《算法设计与分析》

10-网络流 (Network Flow)

杨启哲

上海师范大学信机学院计算机系

2023 年 11 月 21 日

主要内容

- 网络流问题简介
- Ford–Fulkerson 算法
- Edmonds–Karp 算法
- Dinic 算法
- 从线性规划的角度理解最大流最小割定理

网络流问题简介

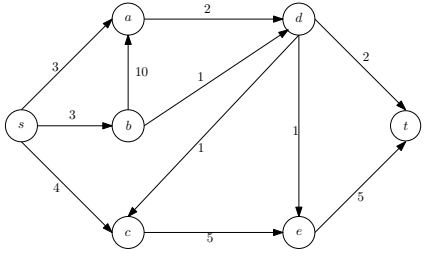
流网络的引入 (I)

有向图给了我们许多解决实际问题的方法，比如我们可以将交通地图转化为有向图来找到两地之间的最短路径。

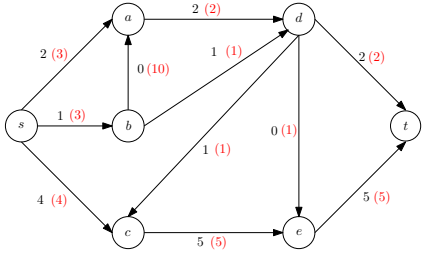
我们也可以用有向图来解决一些更加复杂的问题，比如物流运输问题。

问题 1 **[石油运输问题].**

假设左图是一个输送石油的管道网络，我们的目标是利用该网络从源点 s 到汇点 t 输送尽可能多的石油。每条管线都有自身的容量，并且在输送的过程中原油无法被存储。



流网络的引入 (II)



上图便是一个实际的例子。

- 每条边上的红色数字表示该边的容量，黑色数字表示该边上的流量。
- 除了源点 s 和汇点 t 意外，每个点进去的流量和出去的流量相等。
- 这个例子表明，我们可以从源点 s 到汇点 t 输送 7 单位的石油。

我们称这样的有向图为流网络 (flow network)，其中每条边都有一个容量 (capacity)，表示该边能够承载的最大流量。

流网络的定义

我们现在给出形式定义。

定义 2 **[流网络].**

流网络 $G = (V, E, c)$ 是一个连通的有向图，图中每条边 $c : E \to \mathbb{R}^{\geqslant 0}$ 表示边的容量值，即 $c(u, v)$ 表示从 u 到 v 的边的容量，为了方便起见，定义 $c(u, v) = 0$ 如果 $(u, v) \notin E$。图中有两个特殊节点: 源节点 s 和汇节点 t。

补充说明

1. 我们可以假设流网络中不存在自环，即对于任意的节点 v，$(v, v) \notin E$。
2. 我们同样可以假设流网络中不存在重边，即对于任意的节点 u, v，如果 $(u, v) \in E$，那么 $(v, u) \notin E$。
3. 我们还可以假设流网络中的任何一个节点都存在于某个 s 到 t 的路径上。

流的定义

我们再来定义流的概念。

定义 3 **[流].**

给定一个流网络 $G = (V, E, c)$，一个流 f 是一个函数 $f : E \to \mathbb{R}^{\geqslant 0}$，满足以下两个性质:

1. 容量限制: 对于任意的一对顶点 $(u, v) \in V \times V$，$f(u, v) \leqslant c(u, v)$。
2. 流守恒: 对于任意的节点 $v \in V \setminus \{s, t\}$，$\sum_{u \in V} f(u, v) = \sum_{u \in V} f(v, u)$。

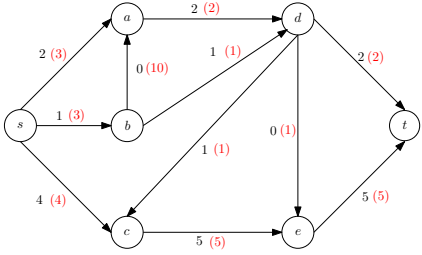
我们称 $f(u, v)$ 为边 (u, v) 上的流量。定义流 f 的值为

$$|f| = \sum_{u \in V} f(s, u) - \sum_{u \in V} f(u, s) = \sum_{u \in V} f(u, t) - \sum_{u \in V} f(t, u)$$

一般来说，s 的入度为 0,t 的出度也为 0, 所以 $|f| = \sum_{u \in V} f(s, u) = \sum_{u \in V} f(u, t)$。

流 f 也被称作 s 到 t 的流。

流的例子



上述图边上的黑色字实际上给了一个流的例子，我们记作 f。

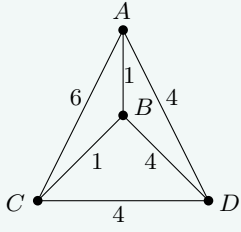
- $|\mathrm{f}| = \mathrm{f}(\mathrm{s},\mathrm{a}) + \mathrm{f}(\mathrm{s},\mathrm{b}) + \mathrm{f}(\mathrm{s},\mathrm{c}) = \mathrm{f}(\mathrm{d},\mathrm{t}) + \mathrm{f}(e,\mathrm{t}) = 7.$
- 这样一个流其实就代表了一种运输过程。从而原问题便可以转化为求 s 到 t 的最大流。

问题 4.

假设现在有 A, B, C, D 四个队伍参加一个锦标赛，目前比赛情况如下:

$$\mathrm{A}:40,\ \mathrm{B}:38,\ \mathrm{C}:37,\ \mathrm{D}:29$$

这里的数字表示队伍当前已经获胜的比赛数，每个队伍还剩下的比赛场次如下图所示:



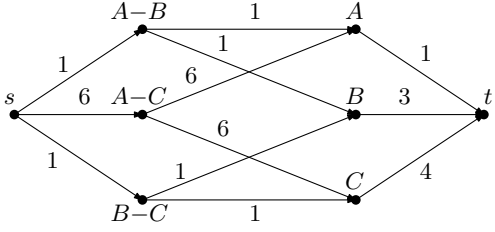
问题是，队伍 D 是否还可能获胜?

流网络的运用-获胜预测 (II)

假设 D 剩下的比赛都能获胜，要想得到冠军则：

- A 至多再获胜 1 场。
- B 至多再获胜 3 场。
- C 至多再获胜 4 场。

我们将其转换成一个流网络。



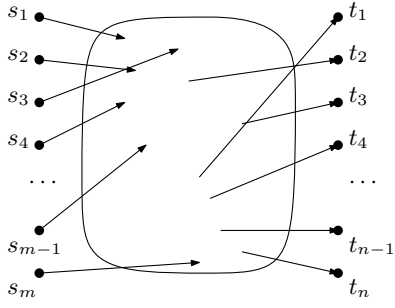
事实 5.

D 能获胜当且仅当上述流网络的最大流为 8。

流网络的运用-多源点和多汇点的流网络 (I)

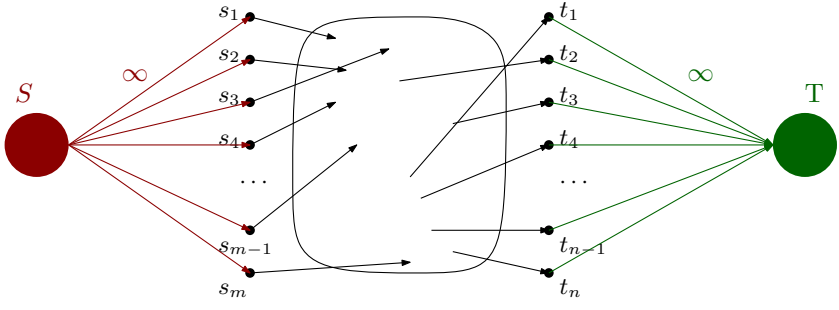
上海师范大学 Shanghai Normal University

上述我们讨论了单源点和单汇点的流网络，而事实上一些运输网络可能有多个源点和多个汇点，这该如何处理?



将所有的源点看成一个源点，所有的汇点看成一个汇点!

流网络的运用-多源点和多汇点的流网络 (II)

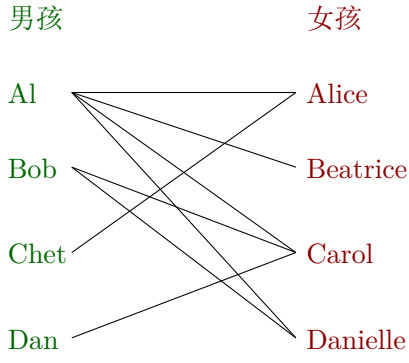


事实 6.

$s_1, \ldots s_m$ 到 $t_1, \ldots t_n$ 的最大流等价于 S 到 T 的最大流。

流网络的运用-图的最大匹配 (I)

假设在一次相亲活动中，一共有如下 4 个男孩和女孩。下图表示了他们之间的好感度，如果男孩和女孩存在一条边，则意味着他们彼此喜欢，比如男孩 AI 喜欢所有的女孩。



问题是，是否存在一种配对方式，使得所有的男孩和女孩都能两两配对，并且配对双方总是互相喜欢？如果不行的话，至多能配对成几对互相喜欢的男孩和女孩？

流网络的运用-图的最大匹配 (II)

我们可以将上述问题先概括成二分图的匹配问题。

定义 7 **[二分图的匹配].**

给定一个二分图 $G = (V_1 \cup V_2, E)$，令 M 是 E 的子集，如果 M 中任意两边都没有公共顶点，则称 M 是 G 的一个匹配。

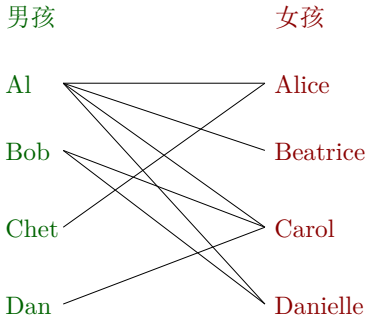
不同的匹配

对于二分图 $G = (V_1 \cup V_2, E)$ 上的一个匹配 M，我们称其

- 最大匹配，如果这是一个包含最多边的匹配。
- 极大匹配，如果往 M 中再加任何一条边都不能构成匹配。
- 完备匹配，$|M| = |V_1| \leqslant |V_2|$，即一个匹配能够覆盖 V_1，V_2 中较小集合的所有顶点。

很显然，一个完备匹配肯定是最大匹配。

流网络的运用-图的最大匹配 (III)

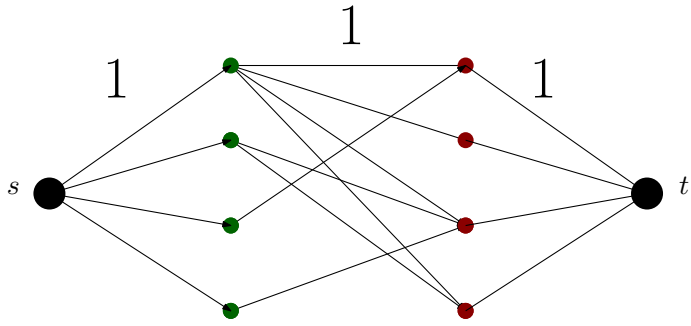


- $(\mathrm{Al}, \mathrm{Alice}), (\mathrm{Chet}, \mathrm{Alice}), (\mathrm{Bob}, \mathrm{Carol})$ 不是一个匹配。
- $(\mathrm{Al}, \mathrm{Beatrice}), (\mathrm{Chet}, \mathrm{Alice}), (\mathrm{Bob}, \mathrm{Carol})$ 是一个极大匹配。
- $(\mathrm{Al}, \mathrm{Beatrice}), (\mathrm{Chet}, \mathrm{Alice}), (\mathrm{Bob}, \mathrm{Danielle}), (\mathrm{Dan}, \mathrm{Carol})$ 是一个完备匹配。

显然完备匹配的存在等价于所有的男孩和女孩都能两两配对，并且配对双方总是互相喜欢。

流网络的运用-图的最大匹配 (IV)

求图的最大匹配问题可以用匈牙利算法解决，但我们使用流网络的视角来解决。



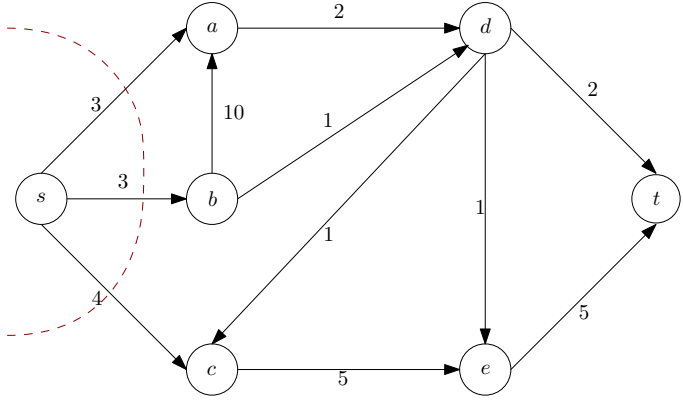
事实 8.

原图的最大匹配数等于上述图中 s 到 t 的最大流。

Ford–Fulkerson 算法

最大流的限制-一个点能出去的最大容量?

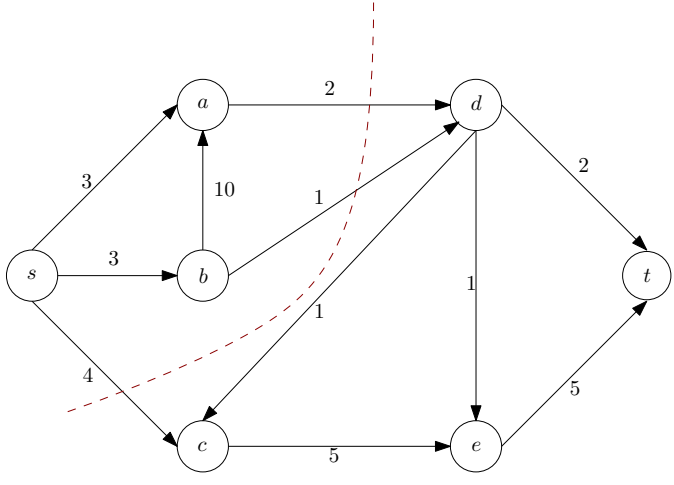
现在让我们回到最大流的问题当中。一个很自然的发现是，一个流网络的最大流会被源点 s 出边的容量所限制。



- 任何一个流 f 的值都不会超过 $3+3+4=10$.

最大流的限制-图的切割 (I)

更进一步，如果我们将图划分成两部分:



最大流的限制-图的切割 (II)

- 一部分是 s 可达的，比如 $S = \{s, a, b\}$
- 剩下一部分记作 T，即 $T = \{c, d, e, t\}$

可以看到横跨 S 到 T 的边的容量最多为 $2 + 4 + 1 = 7$，我们断言任何一个 s 到 t 的最大流都不会超过 7。

> **一个观察**
>
> 直观上可以感受，如果把图分成了类似上述的两部分，则任何一个流都不可能超过横跨这两部分的容量之和。
>
> 下面我们来形式化的叙述这件事。

割的定义

上海师范大学 Shanghai Normal University

定义 9 **[割].**

给定一个流网络 $G = (V, E, c)$，一个割 (S, T) 是一个划分 $V = S \cup T$，满足 $S \cap T = \emptyset$ 且 $s \in S$, $t \in T$。

- 一个割的容量 (S, T) 为所有横跨 S 到 T 的边的容量之和，即

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

- 对于一个流 f，流过割 (S, T) 的流量 $f(S, T)$ 表示为：

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

定理 10 **[Max-flow Min-cut theorem].**

给定一个流网络 $G = (V, E, c)$，其最大流等于最小割。

证明. 对于任意流 f 和任意割 (S, T)，我们有：

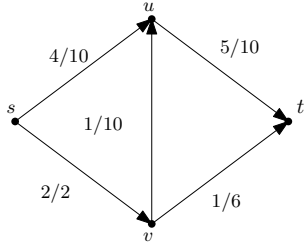
$$f \leqslant f(S, T) \leqslant c(S, T)$$

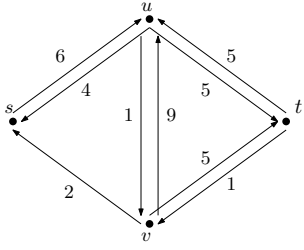
所以任何一个割给与了流的一个上界。

下面我们证明最大流给出了一个相应的割，从而完成定理的证明。为此我们先介绍剩余网络的概念。

剩余图 (I)

给定一个流网络和一个流，能否再增加流量?





我们实际上是在这样一种图上寻找新的流，将已经存在的流 f 图中反向，然后再在这个图上找一条 s 到 t 的路径，这就是可以增加的流量。我们将这种图称为剩余图 (residual graph)。

剩余图 (II)

上海师范大学 Shanghai Normal University

定义 11 **[剩余图 (residual graph)].**

给定一个流网络 $G = (V, E, c)$ 和一个流 f，定义剩余容量函数 $c_f : E \to \mathbb{R}^{\geqslant 0}$ 为：对于 V 中每一对顶点 (u, v) 有 $c_f(u, v) = c(u, v) - f(u, v)$，则关于流 f 的剩余图 $G_f = (V, E_f, c_f)$ 定义为：

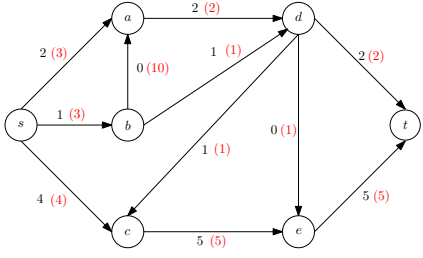
$$E_f = \{(u, v) \in E \mid c_f(u, v) > 0\}$$

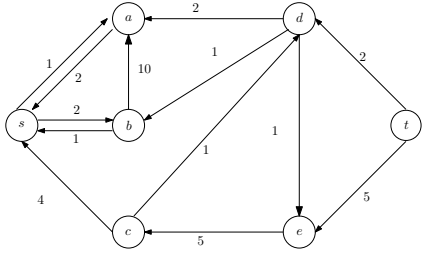
并且我们将在剩余图上一条 s 到 t 的路径 π 称为增广路径 (augmenting path)，π 的瓶颈容量定义为 $\min_{(u,v)\in\pi} c_f(u, v)$，即路径上的最小剩余容量。

直观解释

剩余图的直观理解就是在流 f 的情况下，还有哪些边可以剩余的流量，使其可以继续流动；所以存在的流是反向的边, 而不能是仅仅删除掉。

剩余图举例





可以看到，在剩余图中不再存在 s 到 t 的路径，稍后我们会证明，这个时候的流就是最大流。

引理 12.

设 f 是流网络 G 中的流，f' 是剩余图 G_f 的流，那么 $f+f'$ 是 G 的流并且 $|f+f'|=|f|+|f'|$。

证明.

- 容量限制：对于任意的边 $(u,v)\in E$，

$$f(u,v)+f'(u,v)\leqslant f(u,v)+c_f(u,v)=c(u,v)$$

所以 $f+f'$ 满足容量限制。

- 流守恒：对于任意的节点 $v\in V\setminus\{s,t\}$，

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

所以 $f+f'$ 满足流守恒，并且有 $|f+f'|=|f|+|f'|$。

□

引理 13.

设 f 是流网络 G 中的任意流，f^* 是 G 的最大流，那么 $f^* - f$ 是剩余图 G_f 的最大流。

证明. 反设 G_f 中存在更大的流 f'，那么根据引理12，$f + f'$ 是 G 的流，且 $|f + f'| = |f| + |f'| > |f^*|$，这与 f^* 是 G 的最大流矛盾。 □

剩余图中的增广路径

一条增广路径实际上代表了剩余图中的一个流，因此上述引理其实说明了如下的事实：

- 如果一个流对应的剩余图中不存在增广路径，那么这个流就是最大流。

最大流最小割定理 (续)

有了剩余图的概念，我们可以完成对最大流最小割的全部证明，即一个最大流能给出一个最小割。

最大流最小割定理证明 (续). 假设 f 是一个最大流，由上述引理可得，G_f 中不存在 s 到 t 的增广路径。现在定义如下两个集合:

- $S = \{u \mid u$ 是 s 在 G_f 中可到达的顶点$\}$。
- $T = \{v \mid t$ 是 v 在 G_f 中可到达的顶点$\}$。

显然我们有 $S \cap T = \emptyset$, $S \cup T = V$，从而 (S, T) 是一个分割。注意到 G_f 中不存在 S 到 T 的边，从而有:

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

即 (S, T) 是一个最小割 □

Ford–Fulkerson 算法思路

在上述证明的过程中，自然而然能得到一个求最大流的思路：

1. 从一个初始流 f 开始，不断的在剩余图中寻找增广路径 π。
2. 将 π 的瓶颈容量加至 f 更新新的流 f。
3. 更新剩余图并重复上述过程，直到剩余图中不存在增广路径。

这就是Fold–Fulkson 算法的思路。

Ford–Fulkerson 算法

上海师范大学 Shanghai Normal University

算法 Ford–Fulkerson

输入: 流网络 $G = (V, E, c)$

输出: G 的一个最大流 f

初始化剩余图 $G_f = (V, E_f, c_f)$，其中 $c_f(u, v) = c(u, v)$
初始化流 f，其中 $f(u, v) = 0$
while 存在 G_f 中的 s 到 t 的增广路径 π **do**
    计算 π 的瓶颈容量 b
    for 对于 π 上的每一条边 (u, v) **do**
        if $(u, v) \in E$ **then**
            $f(u, v) \leftarrow f(u, v) + b$
        else
            $f(v, u) \leftarrow f(v, u) - b$
    更新剩余图 G_f

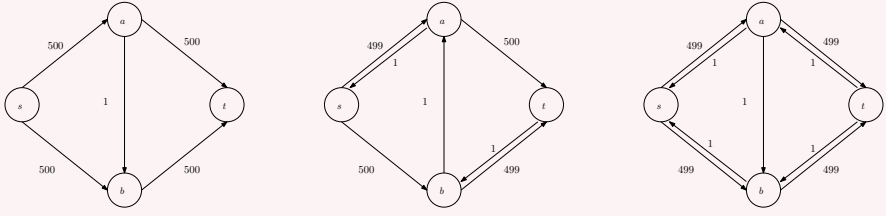
Ford–Fulkerson 分析-容量是整数

当容量是整数时，

- 一条增广路径可以通过 BFS 找出，需要 O(E) 的时间。
- 每次增广的流量至少为 1，所以最多增广 O(|f*|) 次。

因此算法的时间复杂度为 O(E|f*|)。

O(E|f*|) 这一时间复杂性是紧的



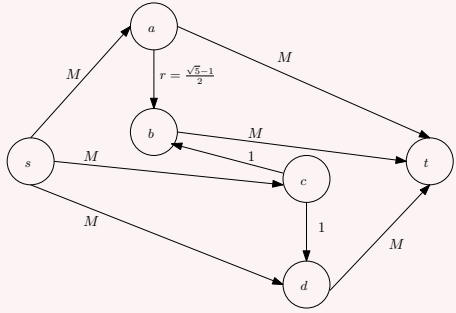
- 交替选择增广路径 sabt 和 sbat 的结果。

Ford–Fulkerson 分析-容量不是整数

当容量不是整数时，Ford–Fulkson 算法可能并不会停止。

容量可能为实数的情况

考察下列例子，其中 M 是任一大于 2 的整数:



- 图中的最大流是 $2M + 1$
- 如果顺着 scbt, sabcdt, scbat, sdcbt 的增广路径进行寻找，其流会收敛于 $2 + 3r$，从而不会终止。

目前讲了什么?

阶段小结

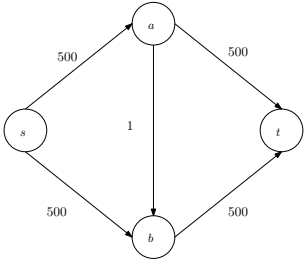
- 流网络的概念、问题引入
  - 什么是流网络？什么是最大流问题
  - 流网络的应用，比如二分匹配。
- Ford – fulkerson 算法
  - 最大流与最小割的关系，最大流最小割定理
  - 剩余图的概念，剩余图中的增广路径
  - Ford–fulkerson 算法
  - Ford–fulkerson 算法分析，时间及其缺陷

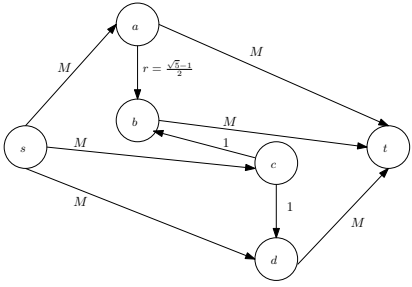
Edmonds–Karp 算法

增广路径的选择

在 Ford–Fulkerson 算法中我们可以看到，如果只是单纯的使用 BFS 在剩余图上寻找增广路径，算法可能会有很差的效果，甚至无法收敛。

一个很自然的想法是如果每次都选择具有最大瓶颈容量的增广路径，那么算法的效率会不会更好呢?





这两个例子中，如果我们每次都寻找最大瓶颈容量的增广路径，则至多只需要 3 次我们就可以找到相应的最大流。

最大容量增值方法 (MCA)(I)

如果我们每次都寻找最大瓶颈容量的增广路径，那么我们就称这样的算法为最大容量增值方法 (MCA)。

引理 14.

一定存在某个序列，使我们从零网络流开始，至多只需要寻找 $|E|$ 次增广路径，就可以找到最大流。

证明. 令 f^* 是一个最大流，G^* 是由所有 G 中满足 $f^*(u,v) > 0$ 的边导出的子图。考虑在 G^* 中一条 s 到 t 的路径 π_i。令 Δ_i 表示其瓶颈容量，则我们构造如下的流 f_{π_i}：

$$f_{\pi_i}(u,v) = \begin{cases} \Delta_i & \text{如果 } (u,v) \in \pi_i \\ 0 & \text{如果 } (u,v) \notin \pi_i \end{cases}$$

则令 $f_i = f^* - f_{\pi_i}$，f_i 也会是图上的一个流。继续考察由 f_i 导出的图，在这个过程中至少有一条边被删除了，从而这个过程至多只会有 $|E|$ 次，因此一定存在某个序列，使得至多只需要寻找 $|E|$ 次增广路径就可以找到最大流。 □

最大容量增值方法 (MCA)(II)

上述引理告诉了我们这样一个序列的存在性，但很遗憾我们并不能根据上述引理直接找到这 |E| 条增广路径：

- 我们并不知道最大流 f 的具体情况。
- 至少存在一条增广路径，其瓶颈容量是 $\frac{|f^*|}{|E|}$.

寻找最大的瓶颈容量的增广路径是有用的！

- 在寻找了 K 次增广路径后，我们可以保证找到的增广路径的瓶颈容量 $\leqslant \frac{|f^*|}{K}$，也就是说我们可以保证在每寻找一定次数之后，增广路径的最大瓶颈容量会按比例缩小。

引理 15.

如果边容量都是整数，MCA 可以在 $O(|E| \cdot \log |f|^*)$ 构造一个最大流，

证明. 假设 MCA 找到 $2|E|$ 条增广路径，则此时剩余增广路径的最大瓶颈容量至多为 $\frac{|f^*|}{2|E|}$，因此至多经历 $2\log|f|^*m$ 次增广路径的寻找，其最大瓶颈容量会小于 1，由于边容量都是整数，所以可以断言此时不存在新的增广路径，从而这是一个最大流。 □

注意到最大瓶颈容量的增广路径可以通过 dijkstra 算法找出，因此

定理 16.

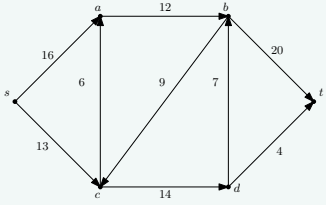
MCA 算法的时间复杂度为 $O(|V|^2 \cdot |E| \cdot \log |f|^*)$。

最短路径增值方法 (MPLA)

MCA 虽然极大的提升了 Fold–Fulkerson 算法的时间复杂性，但其增广路径的寻找次数依旧跟流容量有关，因此我们希望能够进一步提升这一效率。

越来越长的增广路径?

MCA 从瓶颈容量入手，现在我们考虑从路径长度入手。



- sabt 是比 sabcdt 更短的增广路径。

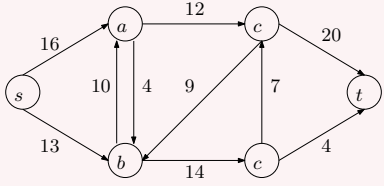
该算法最早由 Edmonds 和 Karp 提出，因此也被称为 Edmonds–Karp 算法。为了从增广路径的长度来寻找增广路径。我们介绍层次图的概念。

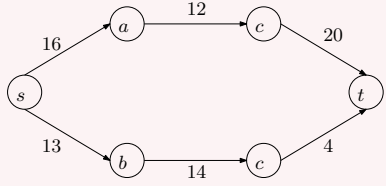
定义 17 **[层次图].**

给定一个流网络 $G = (V, E, c)$，对于其中一个顶点 $u \in V$，u 的层次定义为 s 到 u 的路径的最少边数，记作 $l(u)$。G 的层次图 $G_l = (V, E_l)$ 定义为：

$$E_l = \{(u, v) \in E \mid l(u) + 1 = l(v)\}$$

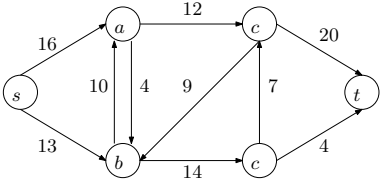
例 18.

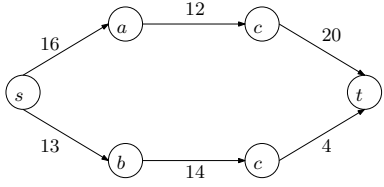


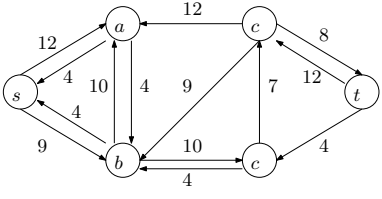


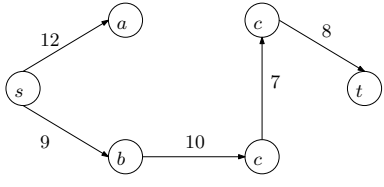
Edmonds–Karp 算法思路 (I)

可以看到，层次图上其实蕴含着当前剩余图下最短的增广路径。









Edmonds–Karp 算法思路 (II)

Edmonds–Karp 算法的思路如下:

1. 根据当前的剩余图构造层次图。
2. 在当前的层次图中寻找 s 到 t 的路径并更新流、剩余图和层次图。
3. 当层次图中不存在 s 到 t 的路径时，根据此时的剩余图计算新的层次图并重复上述过程，直到剩余图中不存在增广路径

为什么最短路径会比最大瓶颈容量路径更好?

- 最短路径被图的顶点所限制。
- 最大瓶颈容量路径被图边上的容量所限制。

Edmonds–Karp 算法

上海师范大学 Shanghai Normal University

算法 Edmonds–Karp

输入: 流网络 $G = (V, E, c)$

输出: G 的一个最大流 f

初始化剩余图 $G_f = (V, E_f, c_f)$，其中 $c_f(u, v) = c(u, v)$
初始化流 f，其中 $f(u, v) = 0$
计算当前的层次图 G_l.
while G_l 中存在 t **do**
    while G_l 中存在 s 到 t 的一条路径 π **do**
        计算 π 的瓶颈容量 Δ
        for 对于 π 上的每一条边 (u, v) **do**
            if $(u, v) \in E$ **then**
                $f(u, v) \leftarrow f(u, v) + \Delta$
        更新剩余图 G_f 和层次图 G_l
    计算新的层次图 G_l

引理 19.
在算法过程中，至多计算 n 个不同的层次图。

证明. 首先证明，每次更新层次图时，增广路径的长度是严格递增的。

- 假设在当前层次图中不存在 s 到 t 的增广路径时，这意味着任何一条 s 到 t 的路径都会使用某条回边，因此在新的层次图上得到的增广路径一定一个大于当前得到的增广路径的长度。

然后注意到增广路径的长度至多为 $n-1$，从而层次图至多有 n 个，这其中包含了一个 t 不在层次图里的图 (循环终止条件)。 □

定理 20.

Edmonds–Karp 算法的时间复杂度为 $O(|V| \cdot |E|^2)$。

证明. 我们来依次计算算法的复杂性：

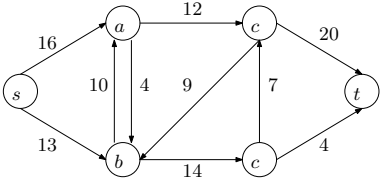
- 层次图的计算需要 $O(|E|)$ 的时间，因为只需要进行一次 BFS。
- 在每个层次图中，寻找一条 s 到 t 的路径，也需要 $O(|E|)$ 的时间。
- 在每个层次图中，至多会找到 $|E|$ 条增广路径，这是因为每找到一条路径 π，尽管可能会增加 π 条新的路径，但这些都是回边，因此不会产生新的最短增广路径；而同时必然有一条边被删除，从而至多会找到 $|E|$ 条增广路径。

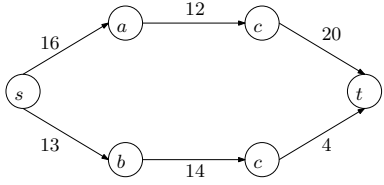
由上述引理，至多会有 n 张层次图，因此整个算法的时间复杂性是 $O(|V| \cdot |E|^2)$。 □

Dinic 算法

更快的算法?

回顾在层次图上的增广路径寻找:





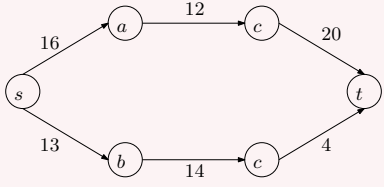
我们可不可能在层次图中同时找到 $sact$ 和 $sbct$ 两条增广路径?

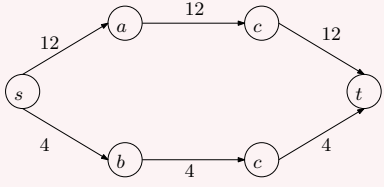
为了完成此事，我们引入阻塞流的概念。

定义 21 **[阻塞流].**

给定一个流网络 $G = (V, E, c)$，H 是一个包含 s, t 的 G 的子图，则 H 中的一个流被称为 H 的一个阻塞流，如果对于任意的 H 中的 s 到 t 的路径 π，H 中存在一条边 $(u, v) \in \pi$，使得 $f(u, v) = c(u, v)$。

例 22.





求阻塞流的思想-深度优先搜索

运用 DFS 的思想，我们可以在层次图上一口气找到所有的增广路径。

1. 从 s 出发进行深度搜索。
2. 如果访问到了一个不是 t 并且没有出度的点，那么算法退回并且删去跟这个点和跟这个点有关的边。
3. 如果访问到了 t，则根据这个路径构建一个具有最大瓶颈容量的流 f，删除相应的饱和边，然后回到路径上最远能到达的点重新进行搜索。

这样通过一次 DFS，我们就可以找到所有层次图上的增广路径，即构造了该层次图上的一个阻塞流。这个算法被称为Dinic 算法。

算法 Dinic

输入: 流网络 $G = (V, E, c)$

输出: G 的一个最大流 f

初始化剩余图 $G_f = (V, E_f, c_f)$，其中 $c_f(u, v) = c(u, v)$
初始化流 f，其中 $f(u, v) = 0$ 并计算当前的层次图 G_l.
while G_l 中存在 t **do**
    $u \leftarrow s$, $p \leftarrow u$
    while s 的出度 > 0 **do**
        while $u \neq t$ and s 的出度 > 0 **do**
            if u 的出度 > 0 **then**
                $p \leftarrow p, v, u \leftarrow v$，这里 (u, v) 是 G_l 中的一条边
            else
                删除 u 和其在 G_l 中所有的邻接边
                p 末尾删除 u，$u \leftarrow p.end()$
        if $u = t$ **then**
            根据 p 的最大瓶颈容量增值流 f，删除 G_l 中相应的饱和边
            将 u 设置为当前路径目前可以达到的最远的点
    计算新的层次图 G_l

Dinic 算法分析

与 Edmonds–Karp 算法相同，至多只会存在 n 个层次图，因此我们只需要考虑一次在层次图中的运行时间。

- 每次增值至少删除一条边，因此至多会有 $|E|$ 次增值，每条路径长度至多为 $|V|-1$，因此增值操作的花费为 $O(|V|\cdot|E|)$。
- 在 DFS 搜索中，最多有 $|V|-1$ 次后退，因为一次后退至少会删除一个点，注意到 DFS 向前搜索的步数要么成为了某条增广路径的一部分，要么抵消了某次后退，因此这部分的花费也是 $O(|V|\cdot|E|)$。

由上述讨论我们可以知道:

定理 23.

Dinic 算法的时间复杂度为 $O(|V|^2\cdot|E|)$。

进一步的改进-从边到点

阻塞流还是从边的角度来考虑最大瓶颈容量的。我们能不能从点的角度来考虑呢?

定义 24 **[最小通过量].**

给定一个流网络 $G = (V, E, c)$，$s, t \in V$，$u \in V \setminus \{s, t\}$，定义 u 的最小通过量为:

$$\delta(u) = \min\{\sum_{u \in V} c(u, v), \sum_{u \in V} c(u, v)\}$$

s 和 t 的最小通过量定义为:

$$\delta(s) = \sum_{v \in V - \{s\}} c(s, v),\ \delta(t) = \sum_{v \in V - \{t\}} c(v, t)$$

通过最小通过量来获取阻塞流

一个点 v 的最小通过量实际代表了这个点的某种容量限制，因此我们可以据此构造一个最大的通过这个带你的流。

- 根据 $\delta(v)$ 的值我们向 v 到 t 的点“推送“流，即使得 v 到 t 一路上边充满流量为 $\delta(v)$ 的流。
- 根据 $\delta(v)$ 的值我们向 s 到 v 的点“拉入“流，即使得 s 到 v 一路上边充满流量为 $\delta(v)$ 的流最终使其流入 v。

这样子的改进方法被称为MPM 算法，由 Malhotra, Pramodh–Kumar, Maheshwari 三人共同发现。

MPM 算法

上海师范大学 Shanghai Normal University

算法 MPM

输入: 流网络 $G = (V, E, c)$

输出: G 的一个最大流 f

初始化剩余图 $G_f = (V, E_f, c_f)$，其中 $c_f(u, v) = c(u, v)$
初始化流 f，其中 $f(u, v) = 0$ 并计算当前的层次图 G_l.
while G_l 中存在 t **do**
    while G_l 中存在 s 到 t 的一条路径 π **do**
        查找最小通过量为 g 的顶点 v
        从 v 到 t 推下流量为 g 的流
        从 s 到 v 拉入流量为 g 的流
        更新流 f，剩余图 G_f 和层次图 G_l
    计算新的层次图 G_l

时间复杂性：$O(|V|^3)!$

从线性规划的角度理解最大流最小割定理

最大流问题的线性规划表示

我们实际上可以把最大流问题转换成如下的线性规划问题：

$$\max \quad \sum_{e=(s,\_)\in E} f_e$$

$$\text{s.t.} \quad \sum_{e=(\_,v)\in E} f_e - \sum_{e=(v,\_)\in E} f_e = 0 \quad \forall v \in V \setminus \{s,t\} \tag{1}$$

$$f_e \leqslant c(u,v) \qquad \forall\, e=(u,v) \in E \tag{2}$$

$$f_e \geqslant 0 \qquad \forall\, e=(u,v) \in E \tag{3}$$

- 我们给每一条边 e 都赋值一个未知数 f_e
- 约束式2和20表示了流的容量限制。
- 约束式1表示了流的守恒性。

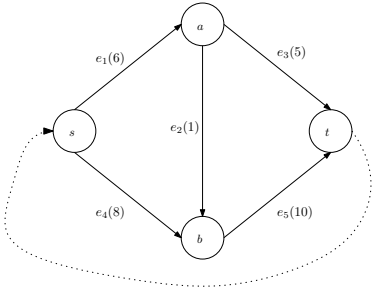
最大流问题的另一种线性规划表示

我们可以稍微修修改一下表示方法，，假设存在一条 t 回到 s 的边 $e_0 = (t, s)$，令其容量没有限制，则我们可以把流守恒性的约束同样作用在 s 和 t 上，从而整个线性规划问题转换为：

$$
\begin{aligned}
\max \quad & f_{e_0} \\
\text{s.t.} \quad & \sum_{e=(\_,v)\in E} f_e - \sum_{e=(v,\_)\in E} f_e = 0 \quad \forall v \in V && (4)\\
& f_e \leqslant c(u,v) \qquad \forall\, e=(u,v)\in E && (5)\\
& f_e \geqslant 0 \qquad \forall\, e=(u,v)\in E && (6)
\end{aligned}
$$

一个例子

对于下面的流网络:



对应的线性规划问题为:

$$\begin{aligned} \max \quad & \mathrm{f_{ts}} \\ \text{s.t.} \quad & \mathrm{f}_1 = \mathrm{f}_2 + \mathrm{f}_3,\ \mathrm{f}_2 + \mathrm{f}_4 = \mathrm{f}_5,\ \mathrm{f_{ts}} = \mathrm{f}_1 + \mathrm{f}_4,\ \mathrm{f}_3 + \mathrm{f}_5 = \mathrm{f_{ts}} && (7) \\ & \mathrm{f}_1 \leqslant 6,\ \mathrm{f}_2 \leqslant 1,\ \mathrm{f}_3 \leqslant 5,\ \mathrm{f}_4 \leqslant 8,\ \mathrm{f}_5 \leqslant 10 && (8) \\ & \mathrm{f}_1,\ \mathrm{f}_2,\ \mathrm{f}_3,\ \mathrm{f}_4,\ \mathrm{f}_5 \geqslant 0 && (9) \end{aligned}$$

理解线性规划

我们可以看到，约束式的线性组合其实给了目标函数的一个上界：

- $\mathrm{f}_3 \leqslant 5$ 和 $\mathrm{f}_5 \leqslant 10$ 组合可以得到 $\mathrm{f_{ts}} = \mathrm{f}_3 + \mathrm{f}_5 \leqslant 5 + 10 = 15$
- $\mathrm{f}_1 \leqslant 6$ 和 $\mathrm{f}_4 \leqslant 8$ 组合可以得到 $\mathrm{f_{ts}} = \mathrm{f}_1 + \mathrm{f}_4 \leqslant 6 + 8 = 14$

上述上界并不能一定取到，因为其不一定满足所有的约束。

将这些约束全部组合起来!

理解线性规划 (II)

$$
\begin{aligned}
\max \quad & f_{ts} \\
\text{s.t.} \quad & f_2 + f_3 - f_1 = 0,\ f_5 - f_4 - f_2 = 0,\ f_1 + f_4 - f_{ts} = 0,\ f_{ts} - f_3 - f_5 = 0 && (10)\\
& f_1 \leqslant 6,\ f_2 \leqslant 1,\ f_3 \leqslant 5,\ f_4 \leqslant 8,\ f_5 \leqslant 10 && (11)\\
& f_1,\ f_2,\ f_3,\ f_4,\ f_5 \geqslant 0 && (12)
\end{aligned}
$$

第二行的每个等式都是跟某个顶点 u 相关的，我们记相应的等式左边为 M_u 并赋给每个等式一个变量 x_u。

第三行的每个不等式都是跟某条边 e 相关的，我们赋给每个不等式一个变量 y_e。由边的记号这里可以用 y_i 来表示。

理解线性规划 (III)

其线性组合可以表示为：

$$\sum_{u \in U} x_u \cdot M_u + \sum_{i=1}^{5} y_i \cdot c_i \leqslant \sum_{i=1}^{5} y_i \cdot c_i$$

当我们有：

$$\sum_{u \in U} x_u \cdot M_u + \sum_{i=1}^{5} y_i \cdot c_i \geqslant f_{ts} \text{ 恒成立}$$

上述不等式的右边 ($\sum_{i=1}^{5} y_i \cdot c_i$) 提供了一个 f_{ts} 的上界.

- 我们可以考虑将这个值最小化的问题，它也是个线性规划

对偶问题

我们将相应的问题表示出来为：

$$\min \quad \sum_{i=1}^{5} y_i \cdot c_i$$

$$\text{s.t.} \quad x_a - x_s + y_1 \geqslant 0,\ x_b - x_a + y_2 \geqslant 0,\ x_t - x_a + y_3 \geqslant 0 \tag{13}$$

$$x_b - x_s + y_4 \geqslant 0,\ x_t - x_b + y_5 \geqslant 0 \tag{14}$$

$$x_s - x_t \geqslant 1 \tag{15}$$

$$x_u, y_i \geqslant 0 \qquad u \in V,\ i \in \{1.2.3.4.5\} \tag{16}$$

这恰巧是求上述流网络最小割的线性规划问题！

- 直观上理解，可以认为 x_u 是一个用来划分顶点集合的变量，当 $x_u \neq x_v$ 不相等时 (u, v) 就是一条跨边，也就是割的一部分，从而相应的 y_i 必须得是 1。

我们称这样的问题为原问题的对偶问题。

最大流与最小割问题联系-对偶问题

一般情况下，给定一个流网络最大流的线性规划问题：

$$
\begin{aligned}
\max \quad & f_{e_0} \\
\text{s.t.} \quad & \sum_{e=(\_,v)\in E} f_e - \sum_{e=(v,\_)\in E} f_e = 0 \quad \forall v \in V && (17)\\
& f_e \leqslant c(u,v) \qquad \forall\, e=(u,v)\in E && (18)\\
& f_e \geqslant 0 \qquad \forall\, e=(u,v)\in E && (19)
\end{aligned}
$$

其相应的对偶问题为：

$$
\begin{aligned}
\min \quad & \sum_{e} c_e \cdot y_e \\
\text{s.t.} \quad & x_v - x_u + y_e \geqslant 0 \quad \forall e=(u,v)\in E && (20)\\
& x_s - x_t \geqslant 1 && (21)\\
& y_e, x_u \geqslant 0 \qquad \forall\, e\in E,\ u\in V && (22)
\end{aligned}
$$

最大流与最小割问题联系-对偶问题 (II)

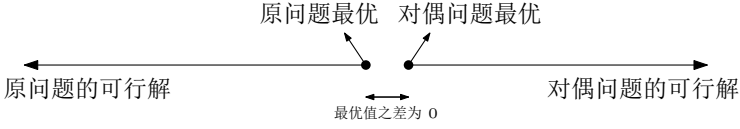
最大流与最小割定理告诉我们，上述两个问题的解是相同的，即：

$$\max\ f_{e_0} = \min \sum_{e} c_e \cdot y_e$$

这不是一个个例，事实上：

定理 25 **[对偶定理].**

如果线性规划问题的目标函数有界，则对偶问题的目标函数也有界，并且两者相等。



本章总结

本章具体内容

- 流网络问题介绍
  - 流网络的定义
  - 流网络的应用
- Ford–fulkerson 算法
  - 最大流最小割定理
  - 剩余图与增广路径
- 改进的算法
  - MCA 算法
  - Edmonds–Karp 算法
  - Dinic 算法
  - MPM 算法
- 从线性规划的角度理解这个问题
  - 最大流问题的线性规划表示
  - 对偶定理

一些补充

关于最大流问题

实际上，最大流问题还有很多高效的算法，比如：

- 基于预流的推送-重贴标签算法和前置重置标签算法。
- 在 STOC2013 中，Orlin 提出了一个时间复杂度为 $O(|V| \cdot |E|)$ 的算法。
- 在一些稀疏的流网络中，也还存在更快的算法。

关于线性规划

- 线性规划本身是建模问题的一个很重要的方法。
- 关于线性规划，比较著名的算法有单纯形法和内点法。
- 对偶定理可以起到一个很重要的作用，有些时候的对偶问题会比原问题要更加简单，同时我们也可以利用对偶问题来设计一些近似算法。