网络流算法详解

网络流算法在许多实际问题中有应用，如匹配问题，著名的 Hall 婚姻定理。这里不证明“最大流最小割定理”，简单解释求最大流的 Ford-Fulkerson 算法。接下来分别详述时间复杂度为 $O(VE^2)$的 Edmonds-Karp 算法和时间复杂度为 $O(V^2E)$的 Dinic 算法。至于较新的预留推进算法就不介绍了，这个算法证明比较难，感兴趣的可以看看算法导论。

本文所用到的网络流如图 1，s 为原点，t 为汇点，边上的值表示边的容量 $c(u,v)$，如 $c(s,v_1)=15$，$c(v_1,v_2)=8$。流用符号 $f(u,v)$表示，如图 2，流的容量 $f(s,v_1)=10$，$f(v_1,v_2)=5$。剩余容量 $c_f(u,v)=c(u,v)-f(u,v)$。在原剩余网络中找到一条流后，修改原网络边的剩余容量得到剩余网络 G_f，如图 3 所示。注意剩余网络中有一些新添加的边即反向边的容量，为流的反馈。在图 3 中，有 $f(v_1,v_3)=5$,那么有 $f(v_3,v_1)=-f(v_1,v_3)=-5$，然后 $c_f(v_3,v_1)=c(v_3,v_1)-f(v_3,v_1)=0-(-5)=5$。

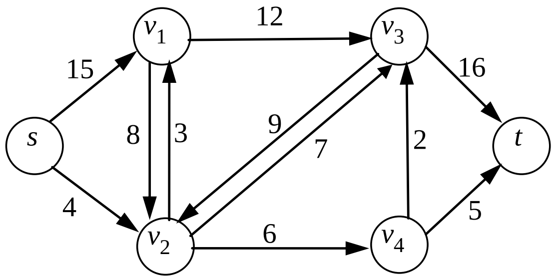


图 1 网络流的一个例子

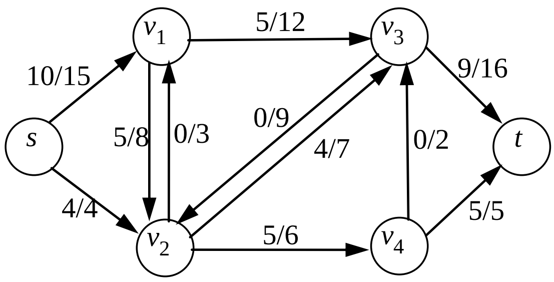


图 2 图 1 的一条流 f

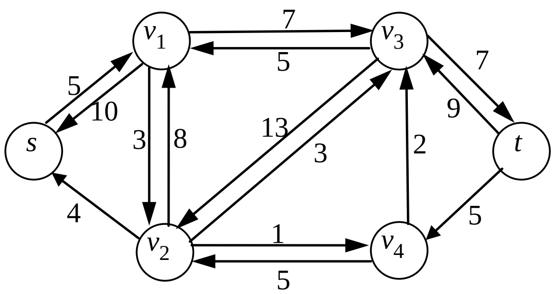


图 3 有图 2 产生的剩余网络 G_f

1. Ford-Fulkerson 算法

算法的主要思想：

1）初始化一条容量为 0 的流 f 和一个剩余网络 G_f，第一个剩余网络为原图 G，每条边的剩余容量初始化为每条边的初始容量 $c_f(u,v)=c(u,v)$。

2）在剩余网络 G_f 中寻找增广路径 p，取增广路径 p 中的边的剩余容量 $c_f(u,v)$ 最小值作为流的增量 Δf，使得 $f'=f+\Delta f$。修改剩余图中每条边的容量 $c_{f'}(u,v)= c_f(u,v)-\Delta f$ 得到剩余网络 G_f。（补充说明：原算法中的 Δf 可以为单位 1）

3）重复步骤 2），直到找不到一条增广路径为止。

寻找一条增广路径的方法如图 4 所示，然后确定增广路径上的流增量 Δf。本例中 $\Delta f=3$。

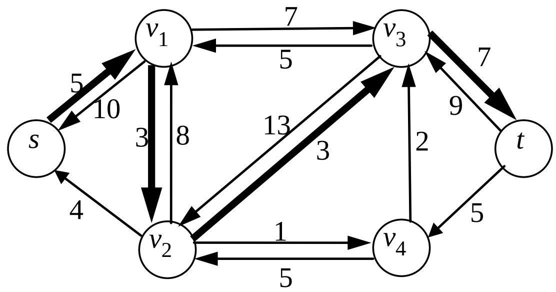


图 4 在图 3 的剩余图中寻找一条增广路径 P

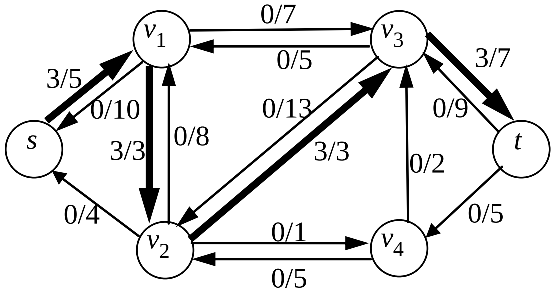
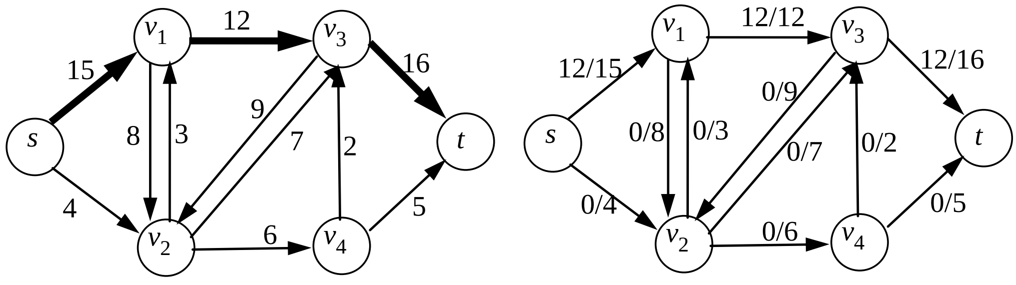


图 5 确定图 4 中增广路径 P 的流增量 Δf

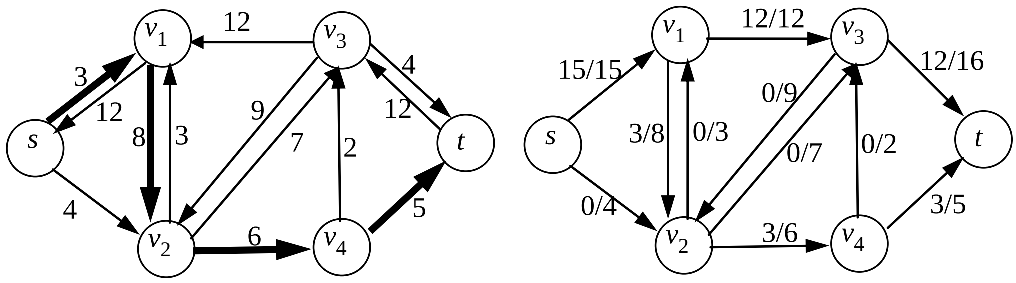
整个网络可以用邻接表或矩阵表示，网络的边即为矩阵中的元素，网络的节点则为矩阵一维的下标。这里网络中每条边最好用一个结构体表示。结构体包含边的容量和每次通过的流量这两个变量。

下面是一个例子：

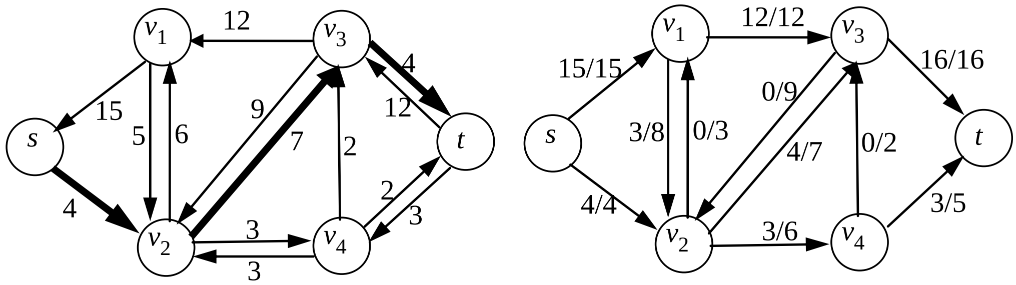
(a)



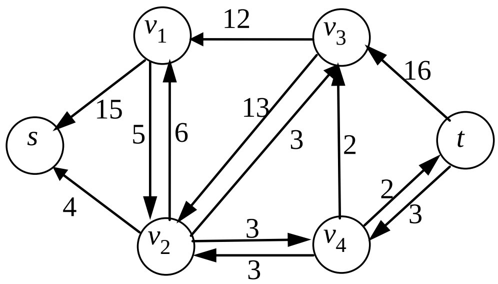
(b)



(c)



(d)



Ford-Fulkerson 算法的伪代码如下：

```
Ford-Fulkerson(G, s, t)
1    for each edge (u, v) ∈ E[G]          //初始化每条边的流量为 0
2     {    f[u, v] ← 0
3          f[v, u] ← 0
4     }
5    //Gf ← G                               //初始化剩余网络 Gf 为原网络 G，这里不需要代码
6    while there exists a path p from s to t in the network Gf  //网络中还存在增广路径，仍然进行迭代
7     {search a path p from network Gf            //Karp 算法采用广度优先，Dinic 算法采用深度优先
8      cf(p) ← Min{ cf(u, v) | (u, v) is in p}      //确定增广路径上的流增量Δ f(p)= cf(p)
9         for each edge (u, v) in p
10        {    f[u, v] ← f[u, v] + cf(p)            //增加剩余网络中增广路径上每条边的流量
11             f[v, u] ← - f[u, v]                  //显然该路径上反方向上的容量为负
12             cf[u, v] ← c[u, v] – f[u, v]          //计算剩余网络 Gf 中的每条边的容量
13             cf[v, u] ← c[v, u] – f[v, u]
14         }
15    }
```

2. Edmonds-Karp 算法

Edmonds-Karp 算法与 Ford-Fulkerson 算法的区别在于在 Ford-Fulkerson 算法的第 7 行，Edmonds-Karp 算法采用广度优先算法（BFS）寻找一条从 *s* 到 *t* 最短增广路径 *p* 代替 Ford-Fulkerson 的随机寻找一条从 *s* 到 *t* 增广路径 *p*。

引理 1：在网络 G=<V,E>中，原点为 s，汇点为 t。Edmonds-Karp 算法中，对于任意顶点 $v \in V - \{s, t\}$，在剩余网络 G_f 中的距离 $\delta_f(s, v)$ 和 $\delta_f(v, t)$ 随着流的增加而单调递增。（每次增加两个单位，这里要用到 BFS 生成最短路径的性质，由于这次的增广路径在剩余网络中已经是最短路径了，在新的剩余网络中，通过 (s, v) 的最短路径要增加。证明略）

引理 2：流增加的总次数不超过 O(VE)。

证明（1）：若在剩余图 G_f 中的边 (u, v) 满足 $c(u,v)=c_f(u, v)$，边 (u, v) 是一条关键边。每次进行增广路径扩充后，关键边 (u, v) 不会在该次的剩余网络 G_f 中出现。每次扩充至少会有一条关键边。可以证明网络中的每条边称为关键边至多 $|V|/2-1$ 次。所以流增加的总次数为 O(VE)。

证明（2）：当边 (u, v) 在上一次剩余网络 G_f 中第一次称为关键边时，有 $\delta_f(s, v)= \delta_f(s, u) + 1$ 成立。然后边 (u, v) 将不会在该次剩余网络 $G_{f'}$ 中，边 (u, v) 下一次出现在某个剩余网络中的时候有，肯定有流通过边 (v, u)。假设当这种情况发生时，有网络 $G_{f''}$ 的流为 f'，我们有：

其中由引理 1：$\delta_{f'}(s, v)\geq\delta_f(s, v)$，有

$\delta_{f'}(s, u)= \delta_{f'}(s, v)+1\geq\delta_f(s, v)+1=\delta_f(s, u)+2$

所以从 s 到 u 的路径中，当其中的一条边称两次为关键边的时候，s 到 u 的距离增加 2 个单位。s 到 v 的距离最长为 n-1，那么 s 到 u 的最长距离为 n-2。所以边 (u, v) 能成为关键边的次数最多为 n/2-1 次。所以流增加的总次数为 O(VE)。

Edmonds-Karp 算法的时间复杂度：$O(VE^2)$

由引理 2 可知，流增加的总次数不超过 O(VE)，又每次扩充增广路径的时间复杂度为 O(E)，故 Edmonds-Karp 算法的时间复杂度为 $O(VE^2)$。

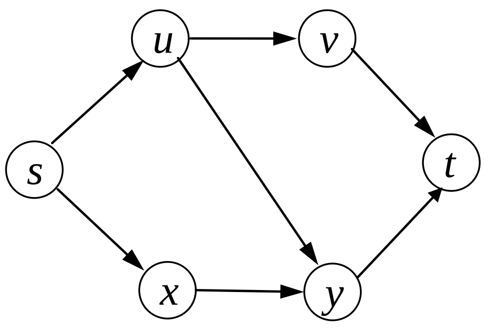
3. Dinic 算法

Dinic 算法要用到层次图的数据结构，即 MSN (Multi-Stage Network)。MSN 可由每次计算得到的剩余网络 G_f 再计算得到。

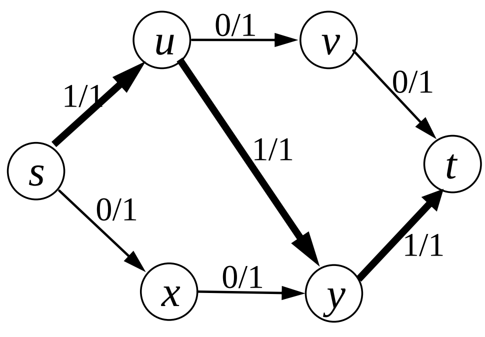
定义 1：k-阶图，k-stage 图 (或网络) G = (V, E)是一个有向图，G 的顶点集合被分成 $(k+1)\geq 2$ 个不相交的集合 V_i，$0 \leq i \leq k$。如果有边(u, v)∈E，则有 $u \in V_i$ 和 $v \in V_{i+1}$，*对某个* $i\in[0,k-1]$ 成立。$V_0=s$，$V_k=t$。k-阶图和 k-阶图中的一条阻塞流如下图所示。图中 6 产生了一条阻塞流表示在该图上从原点 s 到汇点 t 再不可能增加新的流了。

定义 2：饱和边，如果边 (u, v) 满足 $f(u, v)=c(u, v)$。

定义 3：流 f 称为网络 G 的阻塞流，当且仅当从 s 到 t 的任何一条路径中都包含一条饱和边。



(a) 一个 3-阶图



(b) 一条阻塞流的例子

图 6 k-阶图

Dinic 算法中用到的 MSN 数据结构(多阶网络)如下图 8 所示。MSN 可以在剩余网络 G_f 中用 BFS 算法构建，时间复杂度为 O(|V|+|E|)。要得到 Dinic 算法使用的 MSN 还需要一些简答的处理，如图 7 中，利用 BFS 得到的 MSN 可能含有边不能到汇点 t，所以在计算 MSN 的过程中需要将这种边除去。如图 8 中的边(v_3, x)。

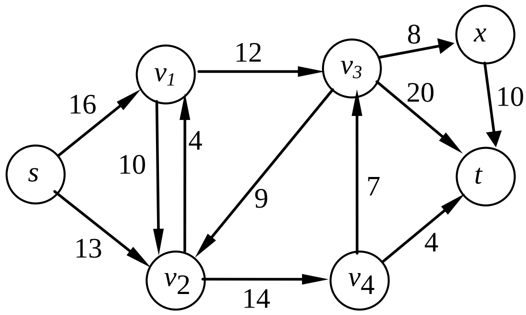


图 7 一个剩余网络 G_f

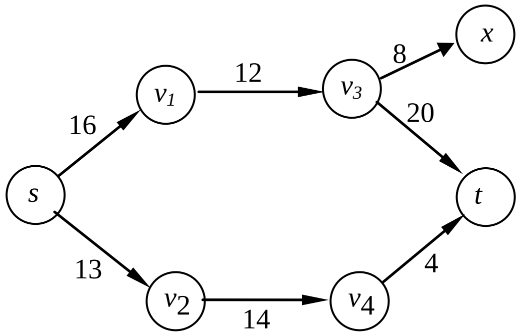


图 8 剩余网络图 7G_f的一个 MSN

引理 1：Dinic 算法中每次迭代后，MSN 的阶数至少增加 2。

假设流 f^* 是一个由 G_f 计算得到的 MSN_f的阻塞流，$f'=f+f^*$，令 $MSN_{f'}$ 和 MSN_f 为 $G_{f'}$ 和 G_f 计算得到的 MSN。$\delta_{f'}(s, t)$和$\delta_f(s, t)$分别是在 $MSN_{f'}$ 和 MSN_f 上从 s 到 t 的最短路径。我们仅仅需要证明$\delta_{f'}(s, t)= \delta_f(s, t)+2$，除非 t 不在 MSN 中。考虑任何一条从 s 到 t 的路径 p，明显有$|p|=\delta_{f'}(s, t)$。如果我们假定路径 p 上所有的边都在 $MSN_{f'}$ 上，那么流 f^*将不是一条阻塞流，因为我们可以继续增大在 MSN_f上的路径 p 的流 f 的容量，使得 MSN_f上的路径 p 的边至少有一条边成为饱和边，那么该边将不会在 $MSN_{f'}$ 上出现。所以这里必有一条边(u, v)将不会在 $MSN_{f'}$ 上出现。令边(u, v)是在 MSN_f上出现而不会在 $MSN_{f'}$ 上出现的边。当边(u, v)再次出现的情况是因为边(v, u)成为增广路径 p' 上的一条饱和边。这意味着有$\delta_f(s, v)= \delta_f(s, u)+1$；$\delta_{f'}(s, u)= \delta_{f'}(s, v)+1$。因为边($u$, v)在路径 p 上，由第 2 节中的引理 1 可知，$\delta_f(s, u) \le \delta_{f'}(s, u)$和$\delta_f(s, v) \le \delta_{f'}(s, v)$成立。所以我们得到

$$
\begin{aligned}
\delta_f(s, t) &= \delta_f(s, v)+ \delta_f(v, t) \\
&= \delta_f(s, u) +1+\delta_f(v, t) \\
&=(\delta_f(s, v)-1)+1+(\delta_f(u, t)-1) \\
&=\delta_f(s, v)+1+\delta_f(u, t)-2 \\
&\le\delta_{f'}(s, v)+1+\delta_{f'}(u, t)-2 \\
&=\delta_{f'}(s, v)+\delta_{f'}(u, t)-1 \\
&=\delta_{f'}(s, v)+(\delta_{f'}(v, t)-1)-1
\end{aligned}
$$

$=\delta_{f'}(s, t)-2$

引理 2：Dinic 算法迭代的轮数至多为$\lfloor n/2 \rfloor$，即计算 MSN 的次数为 O(V)。

第一个 MSN 的计算式从流 f 初始化为 0 开始的（见后面算法伪代码），用 Dinic 算法又计算了 MSN k 次。由引理 1 可知，1+2k≦n-1，因为任何路径的长度，包括$\delta_f(s, t)$都小于或等于 n-1，所以 k ≤$\lfloor n/2 \rfloor$- 1，引理得证。

引理 3：Dinic 算法每次计算阻塞流的时间复杂度为 O(VE)。

阻塞流的寻找可以采用 DFS 算法。在 MSN 中从源点 s 出发寻找一条阻塞流 f^*，初始化 $f^*=0$。我们利用栈 Stack 来构建 DFS 算法寻找一条阻塞流，压栈到最深的一个节点后，然后当一个节点 v 从栈 Stack 中 Pop 出来后，分两种情况讨论：

1）$v=t$

因为存储在栈 Stack 中的顶点序列从栈底到栈顶为一条从 s 到 t 的路径，也是 MSN 中从 s 到 t 的一条最短路径 p。令 $c_f(p)$为路径 p 的容量。在剩余网路 G_f中增加路径 p 上的流 f^*的容量至 $c_f(p)$，然后计算新的 $\mathrm{MSN}_{f'}$。对于在 MSN_f中的路径 p 的每条边(u, v)，减去容量 $c_f(p)$。如果该边的容量减少至 0 后，就将该边标记成饱和边。在 $\mathrm{MSN}_{f'}$ 中这条边就不存在了，相反在 $\mathrm{G}_{f'}$ 中会有一条反向的边出现。

2）$v\neq t$

这意味着顶点 v 不能到达汇点 t，也即是栈中的路径从 s 只能最终到达 v 不能到达 t，需要退栈。从新选择新的边压栈。

每次 DFS 的时间为 O(n)，因为在 MSN_f中路径的长度最多为 n-1，每次压栈到顶点 t 后，开始出栈，修改边的容量，若该边为饱和边则在 MSN_f 中去掉该边。然后，继续退栈，寻找新的边入栈新的顶点，如此反复进行 DFS 收索直到一条阻塞流 f^*产生。实际情况中阻塞流 f^*产生后，MSN_f中已经没有从源点 s 到达汇点 t 的边了。所以在 MSN_f中利用 DFS 寻找一条阻塞流 f^*的时间复杂度为 O(VE)。

引理 4：Dinic 算法的时间复杂度为 $O(V^2E)$。

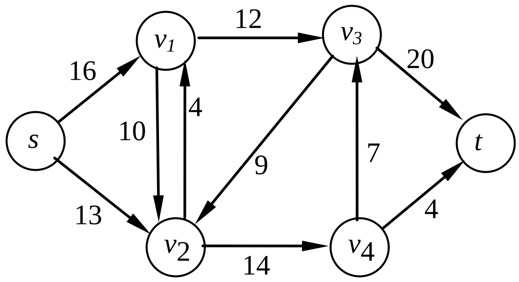
由引理 2 和引理 3 可知 Dinic 算法的时间复杂度为 $O(V^2E)$。

Dinic 算法伪代码

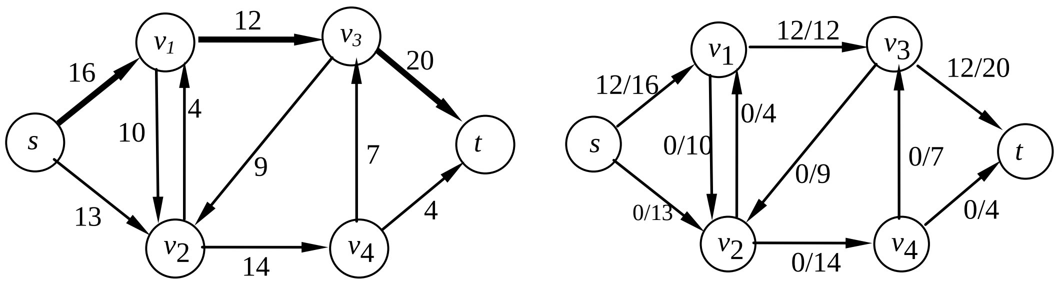
```
Dinic (G, s, t)
    for each edge (u, v) ∈ E[G]
    {       f[u, v] ← 0
            f[v, u] ← 0
    }
    Gf ← G
    Compute the MSN for Gf starting from source s
    while sink t is in MSN
     {          find a blocking flow f* in MSN
                for each edge (u, v) in G
                {     f[u, v] ← f[u, v] + f*[u, v]     }
                Compute Gf for flow f
                Re-compute MSN for Gf
     }
    End
```

Dinic 算法的例子

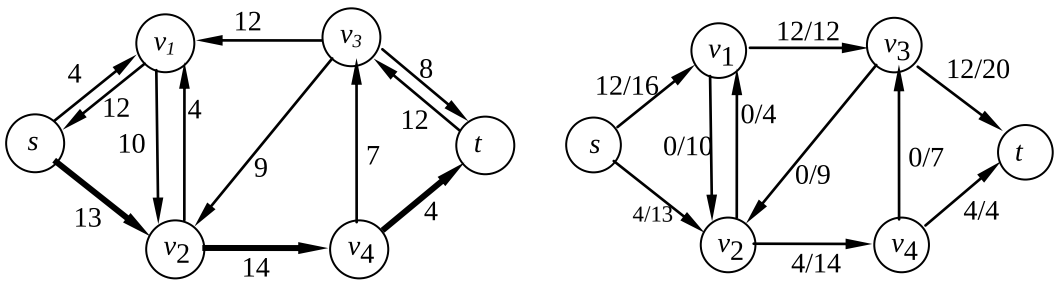
下面是 Edmonds-Karp 算法的一个例子


s
v_1
v_2
v_3
v_4
t
16
13
10
4
12
9
14
7
20
4

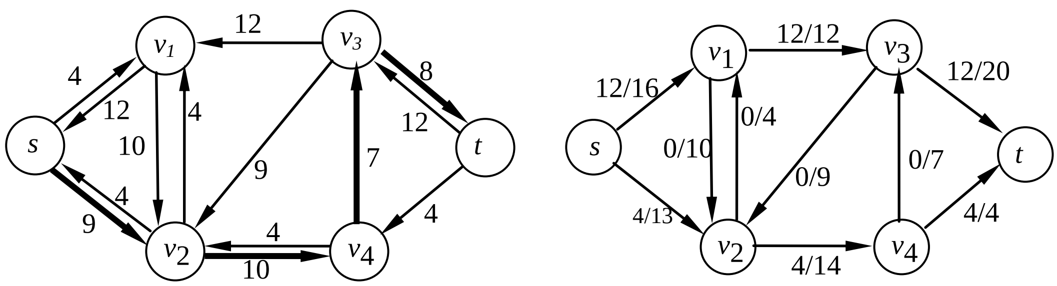
原网络


s
v_1
v_2
v_3
v_4
t
16
13
10
4
12
9
14
7
20
4
12/16
0/13
0/10
0/4
12/12
0/9
0/14
0/7
12/20
0/4

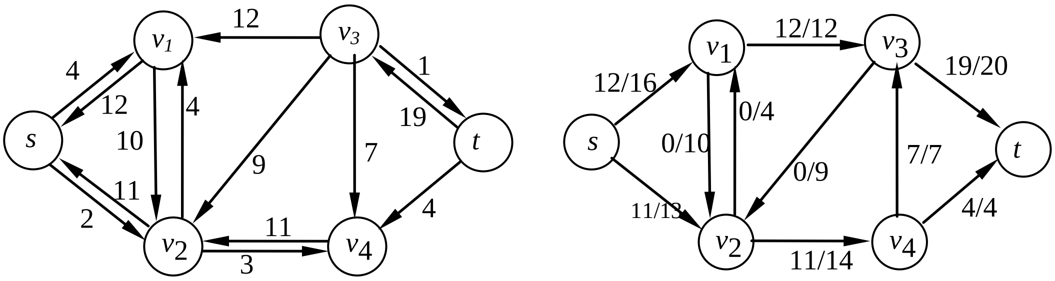
(a)


s
v_1
v_2
v_3
v_4
t
4
12
13
10
4
12
9
14
7
8
12
4
12/16
4/13
0/10
0/4
12/12
0/9
4/14
0/7
12/20
4/4

(b)

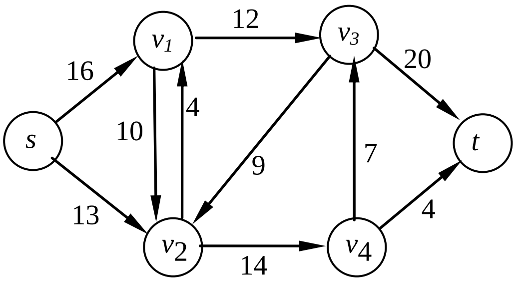

s
v_1
v_2
v_3
v_4
t
4
12
4
9
10
4
12
9
4
10
7
8
12
4
12/16
4/13
0/10
0/4
12/12
0/9
4/14
0/7
12/20
4/4

(c)

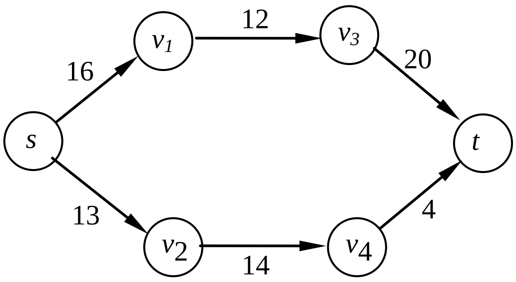


(d)

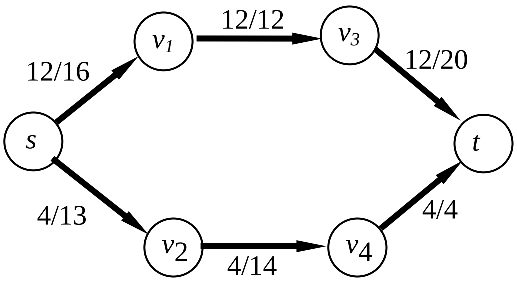
最后举一个 Dinic 算法的例子



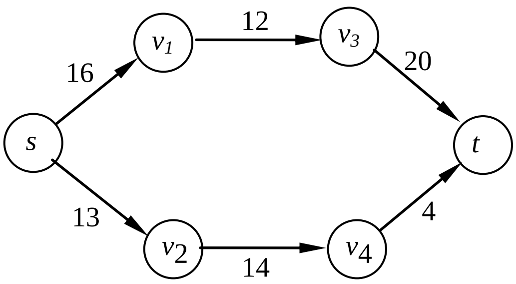
原网络 G，也是第一个剩余网络 G_f



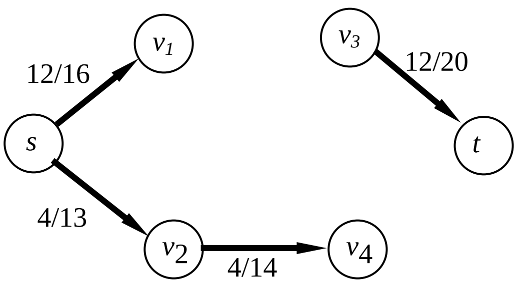
第一个 MSN



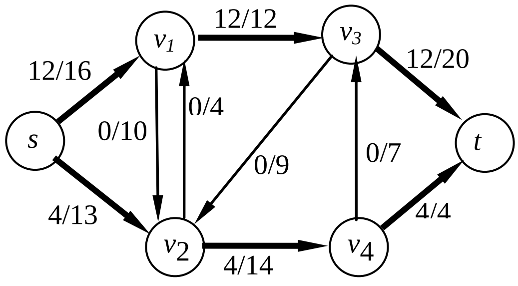
MSN 上的阻塞流 f^*=16



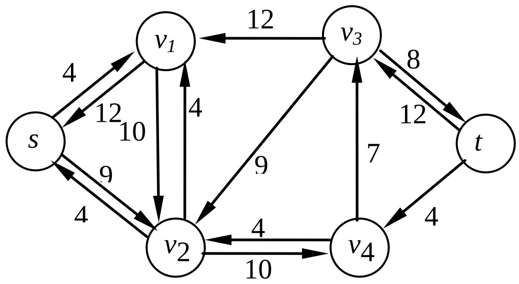
第一个 MSN



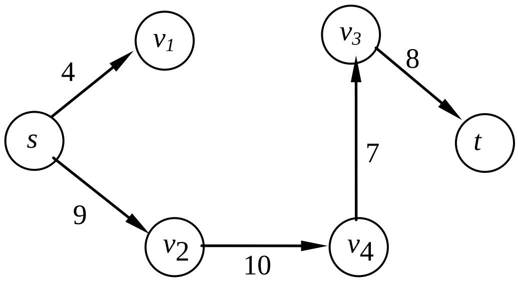
一轮迭代后最终的 MSN f^*=16



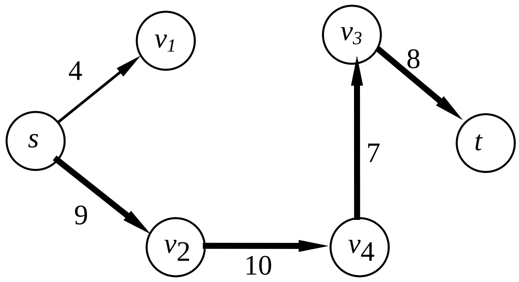
得到阻塞流 f^*后最终的流 $f'=16$



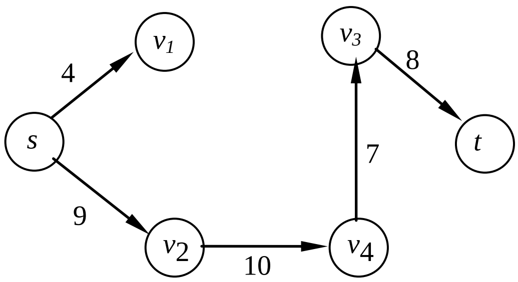
第二个剩余网络 G_f



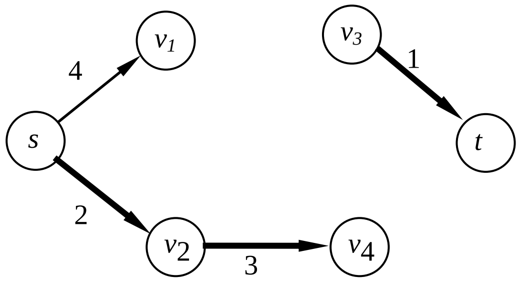
第二个 MSN



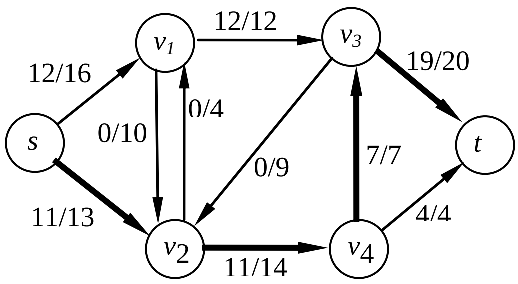
MSN 上的阻塞流 $f^*=7$



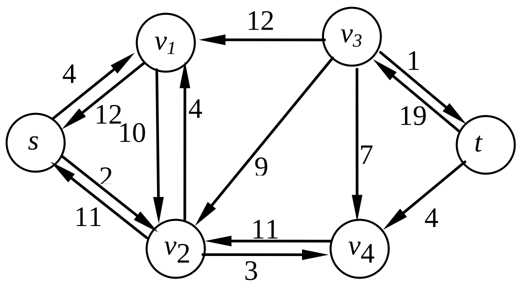
第二个 MSN



第二轮迭代后 MSN 上的阻塞流 $f^*=7$



得到阻塞流 f^*后最终的流 $f'=23$



第二个剩余网络 G_f

从上面的例子可以看出，Dinic 算法迭代计算了 2 次，而 Karp 算法迭代计算了 3 次。