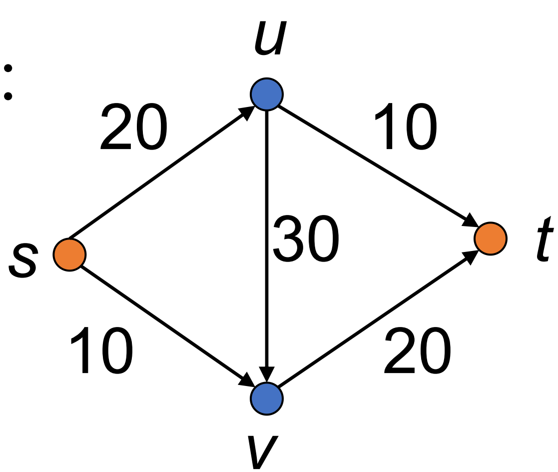


问题

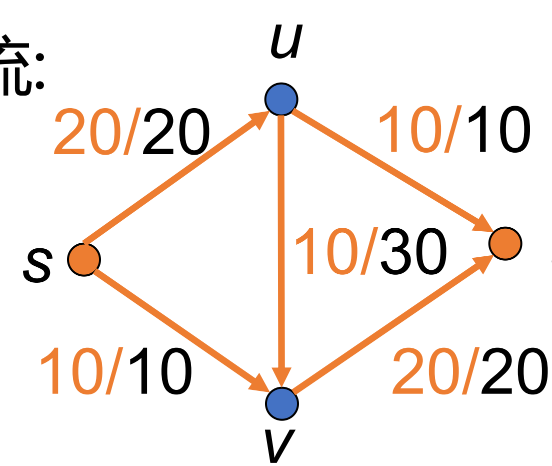
对于给定的图$G = (V,E)$，容量都是正数，如何求最大流?

- 如何高效地计算?
- 如最大流: $f^{in}(t) = f^{out}(s) = 30$

网络:



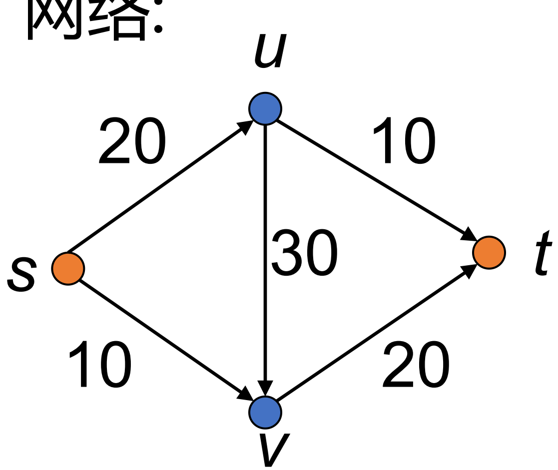
最大流:



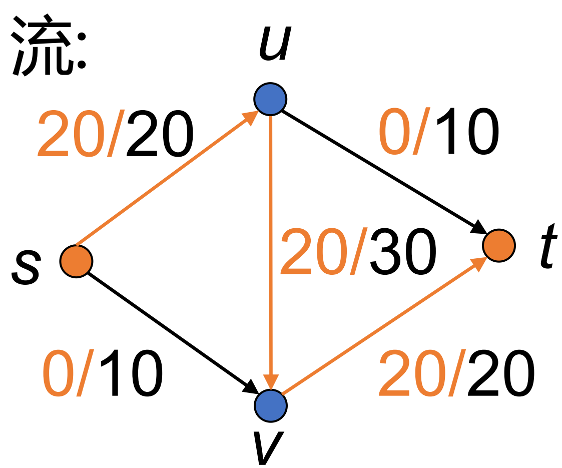
余图

- 给定图G中的流f,余图 G_f 定义如下：
  - 同样的结点，中间结点和s,t,
  - 对于每条边 $e = (u,v)$
    - 满足 $c_e > f(e)$ 赋给权重 $c_f((u,v)) = c_e - f(e)$
    - 对其逆向边(v,u)赋给权重 $c_f((v,u)) = f(e)$

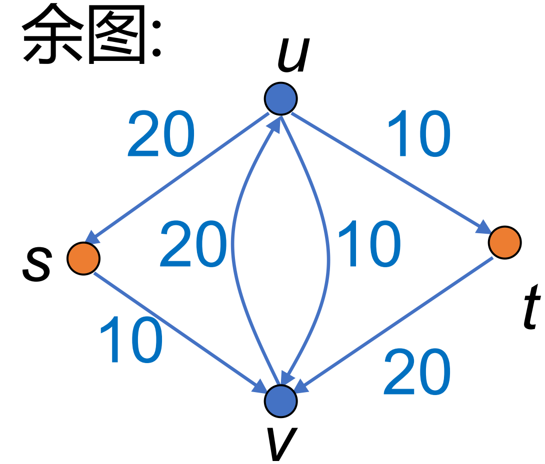
网络:



流:



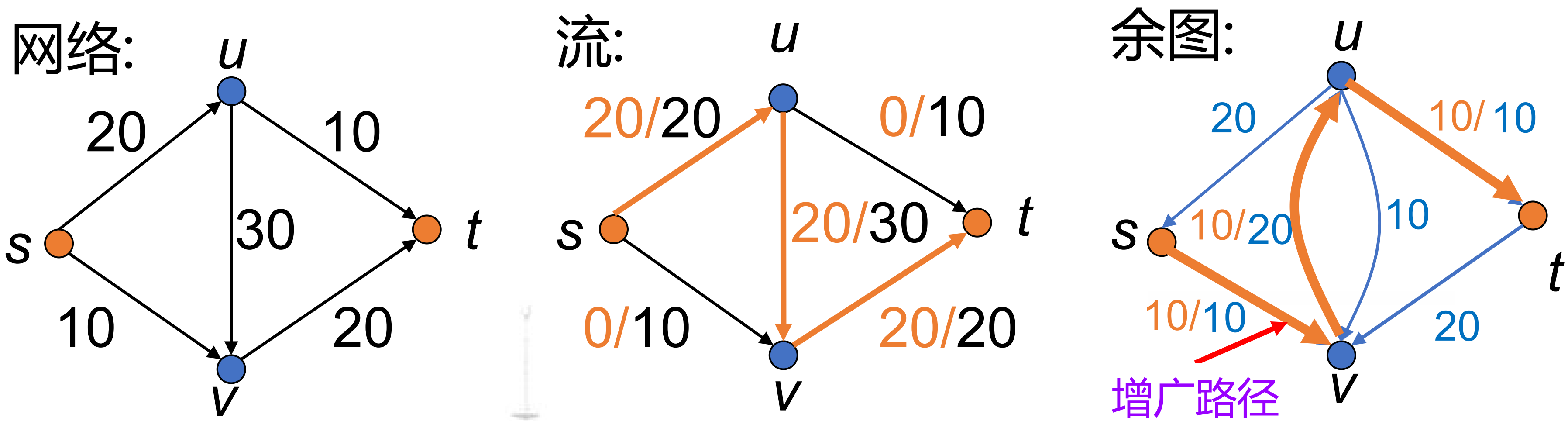
余图:



增广路径和增广

给定图G中的流f, 及其对应的余图G_f

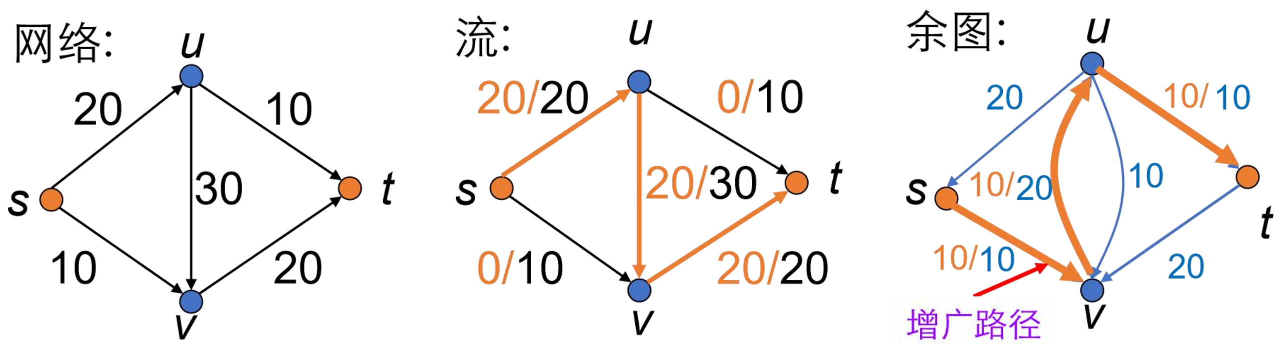
- 找到余图中的一条流，该流通过一条没有重复结点的路径，并且值和该路径上的最小容量相等(增广路径)



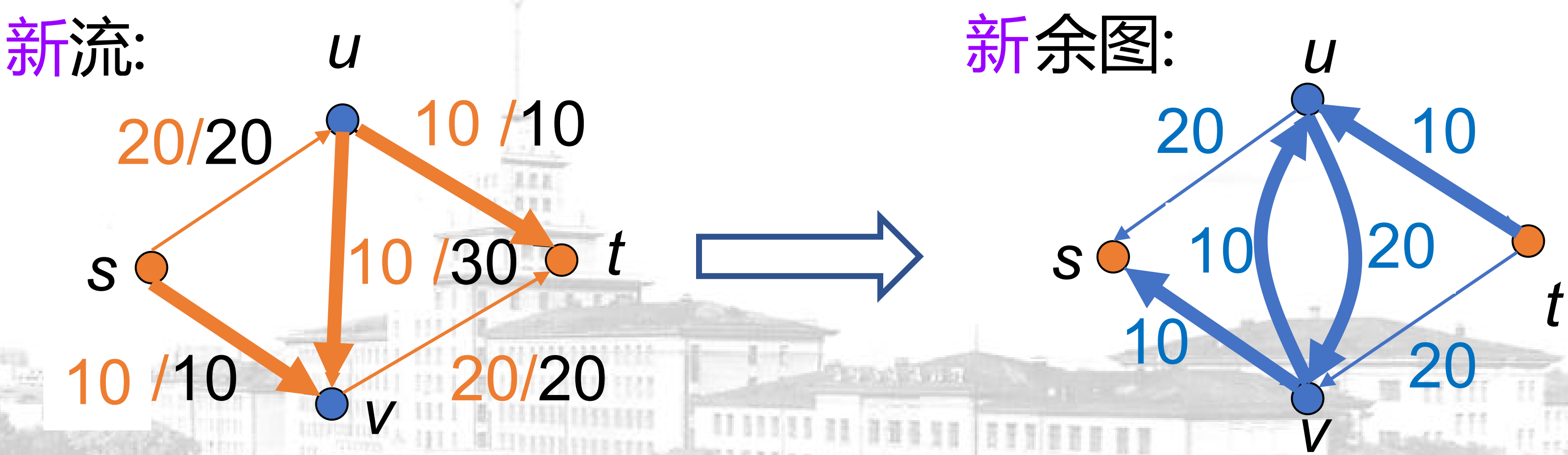
增广路径和增广

给定图G中的流f, 及其对应的余图G_f

- 找到余图中的一条流，该流通过一条没有重复结点的路径，并且值和该路径上的最小容量相等(增广路径)



- 根据增广路径上的最小流，沿着增广路径更新流（增广）



Ford-Fulkerson 方法

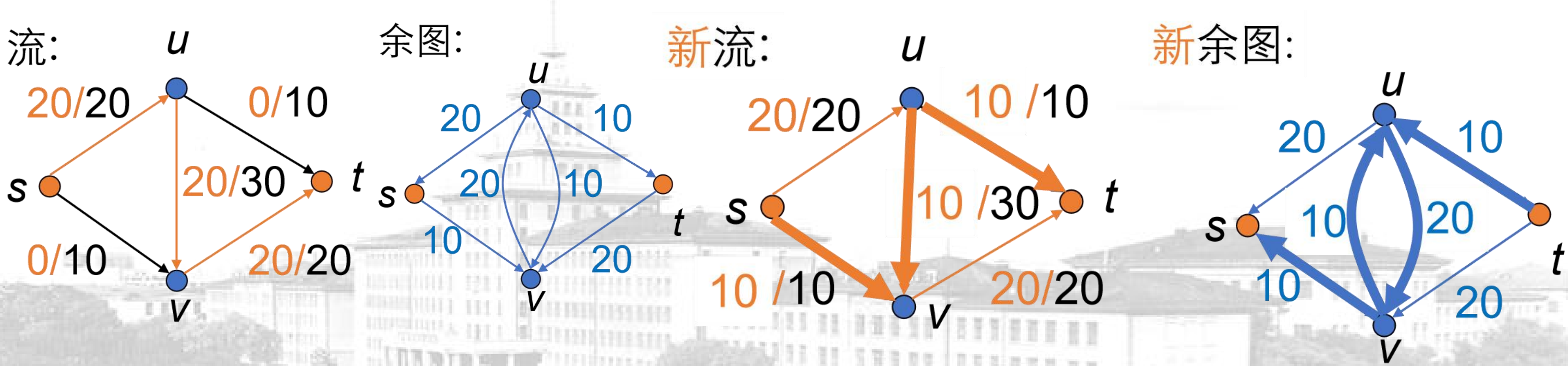
1.对于所有e初始化 f(e) = 0

2.While 余图中存在s-t 路径 P

3. 沿着路径P增广f得到新的f和新的余图

沿着P增广f:找到路径的最小容量，沿着路径修改权重

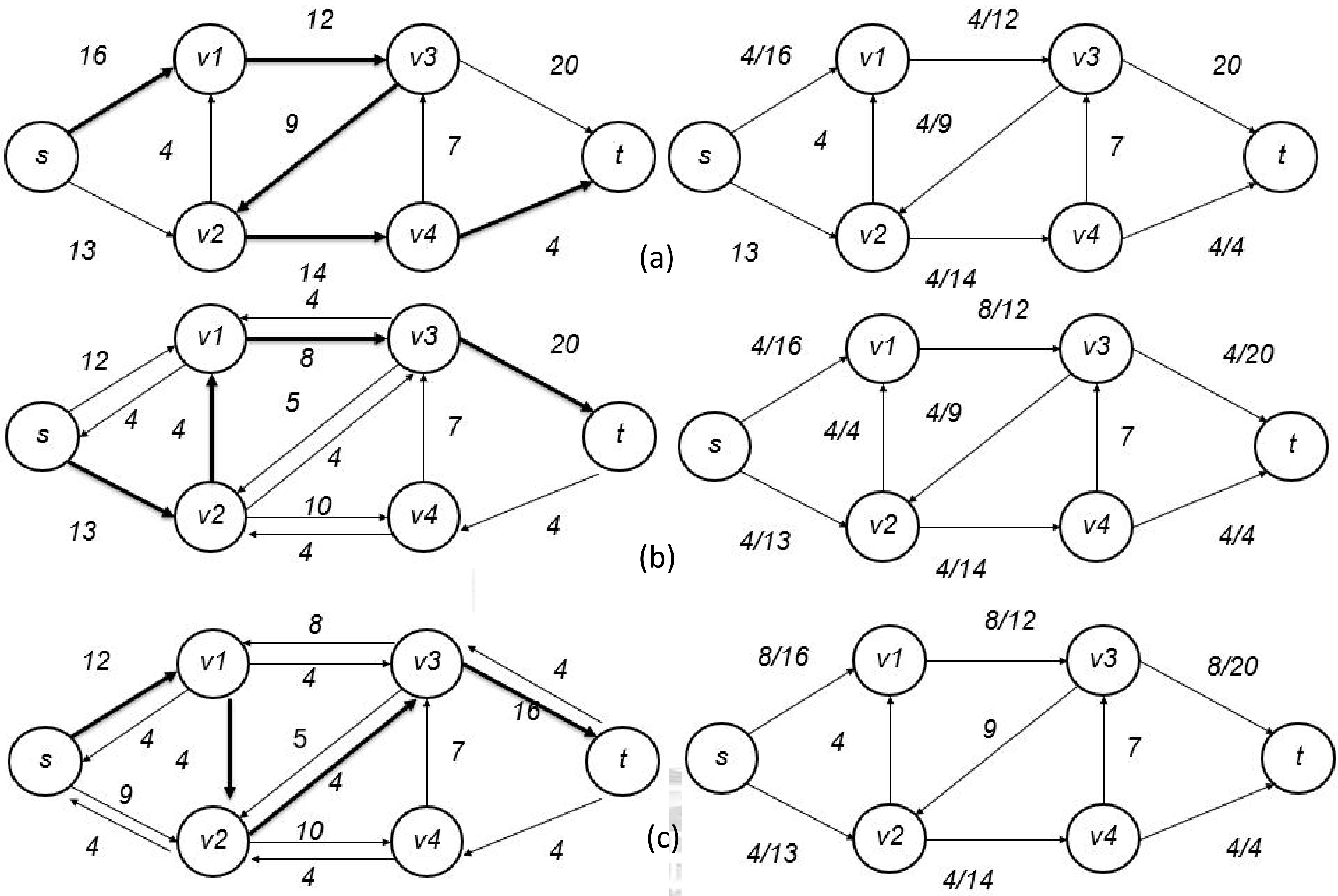
```
Ford-Fulkerson(G,s,t)
    For each edge(u,v) in G.E Do
        f(u,v)=0
    While there exists a path p from s to t in G_f Do
        c_f(p))←min{c_f(u,v):(u,v) is in p}
        For each edge (u,v) in p Do
            If (u,v) in G.E Do
                f(u,v)←f(u,v)+c_f(p)
            Else f(v,u)←f(v,u)-c_f(p)
```



示例

余图

流

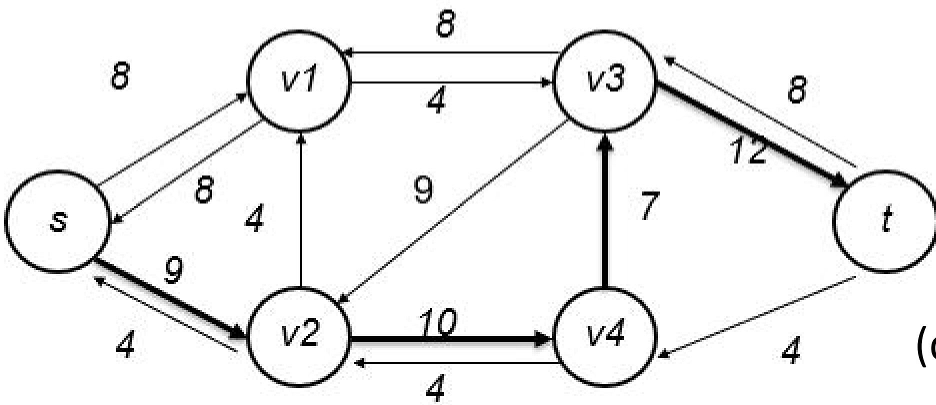


s
v1
v2
v3
v4
t
16
12
20
4
9
7
13
14
4
(a)
4/16
4/12
20
4
4/9
7
13
4/14
4/4
12
4
8
20
4
4
5
4
7
13
10
4
4
(b)
4/16
8/12
4/20
4/4
4/9
7
4/13
4/14
4/4
12
8
4
4
4
16
4
5
4
7
9
4
10
4
4
(c)
8/16
8/12
8/20
4
9
7
4/13
4/14
4/4

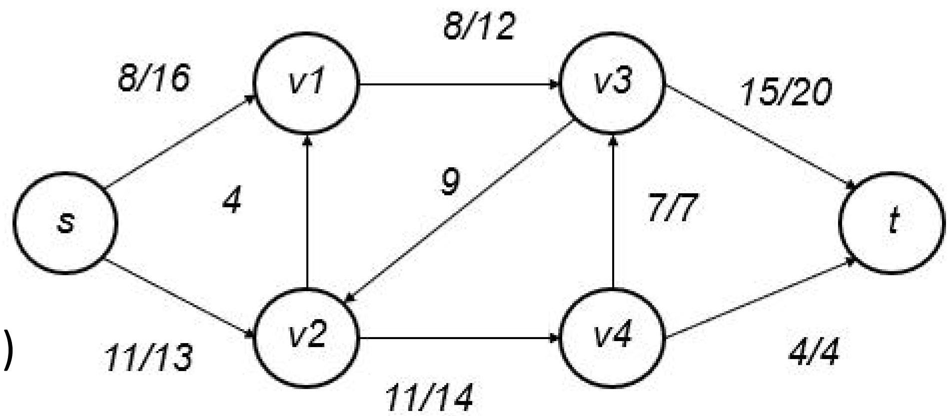
示例

余图

流

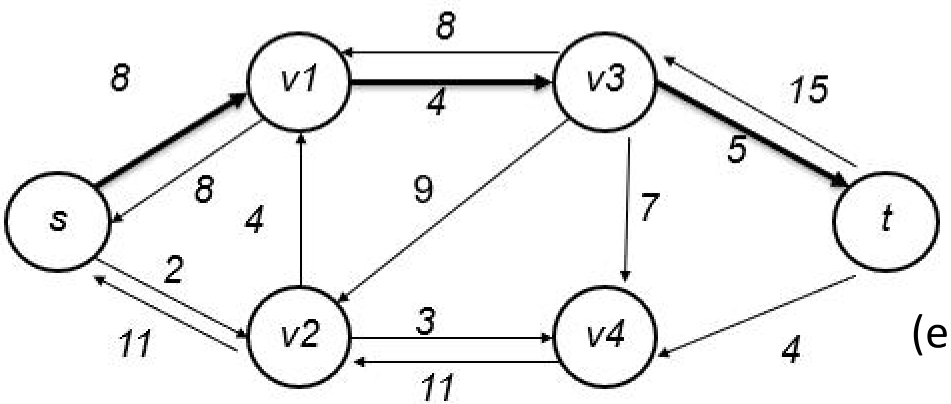


s
v1
v2
v3
v4
t
8
8
4
8
12
9
4
4
9
7
10
4
4

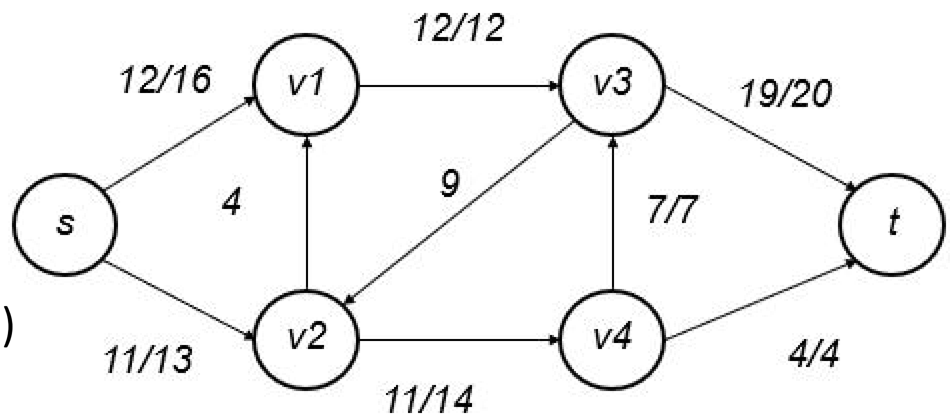


s
v1
v2
v3
v4
t
8/12
8/16
15/20
4
9
7/7
11/13
11/14
4/4

(d)

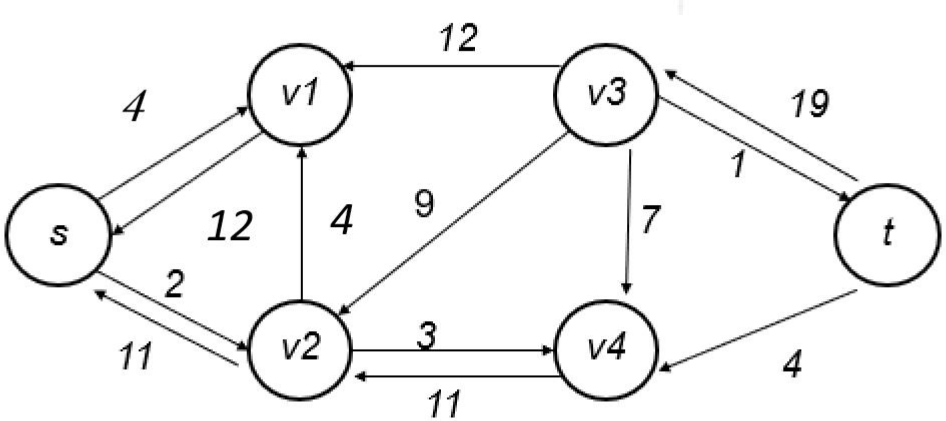


s
v1
v2
v3
v4
t
8
8
4
15
8
5
4
9
7
2
11
3
11
4



s
v1
v2
v3
v4
t
12/12
12/16
19/20
4
9
7/7
11/13
11/14
4/4

(e)



s
v1
v2
v3
v4
t
12
4
19
12
4
9
7
1
2
11
3
11
4

(f)

分析

正确性:

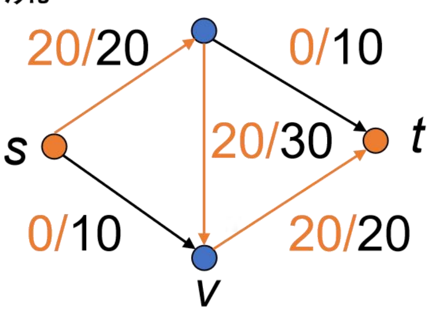
- 得到的是否是最大流（证明不做要求）
- 可终止性– 每次流是一个整数且最少增加1 (假定整数容量)

时间: O(mC)

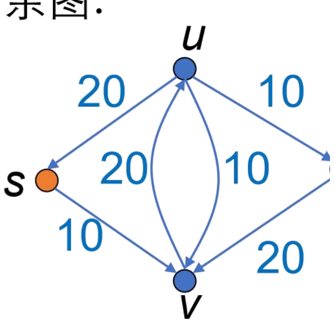
- 最多 C 轮iterations, 其中 C 是最大流的值 m 是每轮边数
- O(m+n) 步，使用 深度优先搜索(DFS) 查找路径 P

```
Ford-Fulkerson(G,s,t)
For each edge(u,v) in G.E Do
    f(u,v)=0
While there exists a path p from s to t in G_f Do
    c_f(p))←min{c_f(u,v):(u,v) is in p}
    For each edge (u,v) in p Do
        If (u,v) in G.E Do
            f(u,v)←f(u,v)+c_f(p)
        Else f(v,u)←f(v,u)-c_f(p)
```

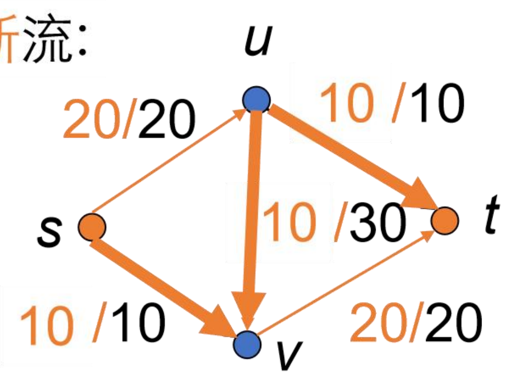
流:



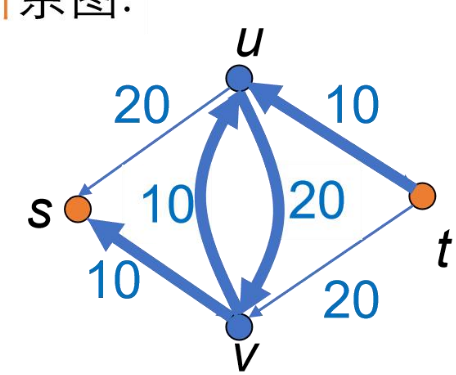
余图:



新流:



新余图:



提纲

8.1 最短路径问题

8.2 网络流问题

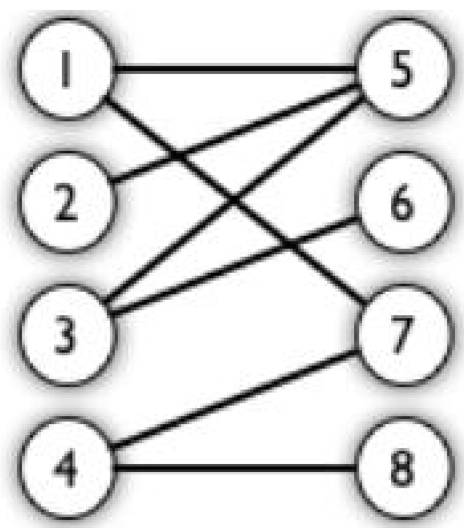
8.3 匹配问题

匹配

- 边集合 $\mathrm{M} \subseteq \mathrm{E}$，M中任意两条边都没有公共顶点。M中的边称为匹配边，M中的点称为匹配点。不在M中的边和点则称为未匹配边，未匹配点。
  - 极大匹配，不存在 $e \notin \mathrm{M}$ 满足 $\mathrm{M} \cup \{e\}$ 也是匹配
  - 最大匹配，$|\mathrm{M}|$ 最大的匹配
  - 完美匹配，$|\mathrm{M}| = n/2$: 每个结点都是M中边的顶点

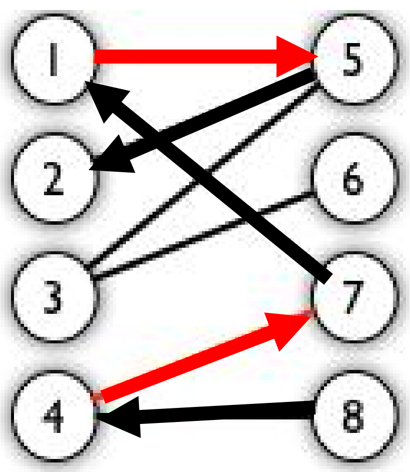
二分图

- 二分图又称作二部图，是图论中的一种特殊模型。 设G=(V,E)是一个无向图，如果顶点V可分割为两个互不相交的子集(A,B)，并且图中的每条边（i，j）所关联的两个顶点i和j分别属于这两个不同的顶点集，则称图G为一个二分图。



交替路和增广路

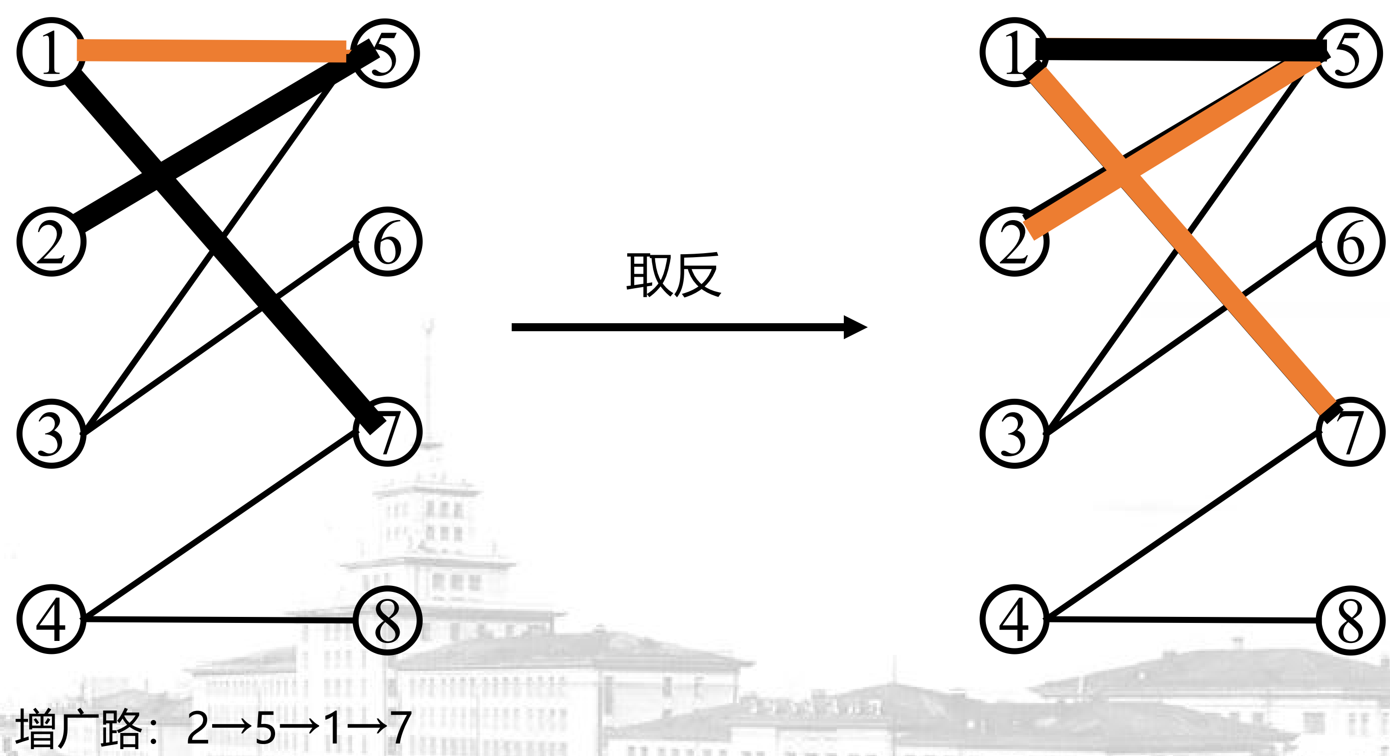
- **交替路**：从一个未匹配点出发，依次经过非匹配边、匹配边、非匹配边...形成的路径叫交替路。
- **增广路**：从一个未匹配点出发，走交替路，如果终点为另一个未匹配点（出发的点不算），则这条交替路称为增广路。



$8 \rightarrow 4 \rightarrow 7 \rightarrow 1 \rightarrow 5 \rightarrow 2$

关于增广路的推论

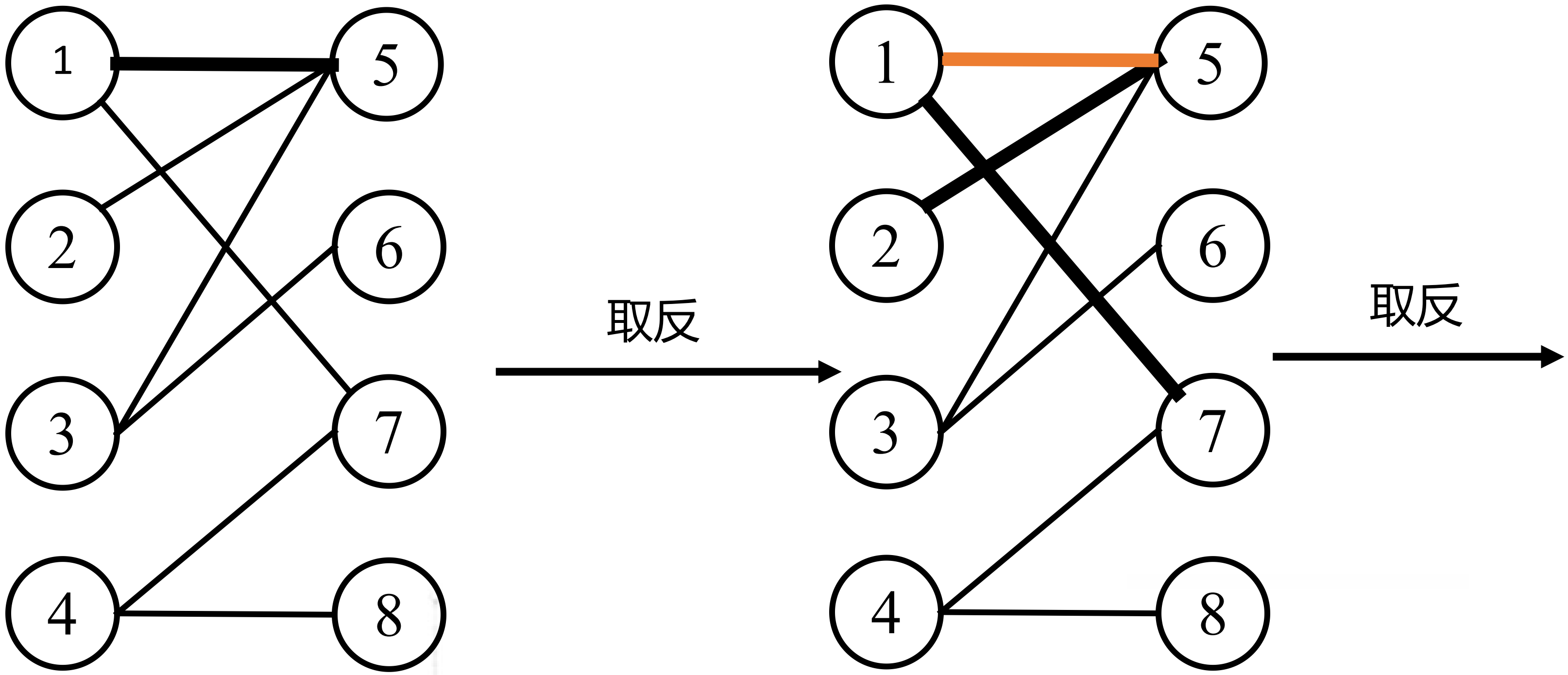
- 增广路的路径长度必定为奇数,第一条边和最后一条边都不属于M。
- 增广路经过取反操作,可以得到一个更大的匹配。
- M为G的最大匹配当且仅当不存在相对于M的增广路径。



匈牙利算法

- 求二分图的最大匹配
- 基本步骤：
  - 1. 置M为空;
  - 2. 找出一条增广路径P,通过取反操作,得到更大的匹配。
  - 3.重复步骤2,直到找不出增广路为止.

示例

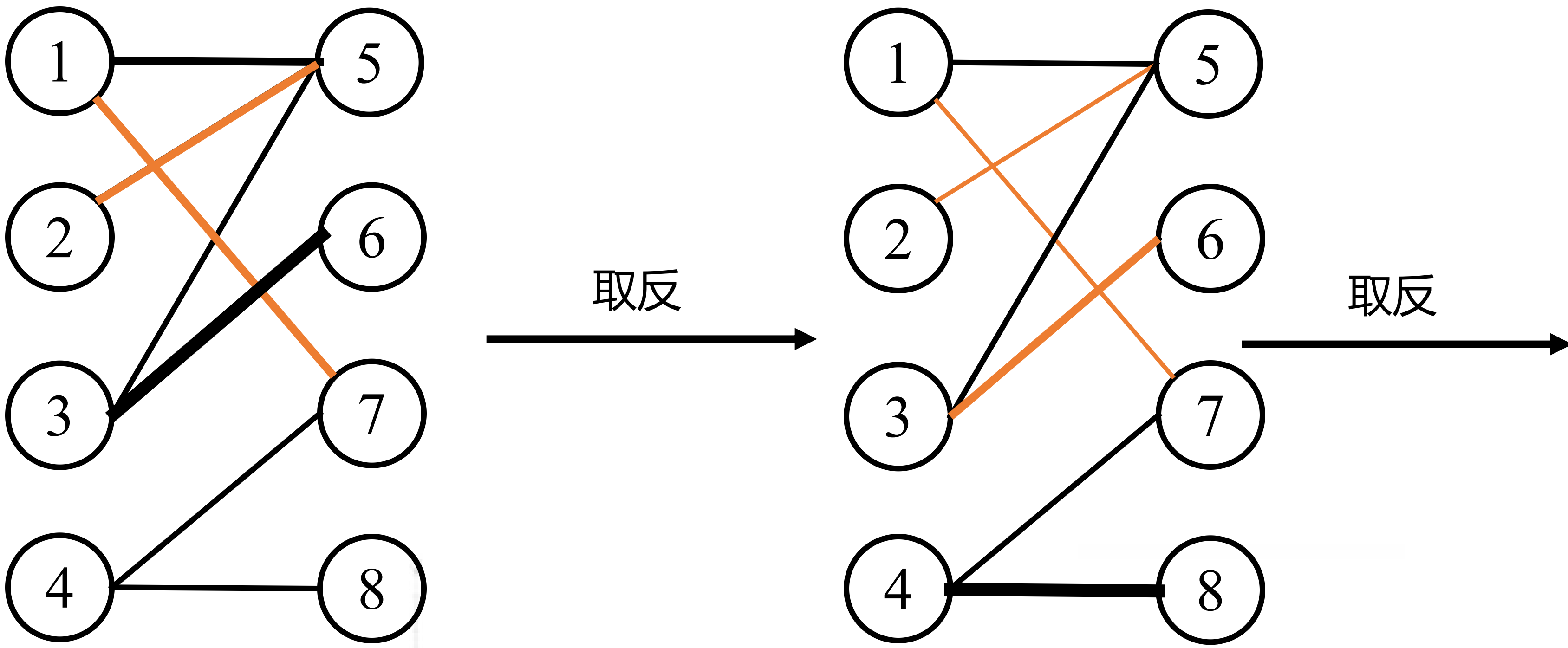


增广路:1→5

M={(1,5)}

增广路：2→5→1→7

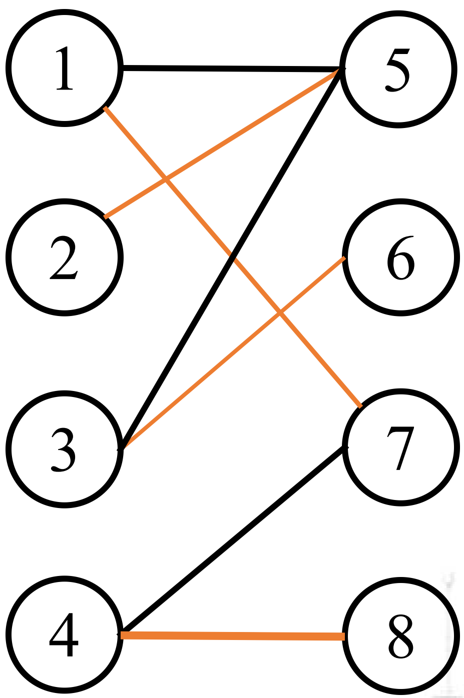
示例



M={(2,5),(1,7)}　　　　M={(2,5),(1,7),(3,6)}

增广路：3→6　　　　增广路：4→8

示例



M={(2,5),(1,7),(3,6),(4,8)}

此时图中已无增广路，故该二分图的最大匹配为4

应用

现要给4个工人A,B,C,D分配任务，每个工人可完成不同的任务，但最多只能接受一个任务。共有4个任务，每个任务也只能分配给一个工人，问最多可以分配多少个任务给工人?

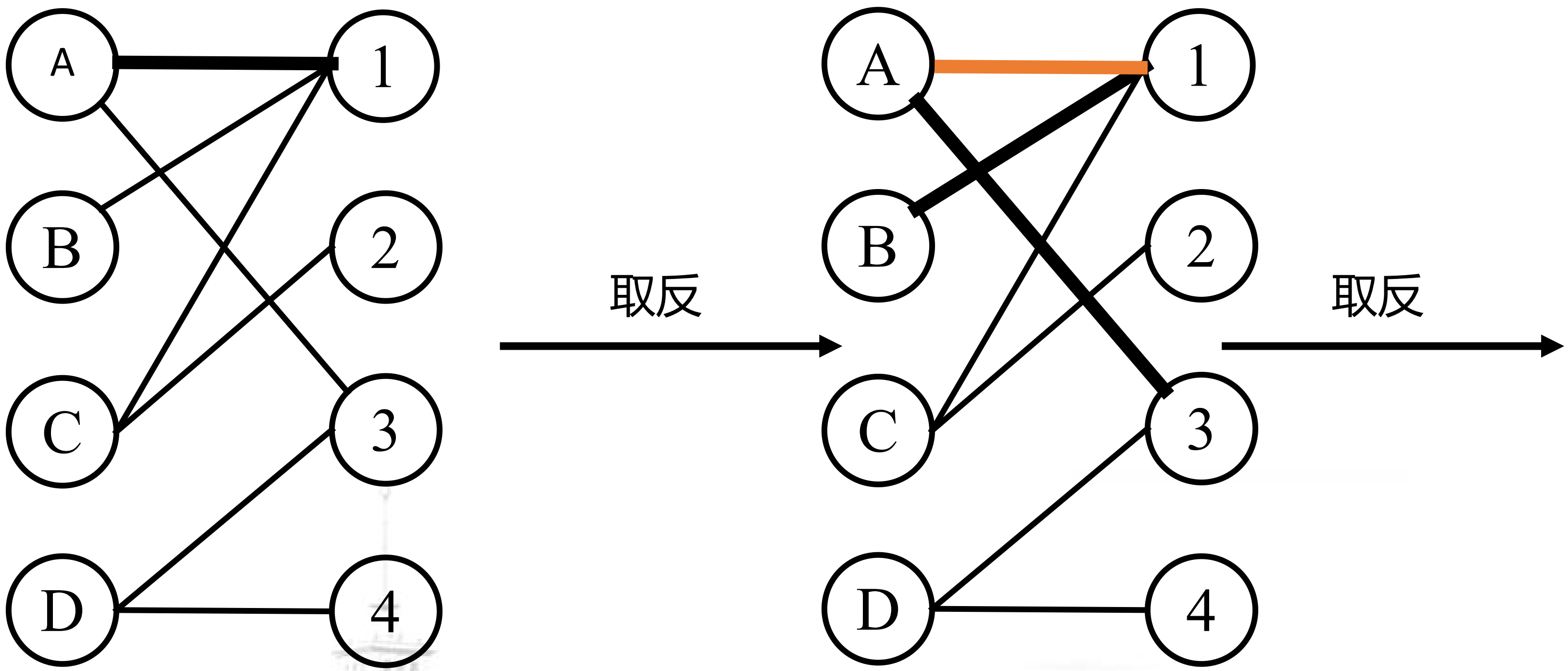
	任务1	任务2	任务3	任务4
A	1	0	1	0
B	1	0	0	0
C	1	1	0	0
D	0	0	1	1

1代表能完成，0代表不能完成

应用

工人和任务可以看作两个不相交的点集合，将工人和他能完成的任务相连，得到一个二分图。因为“每个工人只能接受一个任务，每个任务只能分配给一个工人”，则意味着我们要寻找一个边集合，使得任意两条边没有公共顶点，这就是该图的匹配。“最多可以分配多少个任务”就是要寻找最大匹配，可以用匈牙利算法求解。

应用

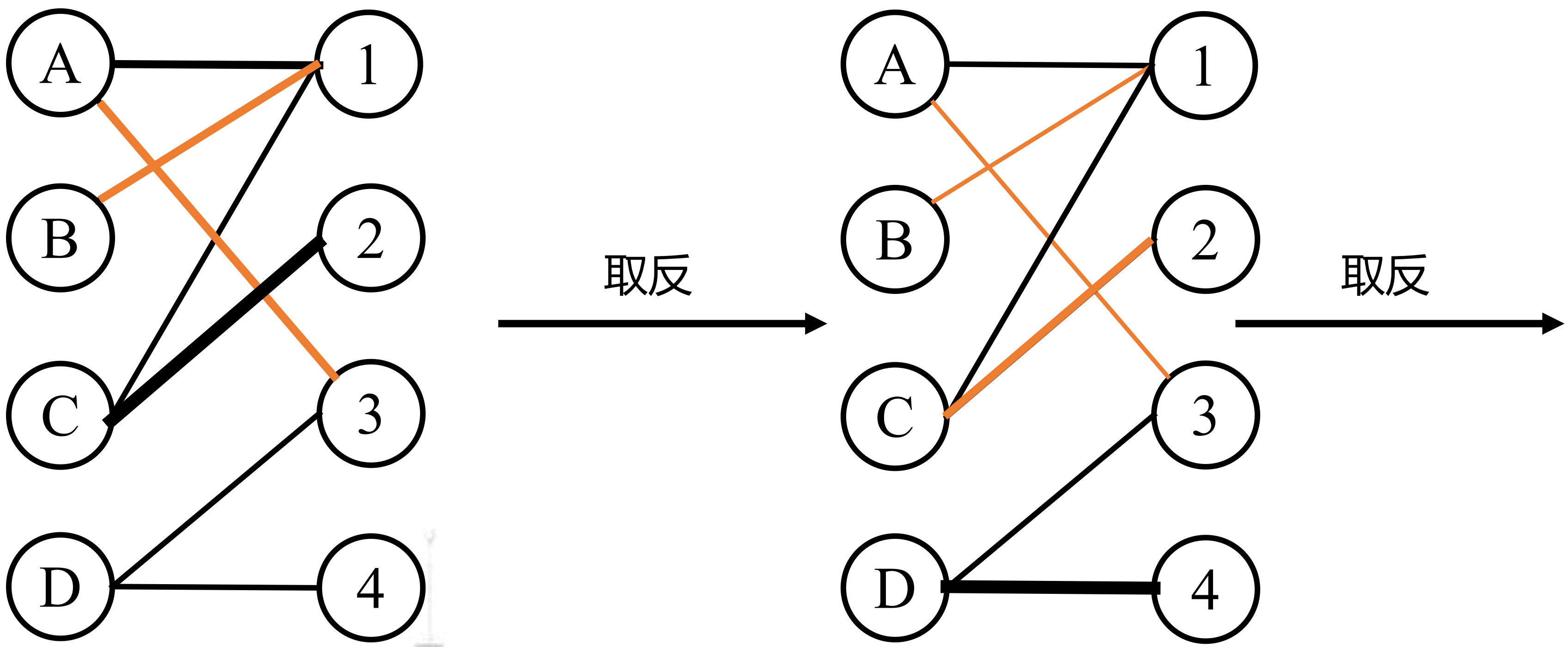


增广路:A→1

M={(A,1)}

增广路：B→1→A→3

应用



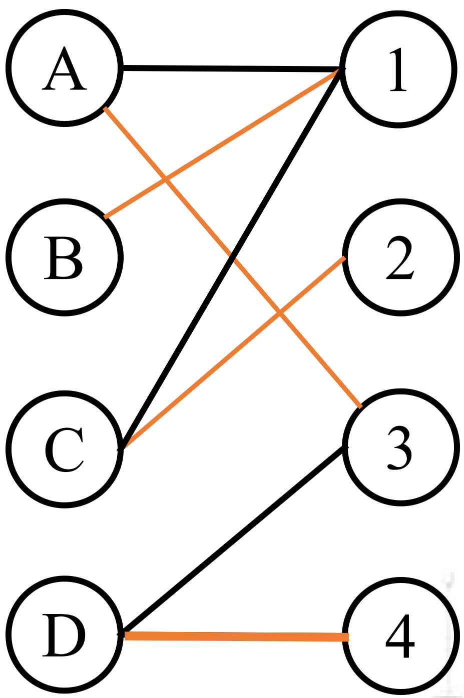
M={(A,3),(B,1)}

增广路：C→2

M ={(A,3),(B,1),(C,2)}

增广路：D→4

应用



M ={(A,3),(B,1),(C,2),(D,4)}

此时图中已无增广路，故该二分图的最大匹配为4。

所以最多能分配4个任务。