网络流

Jun Wu

wujun@yzu.edu.cn

May 7, 2018

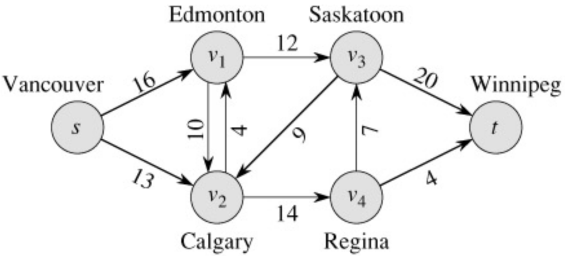
流网络

Definition 1 (流网络)

有向图$G = (V, E)$，如果图G满足：

- 存在源结点(source) s(s的入度为0);
- 存在汇结点(sink) t(t的出度为0);
- 任意结点$v \in V$，有$s \rightarrow v \rightarrow t$;
- 容量函数$c : V \times V \rightarrow R^{\geq 0}$;(若$(u, v) \notin E$则$c(u, v) = 0$.)

称G为流网络。

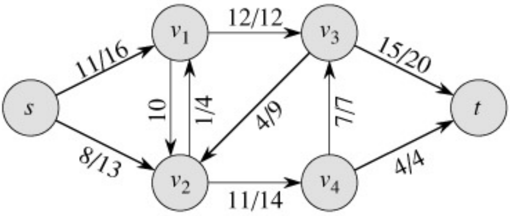


Definition 2 (网络流)

流网络$G = (V, E, c)$。映射$f : V \times V \to R$。若f满足下列三个性质：

- 容量限制：$\forall u, v \in V : f(u, v) \leq c(u, v)$;
- 反对称性：$\forall u, v \in V : f(u, v) = -f(v, u)$;
- 守恒性：$\forall u \in V \setminus \{s, t\} : \sum_{v \in V} f(u, v) = 0$;

则称f为G上的网络流。



流值

- 记号：$f(X,Y)=\sum_{x\in X}\sum_{y\in Y}f(x,y)$。
- 流值：$|f|=f(s,V)$。

Lemma 3

设$G=(V,E)$是一个流网络，f是G中的一个流，那么下列成立：

- 对所有$X\subseteq V, f(X,X)=0;$
- 对所有$X,Y\subseteq V, f(X,Y)=-f(Y,X);$
- 对所有$X,Y,Z\subseteq V$,其中$X\cap Y=\emptyset$, 有$f(X\cup Y,Z)=f(X,Z)+f(Y,Z).$

- 该引理是上述记号的引申。

流值

Theorem 4

$$|f| = f(V, t).$$

Proof.

$$\begin{aligned}
|f| &= f(s, V) \\
&= f(V \setminus (V \setminus \{s\}), V) \\
&= f(V, V) - f(V \setminus \{s\}, V) \\
&= f(V, \{t\} \cup V \setminus \{s, t\}) \\
&= f(V, t) + f(V, V \setminus \{s, t\}) \\
&= f(V, t)
\end{aligned}$$

□

Definition 5 (Cut)

设流网络$G = (V, E)$。V的划分(S, T)称作流网络G的割当且仅当$s \in S \wedge t \in T$。

- 流经割(S, T)的流量：

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

- 割(S, T)的容量：

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

割的容量与流量示例

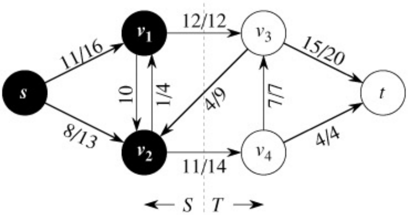


图: 割的例子

- $f(S,T) = f(v_1,v_3) + f(v_2,v_3) + f(v_2,v_4) = 12 - 4 + 11 = 19$
- $c(S,T) = c(v_2,v_3) + c(c_2,v_3) + c(v_2,v_4) = 12 + 0 + 14 = 26$
- $f(S,T) \leq c(S,T)$.

流值与割的容量

Theorem 6

设$G = (V, E, c)$为流网络，f为G中的流，而(S, T)为G的一个割。那么，

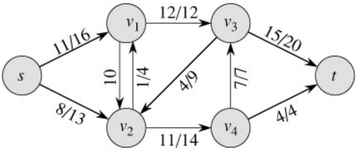
1. $|f| = f(S, T)$；
2. $|f| \leq c(S, T)$。

Proof.

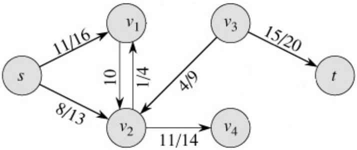
$$
\begin{aligned}
f(S,T) &= f(S, V \setminus S) \\
&= f(S,V) - f(S,S) \\
&= f(s,V) + f(S \setminus \{s\}, V) \\
&= |f|
\end{aligned}
$$

□

- 流本质上可以看成是由一组s到t的有向路合成的；
- 因此，我们可以从一个空的流f开始;
- 若在图上找到一条s到t的有向路P，就可以将P合并到f中，来扩大f;
- 重复这一过程，直到找不到s到t的有向路。但问题是：

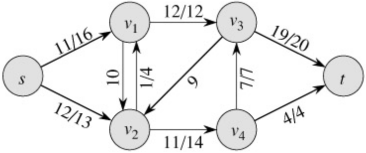


(a) 流f

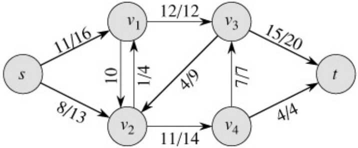


(b) 不存在容量>0的$s\to t$路

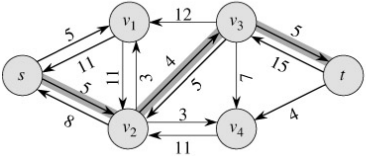
剩余网络



(c) (a)所示的流f不是最大流



(d) 流f



(e) f的剩余网络

图: 剩余网络的例子

Definition 7 (Residual network)

设流网络$G = (V, E, c)$，而f为G中的一个流。定义顶点对$u, v \in V$的剩余容量为$c_f(u, v) = c(u, v) - f(u, v)$。称$G_f = (V, E_f, c_f)$为网络$G$在流$f$下的剩余网络，其中

$$E_f = \{(u, v) \in V \times V | c_f(u, v) > 0\}.$$

- 注意：虽然$(u, v) \notin E$，但(u, v)可能属于E_f。
- 这时，$(v, u) \in E$，并且$f(v, u) > 0$。
- 若$(v, u) \notin E \wedge (u, v) \notin E$，那么$(v, u) \notin E_f \wedge (u, v) \notin E_f$。
- $|E_f| \leq 2|E|$。

增广路

Definition 8 (Augmenting path)

- 剩余网络G_f上一条$s \to t$路P称作流f的增广路。
- 增广路P的最大剩余容量定义为$c_f(P) = \min_{e \in P}\{c_f(e)\}$.
- 将增广路P合并到流f中的操作称为对流f的增广，增广后的流f'为

$$f'(u,v) = \begin{cases} f(u,v) + c_f(P), & \text{if } (u,v) \in P \\ f(u,v) - c_f(P), & \text{if } (v,u) \in P \\ f(u,v), & \text{otherwise} \end{cases} . \tag{1}$$

- 实际上，P对应了G_f中的一条流，记作f_P。
- $f_P(u,v) = c_f(P)$ and $f_P(v,u) = -c_f(P)$,if $(u,v) \in P$ else $f_P(u,v) = f_P(v,u) = 0$，而$|f_P| = c_f(P)$。
- 这样式(1)可以写成，$f' = f + f_P$。

Lemma 9

设f为流网络$G=(V,E,c)$中的流，P为G_f中的增广路，而f'为P对f的增广。那么，

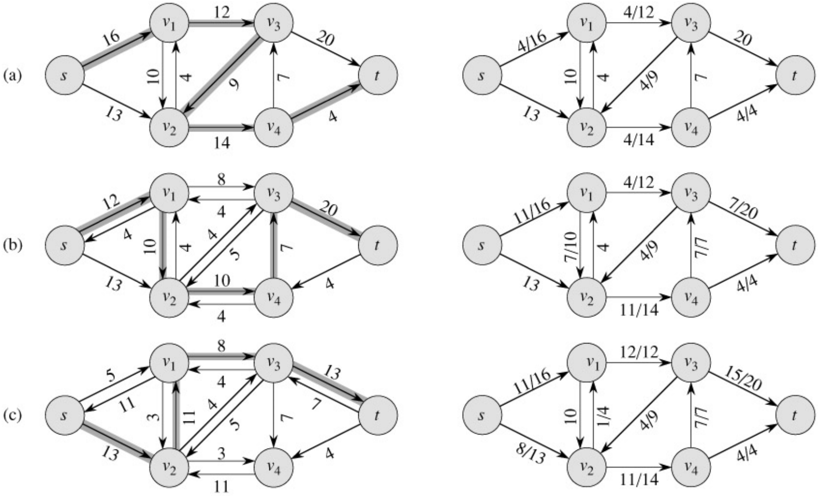
1. f'为G中的合法流；
2. $|f'|=|f|+c_f(P)$。

Proof.

- 容量限制：$f'(u,v)=f(u,v)+f_P(u,v)\le f(u,v)+(c(u,v)-f(u,v))=c(u,v)$;
- 反对称性：$-f'(u,v)=-f(u,v)-f_P(u,v)=f'(v,u)$;
- 守恒性：$\sum_{v\in V}f'(u,v)=\sum_{v\in V}f(u,v)+\sum_{v\in V}f_P(u,v)=0$;
- $|f'|=f'(s,V)=f(s,V)+f_P(s,V)=|f|+c_f(P)$.

$\square$

Ford-Fulkerson算法执行过程



Ford-Fulkerson算法执行过程

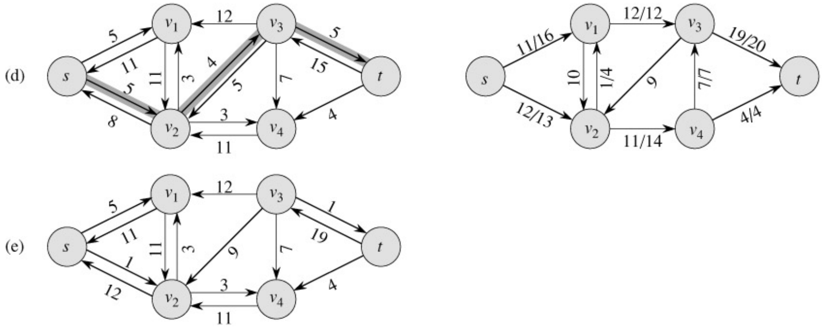


图: Ford-Fulkerson算法执行过程示例

FORD-FULKERSON(G, s, t)

for each edge $(u, v) \in E$ **do**

  $f[u, v] \leftarrow 0$

  $f[v, u] \leftarrow 0$

end for

while there exists a path P from s to t in the residual network G_f **do**

  $c_f(P) \leftarrow \min_{e \in P}\{c_f(e)\}$

  for each edge $(u, v) \in P$ **do**

    $f[u, v] \leftarrow f[u, v] + c_f(P)$

    $f[v, u] \leftarrow - f[u, v]$

  end for

end while

Ford-Fulkerson 算法的正确性

- 增广操作保证了算法的进展性，
- 而网络的边权重是有限的，因此算法一定停机。
- 因此，我们只需证明算法停止时，f中存放的是最大流。

Theorem 10 (最大流-最小割)

设流网络$G = (V, E)$，f为*Ford-Fulkerson*算法的输出，则下列三个命题等价：

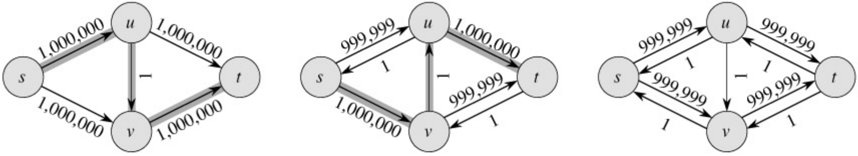
1. G_f中不存在增广路。
2. 存在割(S, T)使得$|f| = c(S, T)$。
3. f是最大流。

证明思路：

- $2 \Rightarrow 3$，$3 \Rightarrow 1$。只需证明$1 \Rightarrow 2$。
- 定义$S = \{u \in V | 在G_f中存在s到u的有向路\}$。
- 再证明$|f| = c(S, V \setminus S)$。

Ford-Fulkerson算法复杂度

- 算法的迭代次数取决于第5步while循环的次数；
- 最多$O(|f^*|)$次；
- 找增广路的复杂度$O(|E|)$。
- 算法总的复杂度为$O(|E||f^*|)$。
- 由于可能$|f^*| = \Omega(2^{|E|})$，所以算法是伪多项式的。



(a) (b) (c)

- Edmonds-Karp算法是Ford-Fulkerson算法的一种实现；与Ford- Fulkerson算法的唯一区别：以G_f中的最短路作为增广路。
- 设$f_0 \xrightarrow{P_1} f_1 \xrightarrow{P_2} f_2 \xrightarrow{P_3} \cdots \xrightarrow{P_k} f_k$。
- 定义$\delta_f(s,v)$为G_f中$s \to v$最短路的长度。

Lemma 11 (单调性引理)

$$\forall v \in V : \delta_{f_i}(s,v) \le \delta_{f_j}(s,v), 0 \le i < j \le k.$$

证明思路：

- 假设单调性引理不成立。设第一次违背单调性引理地方发生在剩余网络G_f处。
- u是G_f中违背单调性引理的点中$\delta_f(s,u)$最小的点。
- 设$f' \xrightarrow{P} f$，则$\delta_f(s,u) < \delta_{f'}(s,u)$ 。设v为P中u前面的点。分两种情况讨论：

单调性引理证明：Case 1

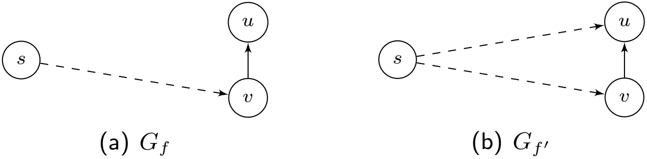


图: Case 1:$(v, u) \in G_{f'}$

$$
\begin{aligned}
\delta_f(s, v) &= \delta_f(s, u) - 1 && \text{property of shortest paths} \\
&< \delta_{f'}(s, u) - 1 && \text{assumption} \\
&\leq \delta_{f'}(s, v) + 1 - 1 && \text{triangle inequality} \\
&= \delta_{f'}(s, v)
\end{aligned}
$$

- 这与u是G_f中违背单调性引理的点中从s到达的距离最短的点矛盾。

单调性引理证明：Case 2

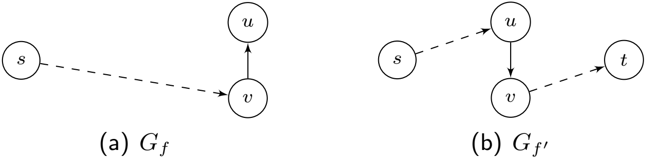


图: Case 2:$(v, u) \notin G_{f'} \Rightarrow (u, v) \in P$

$$
\begin{array}{rcll}
\delta_f(s,u) & = & \delta_f(s,v) + 1 & \text{property of shortest paths} \\
& \geq & \delta_{f'}(s,v) + 1 & \text{assumption} \\
& = & \delta_{f'}(s,u) + 1 + 1 & \text{property of shortest paths} \\
& = & \delta_{f'}(s,u) + 2 &
\end{array}
$$

- 这与u违背单调性引理矛盾。

□

临界边

Definition 12 (Critical edge)

设流网络G，流f和G_f中的增广路P。若边$(u,v) \in P$满足$c_f(u,v) = c_f(P)$，则称(u,v)是增广路P中的临界边。

- 注意：根据$c_f(P)$的定义，每次增广时至少有一条边是临界边。

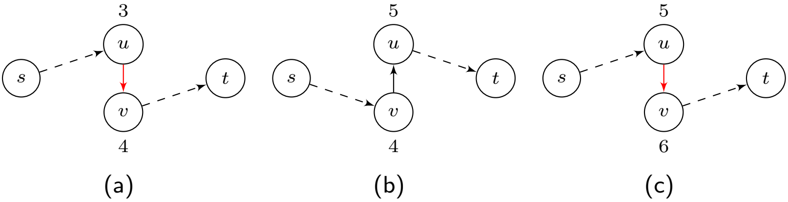


图: (u,v)相邻两次成为临界边

Edmonds-Karp算法的复杂度

Theorem 13

*Edmonds-Karp*算法的迭代次数是$O(|E||V|)$。

Proof.

- 每条边成为临界边的次数最多$O(|V|)$。
- 最多有$2|E|$条边，故总共最多有$O(|E||V|)$次临界边。
- 每一次迭代至少有一条临界边。

$\square$

- **推论：** Edmonds-Karp算法的复杂度$O(|V||E|^2)$。

算法思路

- 将流网络$G = (V, E)$看成管道网络；
- 水流将从高处往地处流；算法赋予每个节点一个高度值(h)；
- 初始时$h[s] = |V|$，其它顶点高度为0；
- 水流从高处往地处放，称为Push操作；
- 修改顶点高度，称为Relabel操作；
- 初始时，水流将尽可能多的从s流向它的邻居；通过逐步调整顶点高度使得水流流向全网络；
- 算法执行中，网络中的流不再满足守恒条件，我们将满足容量限制和反对称条件的流称为预流(pre-flow)；
- 算法为每个顶点设定一蓄水池(e)，用于存放溢出的流量。

高度函数

Definition 14

设流网络$G = (V, E)$，f为G中预流。映射$h : V \to \mathcal{N}$称作高度函数，当且仅当

1. $h(s) = |V|$;
2. $h(t) = 0$;
3. $h(u) \leq h(v) + 1, \forall (u, v) \in E_f$.

- 根据高度的定义，我们立刻可得下述引理：

Lemma 15

任意顶点对$u, v \in V$，如果$h(u) > h(v) + 1$，那么$(u, v) \notin E_f$。

INITIALIZE-PREFLOW(G, s)

for each $u \in V$ **do**
    $h[u] \leftarrow 0$
    $e[u] \leftarrow 0$
end for
for each $(u, v) \in E$ **do**
    $f(u, v) \leftarrow 0$
    $f(v, u) \leftarrow 0$
end for
$h[s] \leftarrow |V|$
for each $v \in Adj(s)$ **do**
    $f(s, v) \leftarrow c(s, v)$
    $f(v, s) \leftarrow -f(s, v)$
    $e[v] \leftarrow c(s, v)$
    $e[s] \leftarrow e[s] - c(s, v)$
end for

PUSH(u, v)

$\triangleright$ **Applies when:** u is overflowing, $c_f(u, v) > 0$, and $h[u] = h[v] + 1$.

$\triangleright$ **Action:** Push $d_f(u, v) = \min\{e[u], c_f(u, v)\}$ units of flow from u to v.

$d_f(u, v) \leftarrow \min\{e[u], c_f(u, v)\}$

$f(u, v) \leftarrow f(u, v) + d_f(u, v)$

$f(v, u) \leftarrow -f(u, v)$

$e[u] \leftarrow e[u] - d_f(u, v)$

$e[v] \leftarrow e[v] + d_f(u, v)$

RELABEL(u)

▷ **Applies when:** u is overflowing and for all $v \in V$ such that $(u, v) \in E_f$, we have $h[u] \leq h[v]$.

▷ **Action:** Increase the height of u.

$h[u] \leftarrow 1 + \min\{h(v) | (u, v) \in E_f\}$

GENERIC-PUSH-RELABEL(G, s, t)

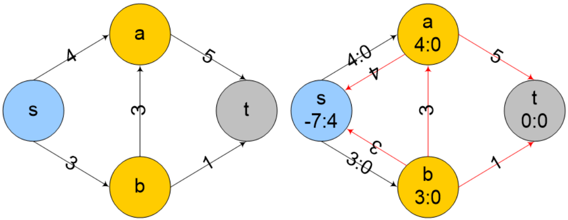
INITIALIZE-PREFLOW(G, s)

while there exists an applicable push or relabel operation **do**

    Select an applicable push or relabel operation and perform it

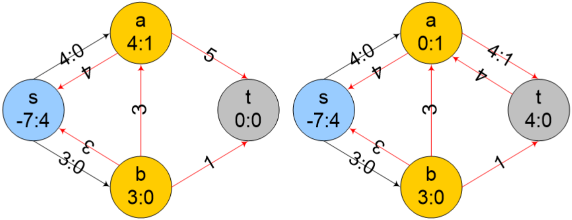
end while

Push-Relabel算法执行过程



INITIALIZE-PREFLOW(*G*, *s*)

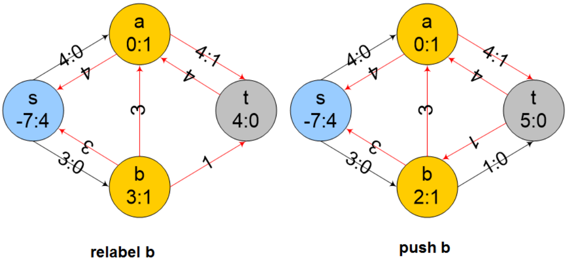
(a)



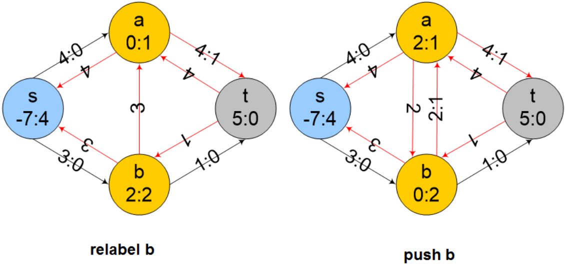
relabel a **push（a，t）**

(b)

Push-Relabel算法执行过程

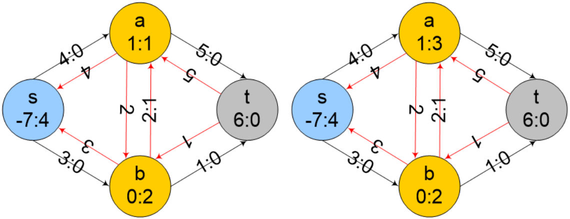


(c)



(d)

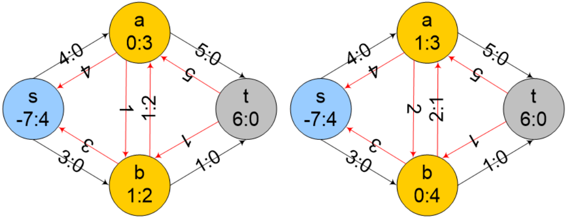
Push-Relabel算法执行过程



push a

relabel a

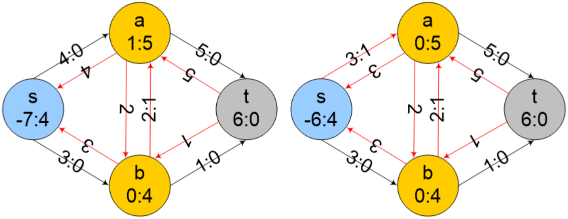
(e)



push a

relabel b and push b

(f)



relabel a　　push a

(g)

图: Push-Relabel执行过程示例

几个观察

Lemma 16

在算法*GENERIC-PUSH-RELABEL*执行过程中，如下结论成立：

1. h始终保持为高度函数。
2. 任意顶点的高度是非降的。
3. $u \in V$，若$e[u] > 0$，那么对u要么可以进行*PUSH*要么可以进行*RELABEL*。
4. G_f中无$s \to t$路。

证明思路：

1. 对迭代步数进行归纳。
2. PUSH不改变高度，RELABEL只增加高度。
3. PUSH和RELABEL操作的前提条件和高度函数决定的。
4. 反证，如果有则与s的高度矛盾。

GENERIC-PUSH-RELABEL算法的正确性

Theorem 17

*GENERIC-PUSH-RELABEL*算法结束时，f是G中的最大流。

证明思路：

- 循环不变量：f是预流。
- 根据引理16-3，$\forall u \in V \setminus \{s, t\} : e[u] = 0$，
- 因此，f是合法流。
- 再根据引理16-4和最大流-最小割定理知，f是最大流。

RELABEL操作次数的界

Lemma 18

在*GENERIC-PUSH-RELABEL*执行过程中，若$u \in V : e[u] > 0$，那么在G_f中存在路径$u \to s$。

证明思路：

- 定义$U = \{v \in V | u \to v\}$，$\bar{U} = V \setminus U$；
- $\forall v \in U, \forall w \in \bar{u} : f(w, v) \leq 0$。
- 假设引理不成立，即$s \notin U$;
- $e[U] = f(V, U) = f(\bar{U}, U) + f(U, U) = f(\bar{U}, U) \leq 0$；与$e[u] > 0$矛盾。

Lemma 19

在*GENERIC-PUSH-RELABEL*执行过程中，$\forall u \in V : h[u] \leq 2|V| - 1$。

证明思路：对RELABEL操作进行归纳。

- **推论：** RELABEL操作的次数是$O(|V|^2)$。

饱和PUSH的界

- 若PUSH(u,v)后$c_f(u,v)=0$，称为饱和PUSH；否则是非饱和PUSH。

Lemma 20

在*GENERIC-PUSH-RELABEL*执行过程中，饱和*PUSH*的总次数是$O(|V||E|)$。

Proof.

- 考虑边(u,v)饱和PUSH的次数：
- 饱和PUSH(u,v)后$(u,v)\notin E_f$；
- 下一次PUSH(u,v)时，$h[u]$至少加2；
- 而$h[u]$最多为$2|V|-1$，所以边(u,v)的饱和PUSH此时最多$O(|V|)$。

$\square$

不饱和PUSH的界

Lemma 21

在*GENERIC-PUSH-RELABEL*执行过程中，不饱和*PUSH*的总次数是$O(|V|^2|E|)$。

证明思路：

- 定义$\Phi = \sum_{v:e[v]>0} h[v]$。算法初始和退出时$\Phi = 0$；
- RELABEL：总共增加Φ值$O(|V|^2)$个单位；
- Saturating PUSH：总共增加Φ值$O(|V||E|) \times O(|V|)$；
- Non-Saturating PUSH：每次减少Φ值至少1。

Theorem 22

*GENERIC-PUSH-RELABEL*算法的复杂度为$O(|V|^2|E|)$。

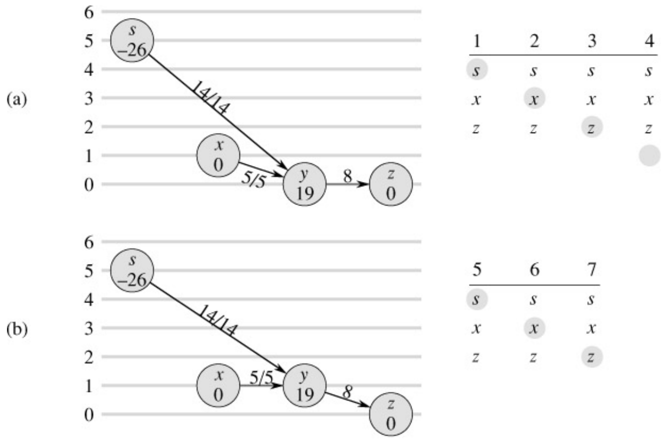
算法思路

- Relabel-to-front算法是Push-Relabel算法的改进；
- 当发现某个点u是溢出，那么连续对u进行PUSH或RELABEL直至u不再溢出；
- 这一操作称作排空(discharge)；
- 维护一个适当的顶点排空顺序；
- 算法以邻接表作为剩余网络的数据结构(看成无向图)；
- $N[u]$为顶点u的表头结点；
- $current[u]$为点u的邻居链表的当前结点。

DISCHARGE(u)

while $e[u] > 0$ **do**
    $v \leftarrow current[u]$
    if $v = nil$ **then**
        RELABEL(u)
        $current[u] \leftarrow head[N[u]]$
    else
        if $c_f(u,v) > 0 \wedge h[u] = h[v] + 1$ **then**
            PUSH(u, v)
        else
            $current[u] \leftarrow next\text{-}neighbor[v]$
        end if
    end if
end while

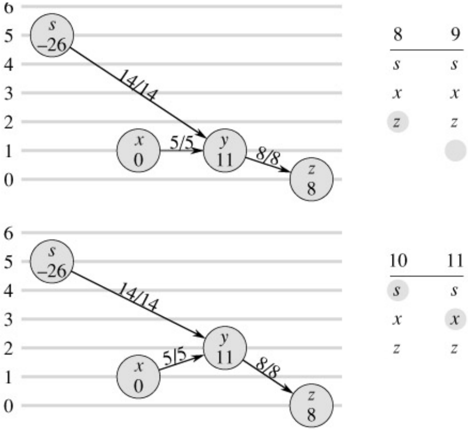
DISCHARGE操作过程



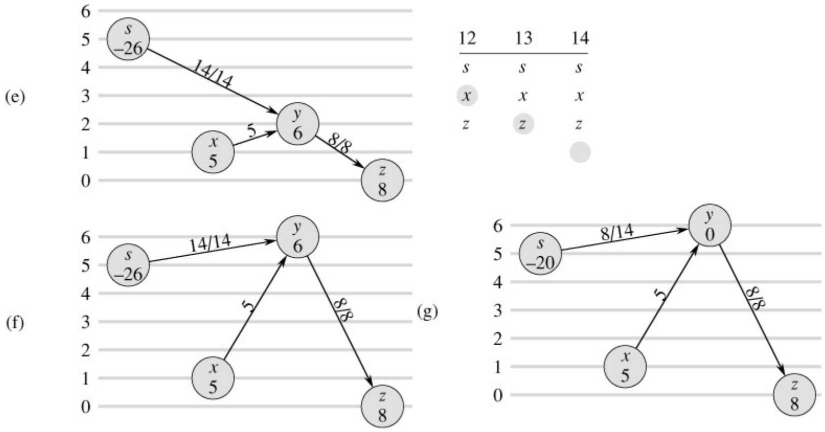
DISCHARGE操作过程

(c)

(d)



DISCHARGE操作过程



RELABEL-TO-FRONT(G, s, t)

INITIALIZE-PREFLOW(G, s)
$L \leftarrow V \setminus \{s, t\}$, in any order
for each vertex $u \in V \setminus \{s, t\}$ **do**
    $current[u] \leftarrow head[N[u]]$
end for
$u \leftarrow head[L]$
while $u \neq nil$ **do**
    $old\text{-}height \leftarrow h[u]$
    DISCHARGE(u)
    if $h[u] > old\text{-}height$ **then**
        move u to the front of list L
    end if
    $u \leftarrow next[u]$
end while

Admissible网络

Definition 23

流网络$G=(V,E,c)$，f是G中的预流，并且h是顶点的高度函数。边$(u,v)\in E_f$是*Admissible*的当且仅当

1. $c_f(u,v)>0$;
2. $h[u]=h[v]+1$.

由所有*Admissible*边构成的网络称为*Admissible*网络。

Lemma 24

*Admissible*网络是*DAG*图。

证明思路：反证。如果有环，则

$$v_1,v_2,\cdots,v_k,v_1\Rightarrow h[v_1]>h[v_k]>h[v_1]$$

矛盾。

PUSH和RELABEL与Admissible边

Lemma 25

u为溢出点，(u,v)是*admissible*边。*PUSH*(u,v)可能使得(u,v)变为非*admissible*，但不会增加新的*admissible*边。

证明思路：

- PUSH(u,v)只改变(u,v)和(v,u);
- 如果是饱和PUSH，过后$c_f(u,v)=0$，(u,v)不再是admissible；
- $h[v]=h[u]+1<h[u]$，因此，(v,u)不可能变为admissible。

Lemma 26

u为溢出点且没有从u发出的*admissible*边。*RELABEL*(u)后至少有一条*admissible*边离开u，但没有*admissible*边进入u。

证明思路：

- 至少有一条离开的admissible边，显然；
- 若(v,u)在RELABEL(u)后变为admissible，则$h(v)>h(u)+1$，根据引理15，$(v,u)\notin E_f$。

排空操作的正确性

Lemma 27

1. *DISCHARGE*(u)调用*PUSH*(u,v)时，*PUSH*适用于(u,v)；
2. *DISCHARGE*(u)调用*RELABEL*(u)时，*RELABEL*适用于u。

Proof.

- DISCHARGE的第一行检查了$e[u] > 0$; 第七行检查了$c_f(u,v) > 0$和$h[u] = h[v]+1$;因此(1)部分成立。
- 前一次DISCHARGE(u)结束于$current[u]$(不饱和PUSH)，
- 因此$head[N[u]]$到$current[u]$之间的边都不是admissible的。
- 如果在两次DISCHARGE之间有过PUSH(w,u)，那么$h[w] > h[u]$。故(2)成立。

□

Relabel-to-front算法的正确性

Theorem 28

*RELABEL-TO-FRONT*终止时，f是最大流。

证明思路：

- 引理27表明，Relabel-to-front算法也是一种Push-Relabel算法。
- 因此，只要证明在算法终止时除s, t外，网络中没有溢出点。
- 循环不变量：在第7步循环检查$u \neq nil$时，表L中存放的是当前admissible网络的一个拓扑排序，且u之前的顶点皆不溢出。
- 初始时，没有admissible边，因此L中的顺序是一个拓扑排序；
- 引理25表明，PUSH操作不改变拓扑排序；
- 引理26表明，RELABEL(u)后将u移至L最前面将维持L为拓扑排序。

Relabel-to-front的复杂度

Theorem 29

*RELABEL-TO-FRONT*对任意流网络$G = (V, E, c)$的运行时间为$O(|V|^3)$。

证明思路：

- 相邻两次RELABEL期间称作一个phase。
- $O(|V|^2)$次RELABEL，因此有$O(|V|^2)$个phases。
- 每个phase期间最多调用$|V|$次DISCHARGE。因此，DISCHARGE的调用最多$O(|V|^3)$。
- DISCHARGE主要有三个操作：第4行RELABEL，第8行PUSH和第10行更新$current[u]$。
- $|V|^2$次RELABEL可以在$O(|V||E|)$时间内完成。
- 每次RELABEL时，$current[u]$回到链表头，即每个链表最多扫描$|V|$次，同样可以在$O(|V||E|)$时间内完成。
- 饱和PUSH共$O(|V||E|)$次，每次不饱和PUSH将导致下一次DISCHARGE，因此不饱和PUSH最多$O(|V|^3)$。