Ch7 网络流

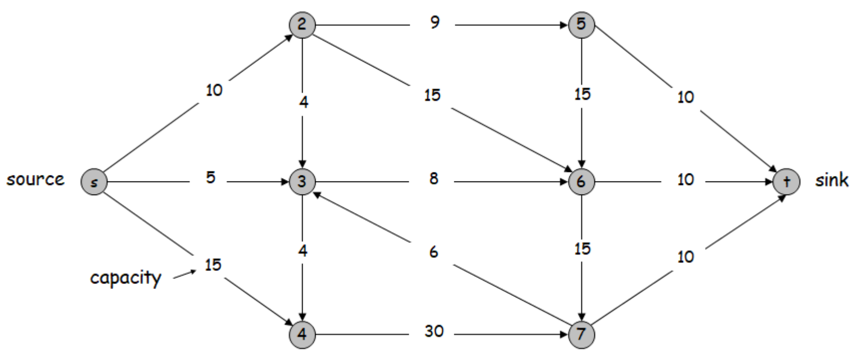
1. 网络流的定义

流网络 Flow network

流网络是在边 edge 上传输材料的一种抽象

- $G=(V,E)$：有向图，没有平行边
- 两个不同的节点
    - s：源节点
    - t：汇节点
- $c(e)$：每个边 e 上的容量 capacity
- 边上的箭头：可行的传输方向

下图为一个流网络

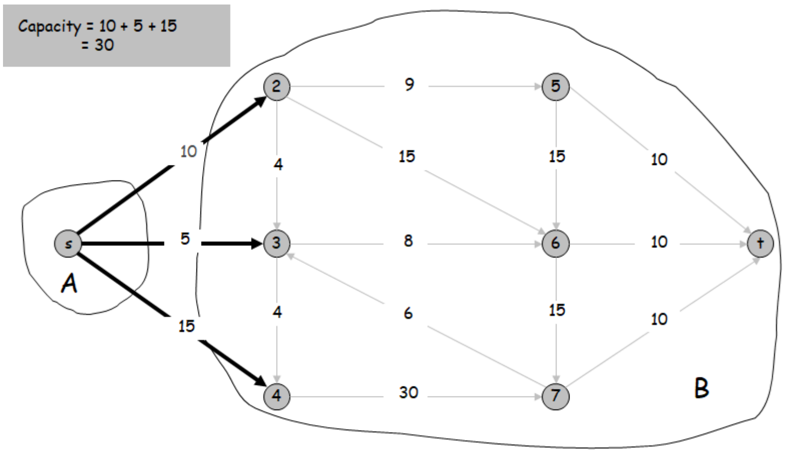


最小割问题

概念介绍

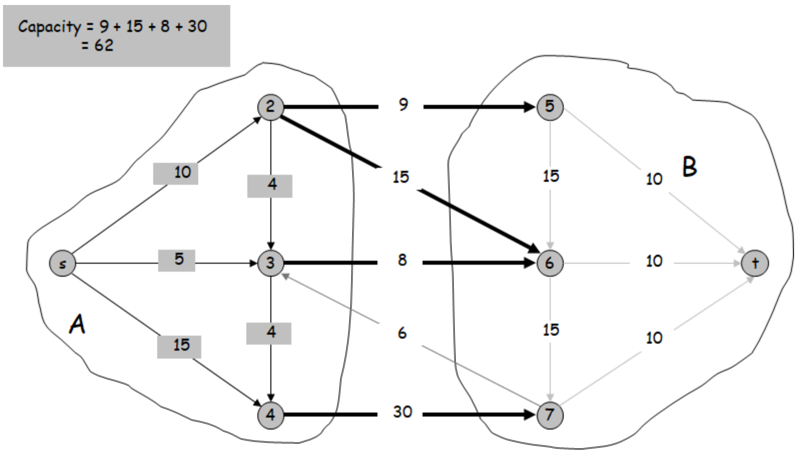
- s-t cut：一个流网络中节点集的划分 (A,B)，使得 $s\in A$ 且 $t\in B$
- $cap(A,B)$：所有从 A 中流入 B 的边上的容量之和
    - $cap(A,B)=\sum_{e\ out\ of\ A} c(e)$

示例1



- A 中只有 s，B 中包含其它所有的节点
- 有 3 条边从 A 指向 B
- 那么 $cap(A, B) = 30$

示例2

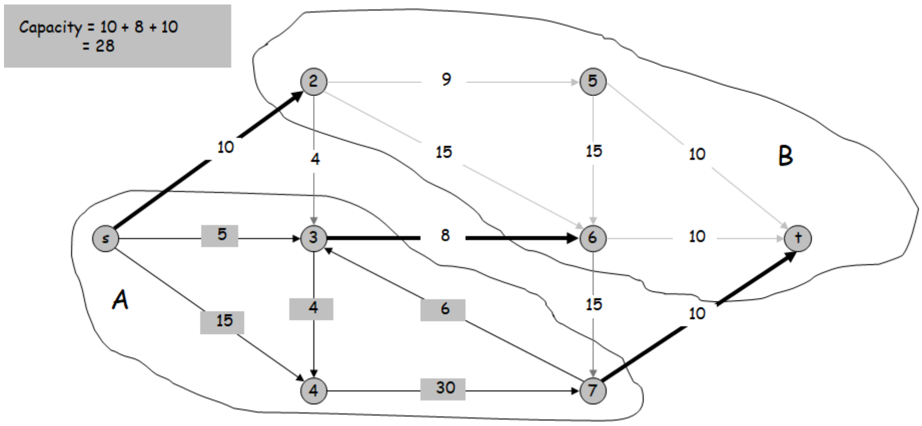


- A 中有 $s, 2, 3, 4$，B 中有 $5, 6, 7, t$
- 有 4 条边从 A 指向 B（注意从 7 指向 3 的不是）
- 那么 $cap(A, B) = 62$

最小割问题定义

求一个 s-t cut 使得 $cap(A, B)$ 最小

在上图的示例中，最小割的答案如下



最大流问题

概念介绍

一个 s-t flow 是对每条边进行赋值，称为边上的流量 flow，用 $f(e)$ 表示

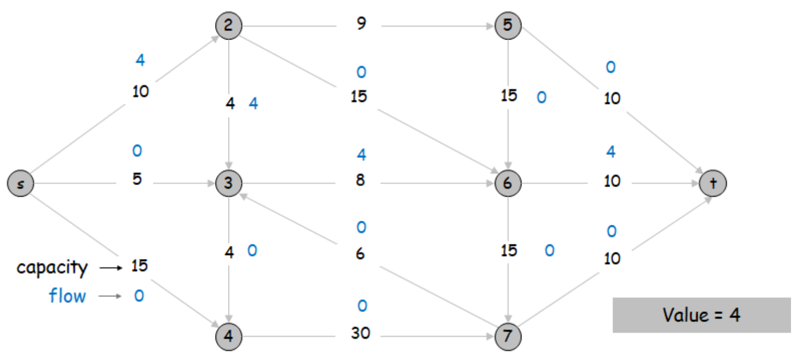
满足以下性质

- $\forall e \in E: 0 \le f(f) \le c(e)$
  每条边上的流量不能大于容量
- $\forall v \in V - \{s, t\}: \sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$
  除了 s 和 t，每个节点流入的流量之和等于流出的流量之和

定义 $v(f)$ 为一个流的值

- $v(f) = \sum_{e \text{ out of } s} f(e) = \sum_{e \text{ into } t} f(e)$
  等于从源 s 流出的流量之和或流入 t 的流量之和

下图为一个示例，满足 s-t 流的定义

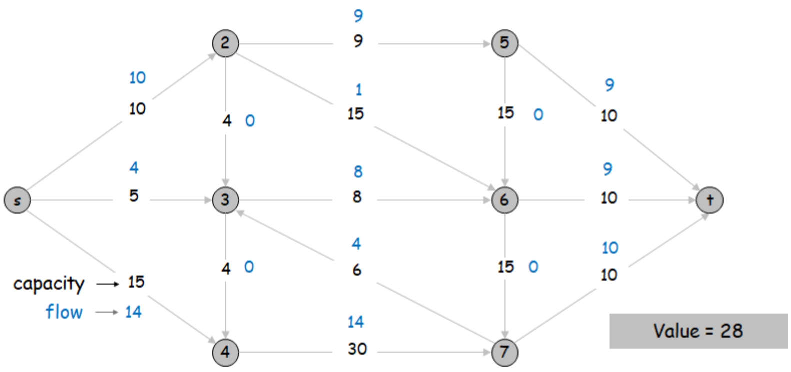


- 黑色的数字为容量 $c(e)$
- 蓝色的数字为流量 $f(e)$
- $v(f) = 4$

最大流问题定义

求一个 s-t flow 使得 $val(f)$ 最大

在上图的示例中，最大流的答案如下



流值引理 flow value lemma

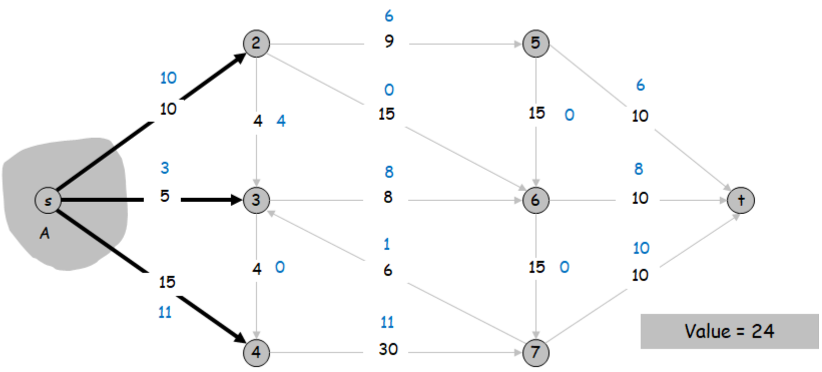
概念介绍

令 f 是任意的流，(A, B) 是任意的 s-t cut

那么定义通过这个割的**流量** value（流出 A 的流量 - 流入 A 的流量）满足

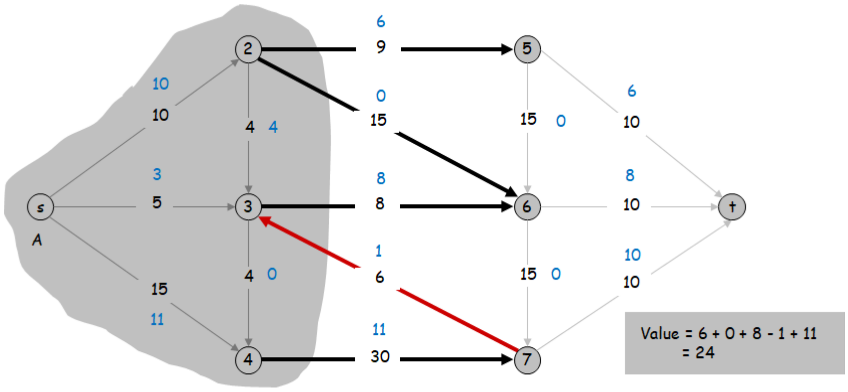
$$value = \sum_{e\ out\ of\ A} f(e) - \sum_{e\ intoA} f(e)$$

示例1



- $\sum_{e\ out\ of\ A} f(e) = 10 + 3 + 11 = 24$
- $\sum_{e\ intoA} f(e) = 0$
- $v(f) = 24 - 0 = 24$

示例2



- $\sum_{e \text{ out of } A} f(e) = 6 + 0 + 8 + 11 = 25$
- $\sum_{e \text{ into} A} f(e) = 1$
- $v(f) = 25 - 1 = 24$

流值引理介绍

令 f 是任意流，令 (A, B) 是任意 s-t cut，则它们都满足

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = v(f)$$

证明

我们已知，f 的流值 $v(f)$ 等于从 s 流出的流量之和

$$v(f) = \sum_{e \text{ out of } s} f(e) = \sum_{e \text{ into } t} f(e)$$

根据流的守恒要求，对于任何不属于 s 或 t 的点

$$\forall v \in V - \{s, t\} : \sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

由于 $s \in A, t \in B$，对于在 A 中的所有点而言，除了 s 之外，都有

$$\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ into } v} f(e) = 0$$

所以上述式子可以改写成

$$v(f) = \sum_{e \text{ out of } s} f(e) = \sum_{v \in A} \left(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ into } v} f(e) \right)$$

将它拆开来恰好可以满足

$$v(f) = \sum_{e \ out \ of \ A} f(e) - \sum_{e \ into \ A} f(e)$$

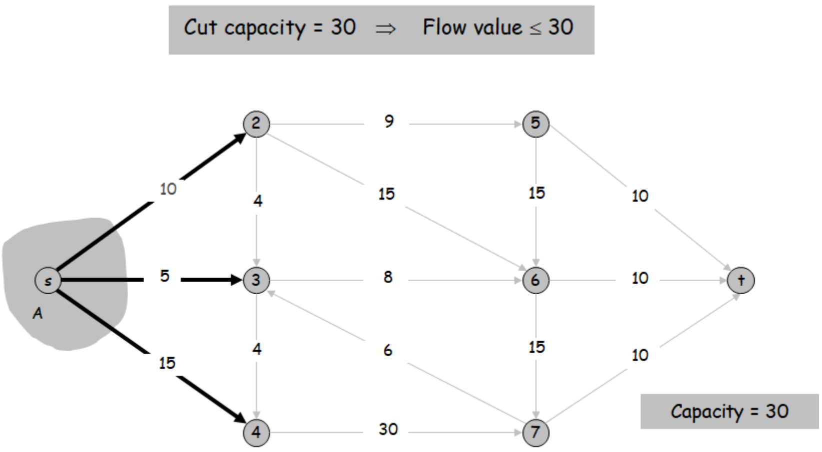
故得证

弱对偶性

弱对偶性定义

令 f 是任意的流，(A, B) 是任意的 s-t cut，则 $v(f) \leq cap(A, B)$

如下图所示



- $cup(A, B) = 30$
- 任意流的流量 $v(f) \leq 30$

证明

我们已经知道了

$$v(f) = \sum_{e \ out \ of \ A} f(e) - \sum_{e \ into \ A} f(e)$$

所以它必然小于等于从 A 流出的量，进一步小于 A 的容量

$$v(f) \leq \sum_{e \ out \ of \ A} f(e) \leq \sum_{e \ out \ of \ A} c(e) = cap(A, B)$$

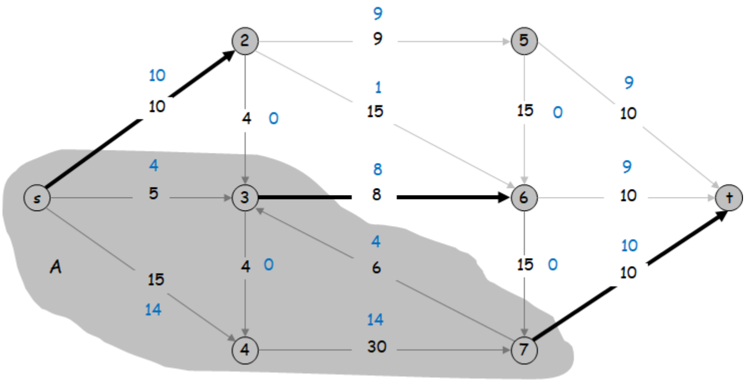
故得证

最优性条件

由弱对偶性，我们知道

若 f 是任意流，(A,B) 是任意 s-t 割，如果 $v(f)=cap(A,B)$，则 f 是最大流，且 (A,B) 是最小割

如下图所示



$$v(f)=\sum_{e\ out\ of\ A} f(e)-\sum_{e\ into\ A} f(e)=10+8+10-0-0=\sum_{e\ out\ of\ s}=28$$

$$cap(A,B)=\sum_{e\ out\ of\ A} c(e)=10+8+10=28$$

故此时 $v(f)=cap(A,B)$

- f 是最大流
- (A,B) 是最小割

2. 最大流算法

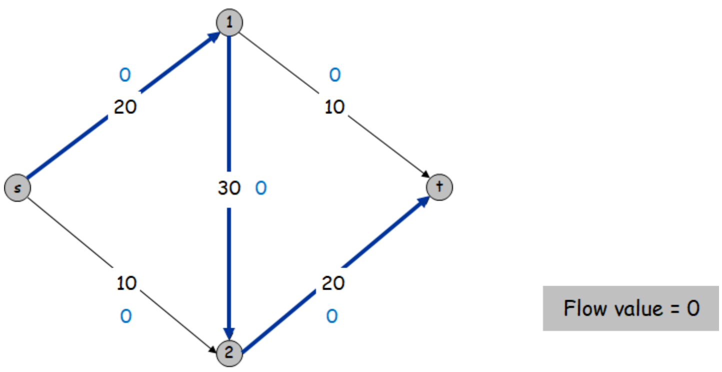
贪心（非最优解）

我们首先尝试贪心的算法

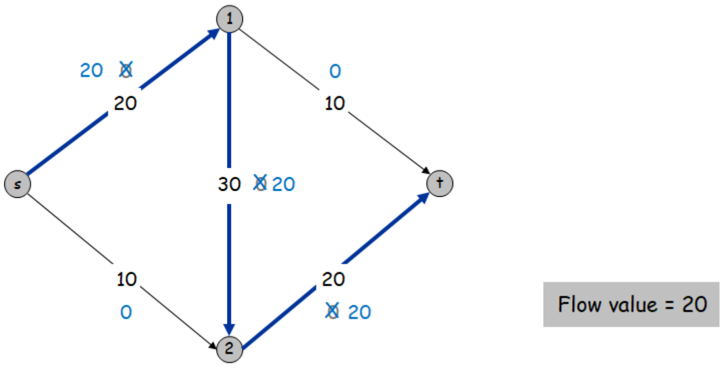
```
初始状态下所有边的流量 f(e) = 0
找到一条从可以 s 到 t 的路径 p，它边上的最小流量 min(f(e)) > 0
扩张这条增广路径 P
重复直到再也找不到这样的路径
```

示例

初始状态如下：

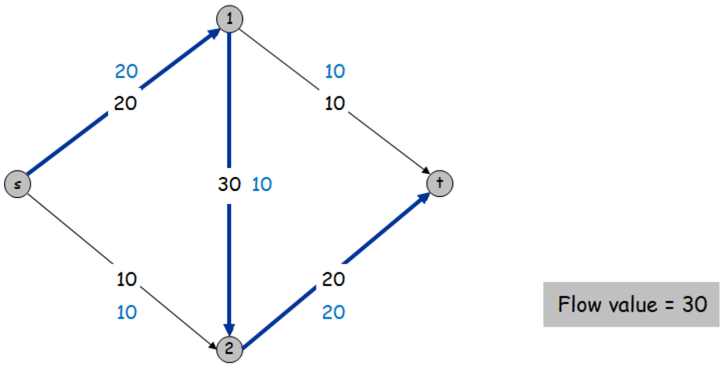


找到一条路径 $P = s \to 1 \to 2 \to t$，其 $min(f(e)) = 20$



此后，不能再找到一条增广路径 P，算法结束，得到的 $v(f) = 20$

事实上，最优结果的 $v(f) = 30$

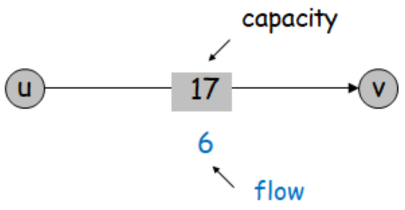


故贪心的算法并不是最优解

残量网络 Residual Graph

在原网络中

对于任何一个边 $e = (u, v) \in E$ 有如下两个属性

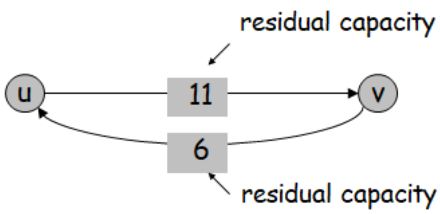


- $f(e)$：边的流量
- $c(e)$：边的容量

残量网络中，将两个点之间的边扩展为两条

- $e=(u,v)$：原网络的边
    - 它的边权 $c_f(e)=c(e)-f(e)$
- $e^R=(v,u)$：原网络的边的反向
    - 它的边权 $c_f(e^R)=f(e)$

示例



- 原网络边 $e=(u,v)=c(e)-f(e)=17-6=11$
- 反向边 $e^R=(v,u)=f(e)=6$

残量网络 $G_f=(V,E_f)$

- E_f 为 e 和 e^R 中边权大于 0 边的集合

Ford-Fulkerson 算法

在残量网络上执行贪婪算法，即为 Ford-Fulkerson 算法

算法伪代码如下：

增广路的更新

- $c_f[]$：边权
- P：增广路径
- f：计算的最大流值

```
ARGUMENT(c_f[], P, f):
找到一条 s-t 的增广路径 P
b = P 中最小的容量 min(c_f[e]) (e ∈ P)
f ← f + b
for(边 e ∈ P){
    if(e ∈ 原网络的边){
        c_f[e] = c_f[e] - b
    }else(e ∈ 原网络的边的反向边){
        c_f[e] = c_f[e] + b
    }
}
return c_f[]
```

Ford-Fulkerson 算法

- G：初始有向图
- s：源节点
- t：终节点
- $c[]$：边的容量

```
FORD-FULKERSON(G,s,t,c[])
// 构建残存网络
for(边 e ∈ 原网络的边){
    c_f[e] = c[e]
}
for(边 e ∈ 原网络的边的反向边){
    c_f[e] = 0
}
// 初始化最大流 f
f ← 0
while(残量网络 G_f 中仍然存在 s-t 增广路径 P){
    ARGUMENT(c_f[],P,f)
    更新 G_f
}
return f
```

简写成如下形式

- f：流量矩阵
- c：容量矩阵
- P：增广路径

- s：源节点
- t：终节点
- G：初始图

```
Argument(f,c,P){
    b ← bottleneck(P)
    foreach e ∈ P{
        if(e ∈ E) f(e) ← f(e) + b
        else       f(e^R) ← f(e^R) - b
    }
    return f
}

Ford-Fulkerson(G,s,t,c){
    foreach e ∈ E f(e) ← 0
    G_f ← residual graph

    while(there exists argumenting path P){
        f ← Argument(f,c,P)
        update G_f
    }
}
```

证明

流 f 是最大流当且仅当没有增广路径 P，即二者互为充分必要条件

证明顺序为

1. 存在一个割 (A,B) 使得 $v(f)=cap(A,B)$
2. 流 f 是最大流
3. 此时 f 不包含增广路径 P

对于证明 1 到 2

- 即弱对偶定理的推论

对于证明 2 到 3

- 证明它的逆否命题：令 f 是一个流，如果存在一条增广路径 P，那么我们可以沿着增广路径进行增广，此时 f 不是最大流
- 根据增广路径 P 的定义，可以轻易证明

对于证明 3 到 1

- 令 f 是一个流，且不存在增广路径 P
- 令 A 是残量网络中 s 可达的节点的集合（即原边的边权 $c_f(e)>0$ 的部分），其余节点构成 B，形成割 (A,B)
- 由 A 的定义，有 $s\in A, t\notin A$
- $v(f)=\sum_{e\ out\ of\ A} f(e)-\sum_{e\ into\ A} f(e)=\sum_{e\ out\ of\ A} c(e)=cap(A,B)$

时间复杂度

假设 n 个节点，所有边的容量在整数 1-C 之间

我们构造一个集合 A，使得只有 $s\in A$，那么最大值流量 $v(f*)\le nC$（s 与除了 t 之外其它 $n-2$ 个节点都相连，且边容量都为 C）

- 每次更新 G_f 需要 $O(E_f)=O(E)$ 的时间
- 每次找到一条 s-t 增广路径 P 需要 $O(V+E_f)=O(V+E)=O(E)$ 的时间

每次增广 $v(f)$ 至少增加 1，所以算法最多循环 $f*$ 次

故整体算法的时间复杂度为 $O(E\cdot|f*|)$

Edmonds-Karp 算法

Edmonds-Karp 算法是对 Ford-Fulkerson 算法的优化

```
Argument(f,c,P){
    b ← bottleneck(P)
    foreach e ∈ P{
        if(e ∈ E) f(e) ← f(e) + b
        else       f(e^R) ← f(e^2) - b
    }
    return f
}

Ford-Fulkerson(G,s,t,c){
    foreach e ∈ E f(e) ← 0
    G_f ← residual graph

    while(BFS(G_f) and can find a path P from s to t){
        f ← Argument(f,c,P)
        update G_f
```

```
    }
}
```

使用 BFS 找到一个增广路径 P，因为 BFS 的特点是找到一条最短路径

时间复杂度被优化到 $O(E^2 \cdot V)$

3. 二部图匹配

匹配定义

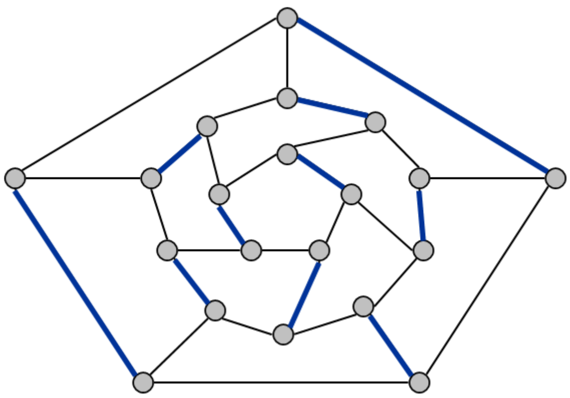
输入：无向图 $G = (V, E)$

称 M 是一个匹配，$M \in E$

- 每个节点最多出现在 M 的一条边中

最大匹配就是求包含边数最多的匹配

示例



- 上图蓝色的边构成了一个匹配 M，包含 10 条边

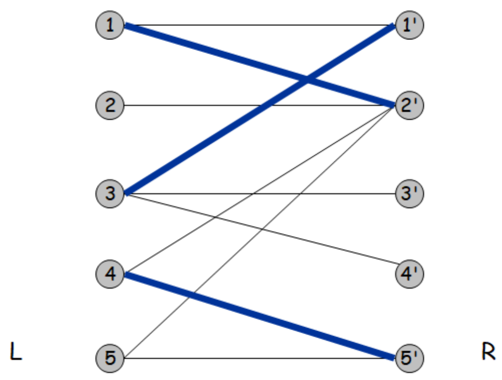
二部图匹配

输入：二部图 $G = (L \cup R, E)$，每条边的两个端点分别属于 L 和 R

称 M 是一个二部图匹配，$M \in E$

- 每个节点最多出现在 M 的一条边中

示例



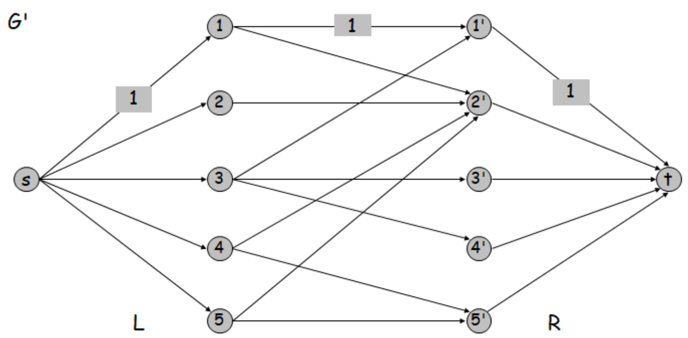
- 上图蓝色的边构成了一个匹配 M，包含 3 条边

算法（最大流算法）

创建一个图 G'

- 它包含了二部图的节点
- 以及添加一个源点 s 和一个终点 t
- s 指向 L 中所有的节点，R 中所有的节点指向 t
- 给所有的边赋容量 $c = 1$

如图所示

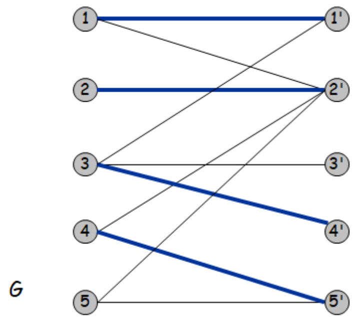


在 G' 上运行最大流算法，将得到 G 的最大匹配 = G' 的最大流值 $v(f)$

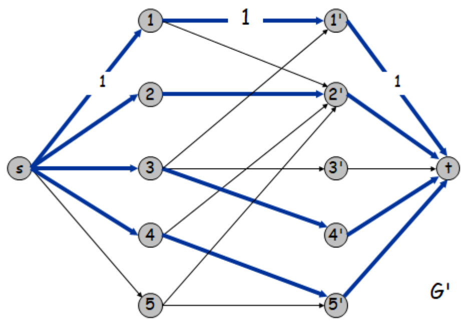
证明

证明最大匹配 ≤ 最大流值

- 给定一个最大匹配 M，它的最大匹配值为 k



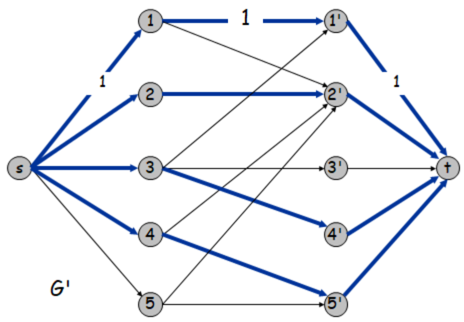
- 如果将图添加为 G' 时，由于每个节点 $v_L \in L$ 与 s 相连，$v_R \in R$ 与 t 相连



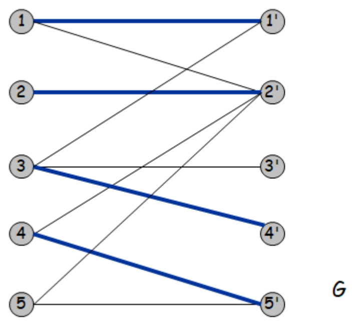
- 我们必然可以得到一个流的值 $v(f) = k$

证明最大匹配 ≥ 最大流值

- 令 f 是 G' 上的最大流，$v(f) = k$



- 由于 k 是整数，并且在每条边上的流量是 0 / 1
- 令 M 是从 L 出发，到 R 的边的集合，则该集合中 $f(e) = 1$
- 由于 L 和 R 中的点最多在 M 中出现 1 次，并且包含 k 条边



- 因此，最大匹配 ≥ 最大流值

故得证

时间复杂度

最大流算法

- 一般最大流算法 $O(E \cdot |f*|) = O(E \cdot V)$（$f* = V$）
- 变尺度计数 $O(E^2\ log|f*|) = O(E^2\ logV) \approx O(E^2)$

最短增广路径

- $O(E \cdot V^{\frac{1}{2}})$