Maximum Flow

Reading: CLRS Chapter 26.

CSE 6331 Algorithms

Steve Lai

Flow Network

- A flow network $G = (V, E)$ is a directed graph with
  - a source node $s \in V$,
  - a sink node $t \in V$,
  - a capacity function c.
- Each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$.
- If $(u, v) \notin E$, assume $c(u, v) = 0$.
- Also, assume that every node v is on some path from s to t.
  - This implies $O(V + E) = O(E)$.
  - A maxflow may only go through such nodes.

Flow

- Let $G = (V, E)$ be a flow network with capacity function c, source node s, and sink node t.
- A flow is a real-valued function $f : V \times V \to \mathbb{R}$ satisfying
  - Capacity constraint: $\forall u, v \in V,\ f(u,v) \le c(u,v)$.
  - Skew symmetry: $\forall u, v \in V,\ f(u,v) = -f(v,u)$.
  - Flow conservation: $\forall u \in V - \{s,\ t\}$,

$$f(u,V) \triangleq \sum_{v \in V} f(u,v) = 0$$

- The value of a flow f is $|f| \triangleq f(s,V) = \sum_{v \in V} f(s,v)$.
- The maxflow problem is to find a flow of maximum value.

Some Properties of Flows

- If no edge between u and v, then $f(u,v) = f(v,u) = 0$.
- Flow conservation implies: $\forall u \in V - \{s,\ t\}$,

  Total positive flow into $u =$ Total positive flow out of u.
- For $X, Y \subseteq V$, define $f(X,Y) = \sum_{x \in X} \sum_{y \in Y} f(x,y)$.
- $f(X,X) = 0$.
- $f(X,Y) = -f(Y,X)$.
- $f(X \cup Y, Z) = f(X,Z) + f(Y,Z)$, if $X \cap Y = \varnothing$.
- $f(Z, X \cup Y) = f(Z,X) + f(Z,Y)$, if $X \cap Y = \varnothing$.

Residual networks and augmenting paths

- Let $G = (V, E)$ be a flow network and f a flow.
- Residual capacity of (u,v) is

  $$c_f(u,v) = c(u,v) - f(u,v).$$

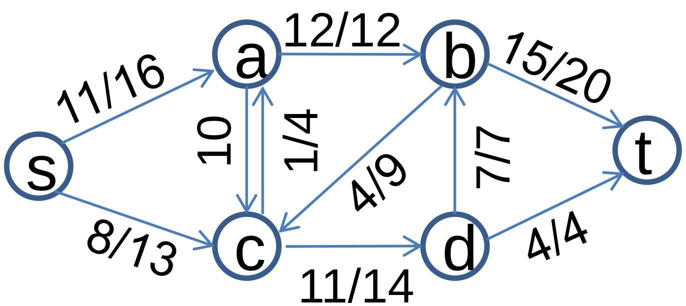
- Residual network induced by f is $G_f = (V, E_f)$, where

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

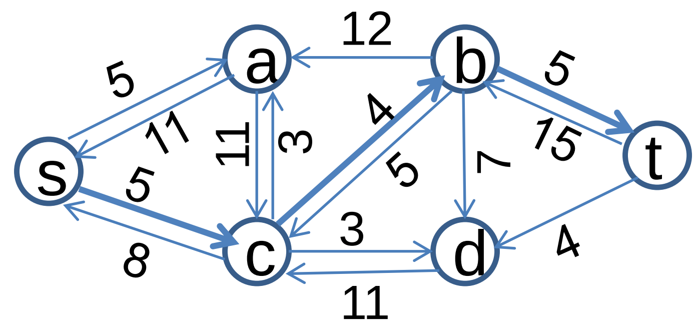
- Augmenting path: a simple path p from s to t in G_f.
- Residual capacity of p:

  $$c_f(p) = \min\{c_f(u,v) : (u,v) \text{ is in } p\}.$$

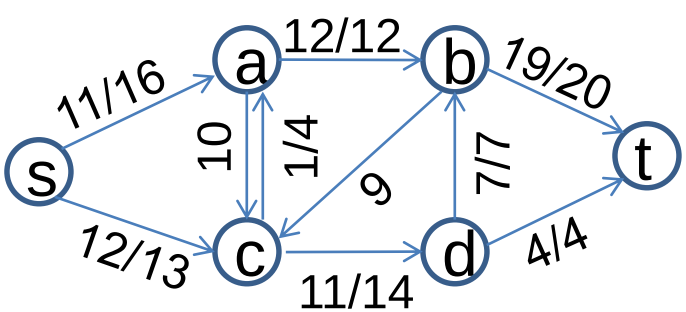
Example



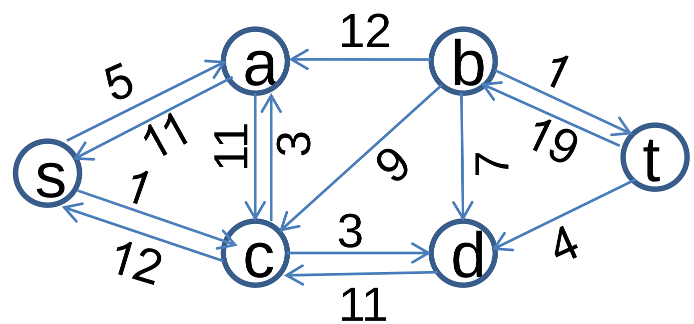
(a) Flow network and flow



(b) Residual network and augmenting path *p* with $c_f(p) = 4$



(c) Augmented flow



(d) No augmenting path

Flow: an alternative definition (CLRS, 3rd ed.)

- Let $G=(V,E)$ be a flow network with a capacity function c, source s, and sink t. Assume G has no parallel edges, i.e., if $(u,v)\in E$ then $(v,u)\notin E$.
- A flow is a real-valued function $f:V\times V\rightarrow R$, satisfying
  - Capacity constraint: $\forall u,v\in V,\ 0\leq f(u,v)\leq c(u,v)$.
  - Flow conservation: $\forall u\in V-\{s,\ t\}$,

$$\sum_{v\in V} f(v,u)=\sum_{v\in V} f(u,v) \qquad \left(\text{i.e.}\quad f\left(V,u\right)=f\left(u,V\right)\right)$$

- The value of a flow is $\left|f\right|=f\left(s,V\right)-f\left(V,s\right)$.
- Note: when $(u,v)\notin E$, $f(u,v)=0$.

Some of these properties do not hold any more (when using the second definition of flows)

- If no edge between u and v, then $f(u,v) = f(v,u) = 0$.
- Flow conservation implies: $\forall u \in V - \{s,\ t\}$,

  Total positive flow into u = Total positive flow out of u.
- For $X, Y \subseteq V$, define $f(X,Y) = \sum_{x \in X} \sum_{y \in Y} f(x,y)$.
- $f(X,X) = 0$.
- $f(X,Y) = -f(Y,X)$.
- $f(X \cup Y, Z) = f(X,Z) + f(Y,Z)$, if $X \cap Y = \varnothing$.
- $f(Z, X \cup Y) = f(Z,X) + f(Z,Y)$, if $X \cap Y = \varnothing$.

Residual networks and augmenting paths (using the second definition of flows)

- Let $G=(V,E)$ be a flow network and f a flow.
- Residual capacity of (u,v) is

$$c_f(u,v) == \begin{cases} c(u,v)-f(u,v) & \text{if } (u,v)\in E \\ f(v,u) & \text{if } (v,u)\in E \\ 0 & \text{otherwise} \end{cases}$$

- Residual network induced by f is $G_f=(V,E_f)$, where

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

- Augmenting path: a simple path p from s to t in G_f.

Ford-Fulkerson Method

Given a flow network $G = (V, E)$ with source s and sink t,

Initialize flow f to 0

while there exists an augmenting path p **do**

do augment flow f along p

return f

Ford-Fulkerson(G, s, t)

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

  do $f(u, v) \leftarrow 0$

   $f(v, u) \leftarrow 0$

while there exists an augmenting path p in residual network G_f

  do $c_f(p) = \min\{c_f(u, v) : (u, v) \text{ is on } p\}$

   for each edge (u, v) is in p

    do $f(u, v) \leftarrow f(u, v) + c_f(p)$

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

Analysis

- The running time depends on how the augmenting path p is determined.
- If capacities are integers, the running time is $O\left(E\left|f^*\right|\right)$, where $\left|f^*\right|$ is the value of the maxflow.
  - Each iteration can be done in $O\left(E\right)$ time.
  - There are at most $\left|f^*\right|$ iterations.

Integrality Theorem. If all capacities are integers, the flow f produced by the Ford-Fulkerson method has the property that $f(u,v)$ is an integer for all $u,v \in V$.

Lemma 1. Let G_f be the residual network induced by flow f. Let f' be a flow in G_f. Then $f+f'$ is a flow in G with $|f+f'| = |f| + |f'|$.

Proof. • Skew symmetry:

$$
\begin{aligned}
(f+f')(u,v) &= f(u,v) + f'(u,v) \\
&= -f(v,u) - f'(v,u) \\
&= -(f+f')(v,u)
\end{aligned}
$$

• Capacity constraint:

$$
\begin{aligned}
(f+f')(u,v) &= f(u,v) + f'(u,v) \\
&\le f(u,v) + c_f(u,v) \\
&= f(u,v) + \big(c(u,v) - f(u,v)\big) \\
&= c(u,v)
\end{aligned}
$$

- Flow conservation: for all $u \in V - \{s,t\}$,

$$
\begin{aligned}
(f + f')(u,V) &= f(u,V) + f'(u,V) \\
&= 0 + 0 = 0
\end{aligned}
$$

- Finally,

$$
\begin{aligned}
|f + f'| &= (f + f')(s,V) \\
&= f(s,V) + f'(s,V) \\
&= |f| + |f'|
\end{aligned}
$$

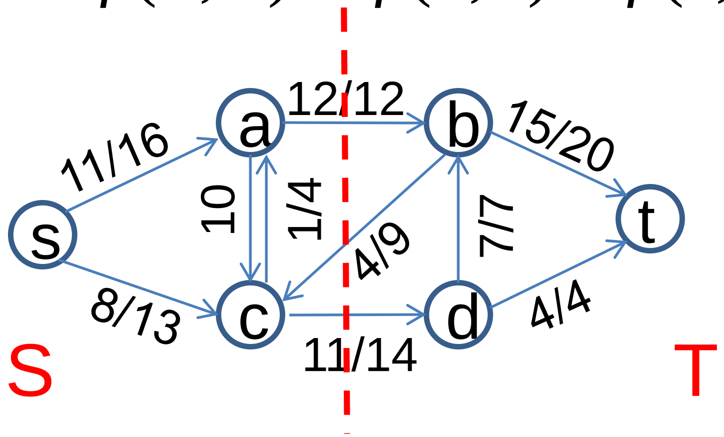
Lemma 2. If p is an augmenting path in G_f, then augmenting f along p yields a flow in G with value $|f| + c_f(p) > |f|$.

Corollary 3. The f produced by Ford-Fulkerson is a flow.

Cuts

- A cut (S,T) of a flow network $G=(V,E)$ is a partition of of V into S and $T=V-S$ such that $s\in S$ and $t\in T$.
- If f is a flow, $f(S,T)$ denotes the net flow across the cut (S,T).
- The capacity of (S,T) is $c(S,T)=\sum_{u\in S}\sum_{v\in T}c(u,v)$.
- Example: $c(S,T)=c(a,b)+c(c,d)=12+14=26.$

$$f(S,T)=f(a,b)+f(c,d)+f(c,b)=12+11-4=19.$$



Lemma 4. For any cut (S,T), $|f| = f(S,T)$.

Proof. Note that $f(u,V) = 0 \ \ \forall\ u \neq s,t.$

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

Corollary 5. $|f| \leq c(S,T)$.

Proof. $|f| = f(S,T) \leq c(S,T)$.

The Max-flow Min-cut Theorem.

Theorem. The following conditions are equivalent:

1. f is a maxflow in G.
2. The residual network G_f contains no augmenting paths.
3. $|f| = c(S,T)$ for some cut (S,T) in G. //minimum cut//

Proof. (3) $\Rightarrow$ (1): Immediately follows from Corollary 5.

(1) $\Rightarrow$ (2): Immediately follows from Lemma 2. (If G_f contains an augmenting path p, augmenting f along p will increase the flow.)

2. The residual network G_f contains no augmenting paths.

3. $|f| = c(S,T)$ for some cut (S,T) in G.

$(2) \Rightarrow (3)$:

Suppose G_f contains no augmenting path. Define

$S = \{v : \text{there is a path from } s \text{ to } v \text{ in } G_f\}$,

$T = V - S$.

(S,T) is a cut since $s \in S$ and $t \in T$ (no path from s to t in G_f).

For all $u \in S$, $v \in T$, we have $f(u,v) = c(u,v)$, since otherwise $(u,v) \in E_f$ and v would be in S. So, $f(S,T) = c(S,T)$.

By Lemma 4, $|f| = f(S,T) = c(S,T)$.

Edmonds-Karp Algorithm

- In the while loop of Ford-Fulkerson, find the augmenting path p with a breadth-first search, that is, the augmenting path is a shortest path from s to t in the residual network, where "shortest" is in terms of number of edges.
- Running time: $O\left(VE^2\right)$ (to be shown).

Analysis of the Edmonds-Karp Algorithm

Lemma 6. In the execution of Edmonds-Karp algorithm, for all $v \neq s,\ t,\ \delta_f(v)$ is nondecreasing with each flow augmentation where $\delta_f(v) =$ shortest distance (# edges) from s to v in G_f.

Proof. By contradiction. Assume the lemma is not true. Consider the first augmentation that decreases some $\delta_f(\cdot)$. Let f and f' be the flows just before and after the augmentation. Let v be the vertex s.t. $\delta_f(v) > \delta_{f'}(v)$ and $\delta_{f'}(v)$ is minimum among those nodes x with $\delta_f(x) > \delta_{f'}(x)$. Let p be a shortest path from s to v in $G_{f'}$, and let (u,v) be the last edge of p. So, $(u,v) \in E_{f'},\ \delta_{f'}(u) + 1 = \delta_{f'}(v),$ and $\delta_f(u) \leq \delta_{f'}(u)$.

- Case 1: $(u,v) \in E_f$. Then, $\delta_f(u)+1 \geq \delta_f(v)$, and then

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

a contradiction.

- Case 2: $(u,v) \notin E_f$. Now, $(u,v) \notin E$, but $(u,v) \in E_{f'}$.

This means, the augmenting path contains edge (v,u).

As Edmonds-Karp always augments flow along shortest paths,

(v,u) is the last edge of a shortest path from s to u in G_f.

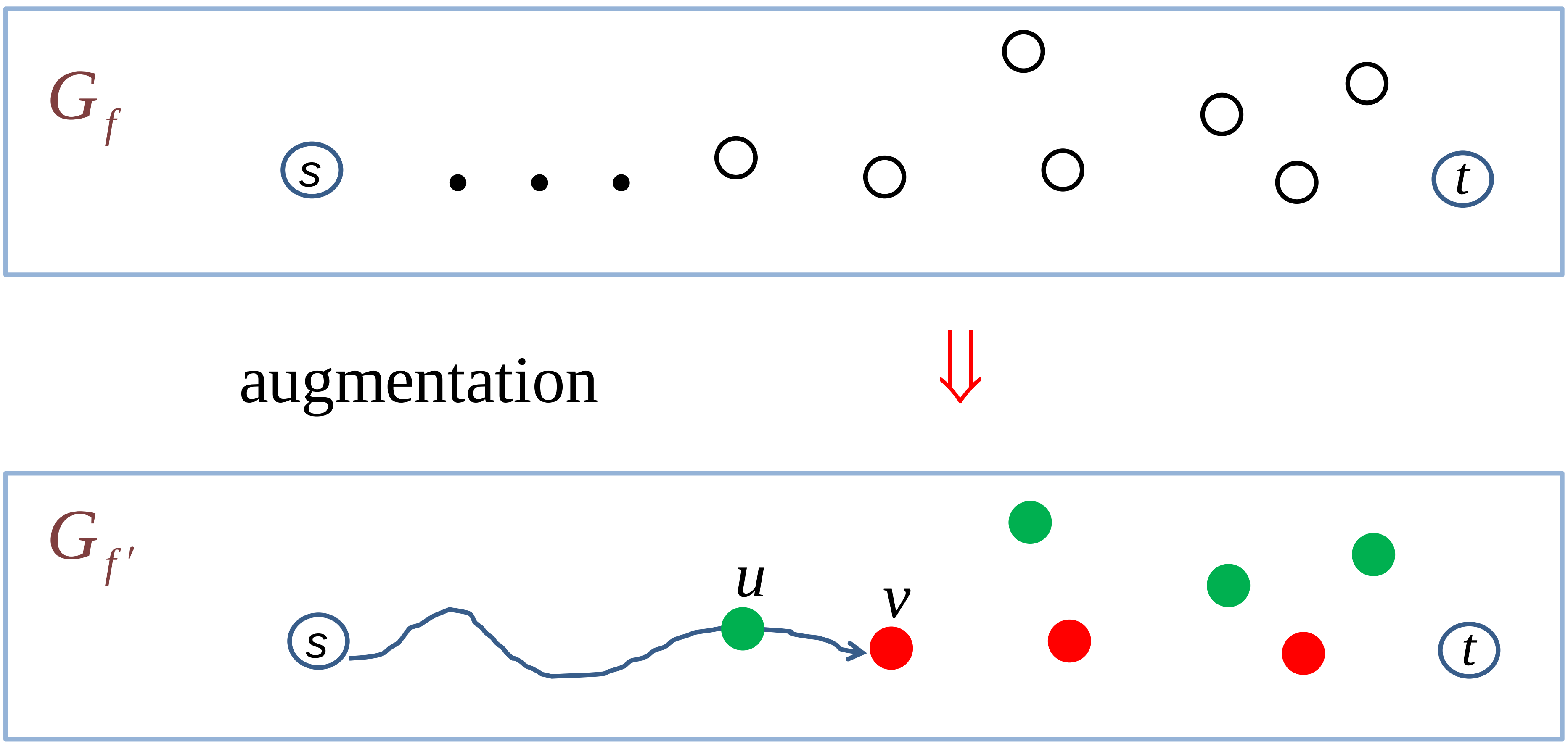
Therefore, $\delta_f(u) = \delta_f(v)+1 \Rightarrow \delta_f(u)+1 \geq \delta_f(v)$.

As in case 1, this will lead to a contrdiction.

$\delta(\bullet)$ decreases for red nodes; does not decrease for green nodes.

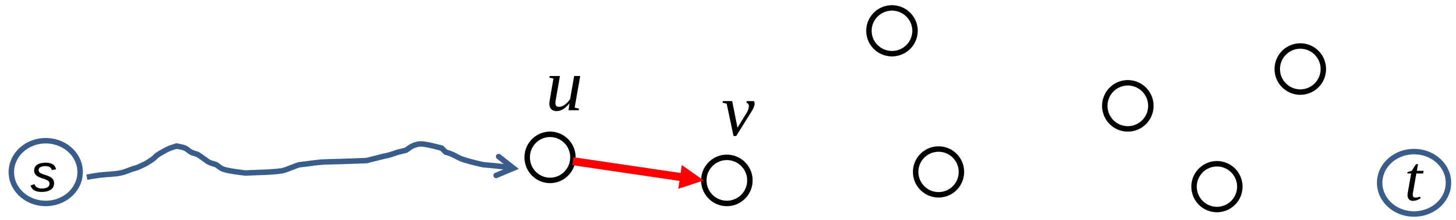
v : the red node closest to s in $G_{f'}$.

u : predecessor of v on shortest path s to v in $G_{f'}$; a green node.



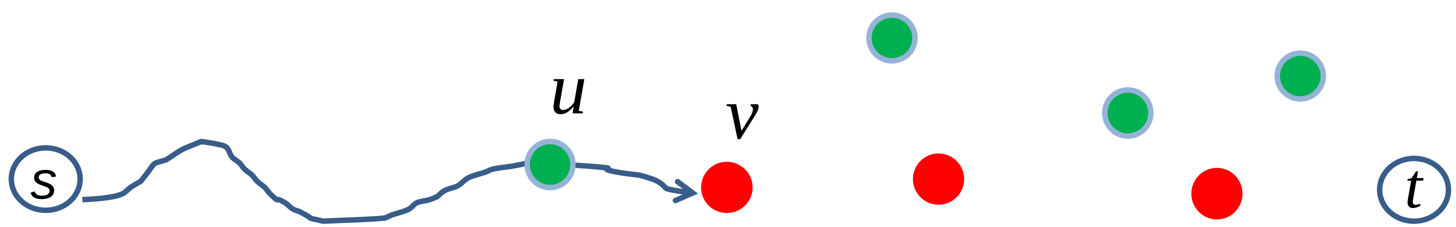
If edge (u, v) exists in G_f

G_f



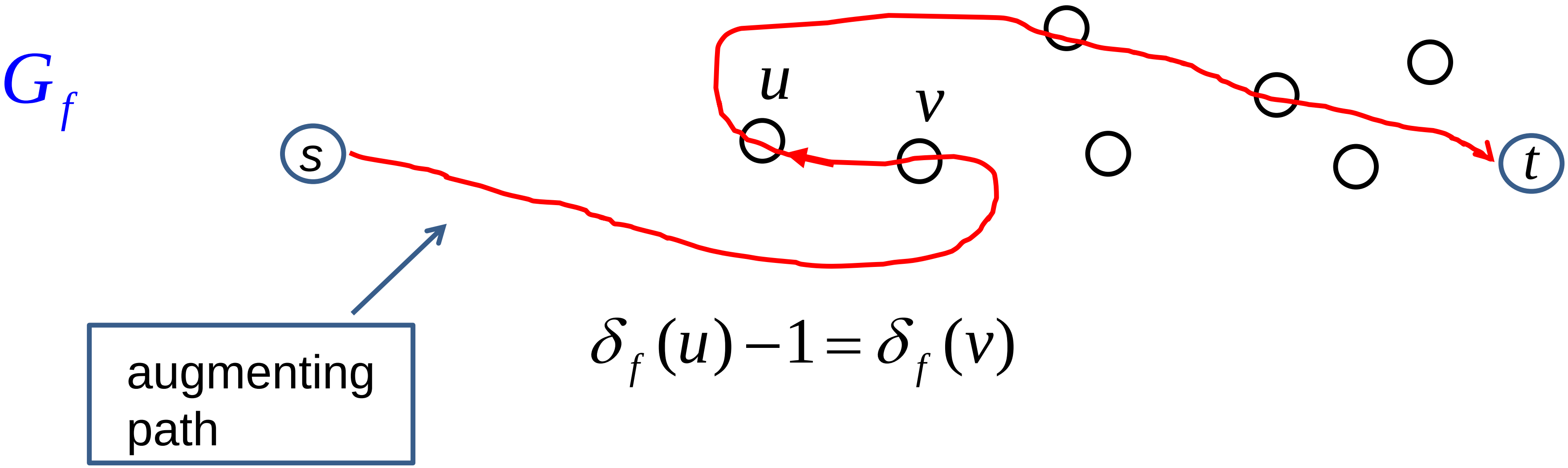
$$\delta_f(u) + 1 \geq \delta_f(v)$$

$G_{f'}$

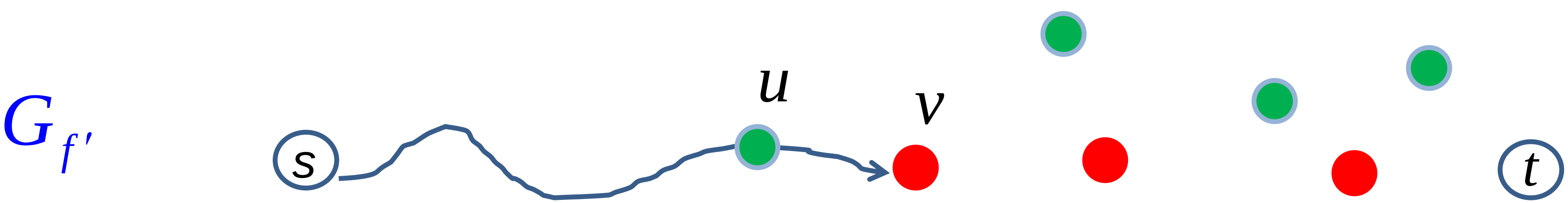


$$\delta_{f'}(u) + 1 = \delta_{f'}(v)$$

If edge (u, v) does not exist in G_f



$$\delta_f(u) - 1 = \delta_f(v)$$



$$\delta_{f'}(u) + 1 = \delta_{f'}(v)$$

Theorem 7. If Edmonds-Karp Algorithm runs on $G=(V,E)$, then the total number of flow augmentations is $O(VE)$ and hence the total running time is $O(VE^2)$.

Proof. An edge (u,v) in G_f is critical on an augmenting path p if $c_f(p)=c_f(u,v)$. Every augmenting path has a critical edge. An edge (u,v) may become critical only if $(u,v)\in E$ or $(v,u)\in E$. So there are at most $2|E|$ edges that may become critical during the algorithm's execution. We will show that each of these edges may become critical at most $|V|/2$ times, which will imply that during the execution of the Edmonds-Karp algorithm there are at most $O(VE)$ augmentations.

Claim: an edge (u, v) can become critical at most $|V|/2$ times.

- When (u, v) becomes critical in a residual network G_f,

  $\delta_f(v) = \delta_f(u) + 1. \quad (1)$

- After flow augmentation along the augmenting path, (u, v) disappears from the residual network.
- It may reappear later on another augmenting path only after $f(u, v)$ decreases, which occurs only if edge (v, u) appears on an augmenting path in some $G_{f'}$, in which case,

$$\begin{aligned} \delta_{f'}(u) = \delta_{f'}(v) + 1 \geq \delta_f(v) + 1 & \qquad \text{by Lemma 6} \\ \geq \delta_f(u) + 2 & \qquad \text{by (1).} \end{aligned}$$

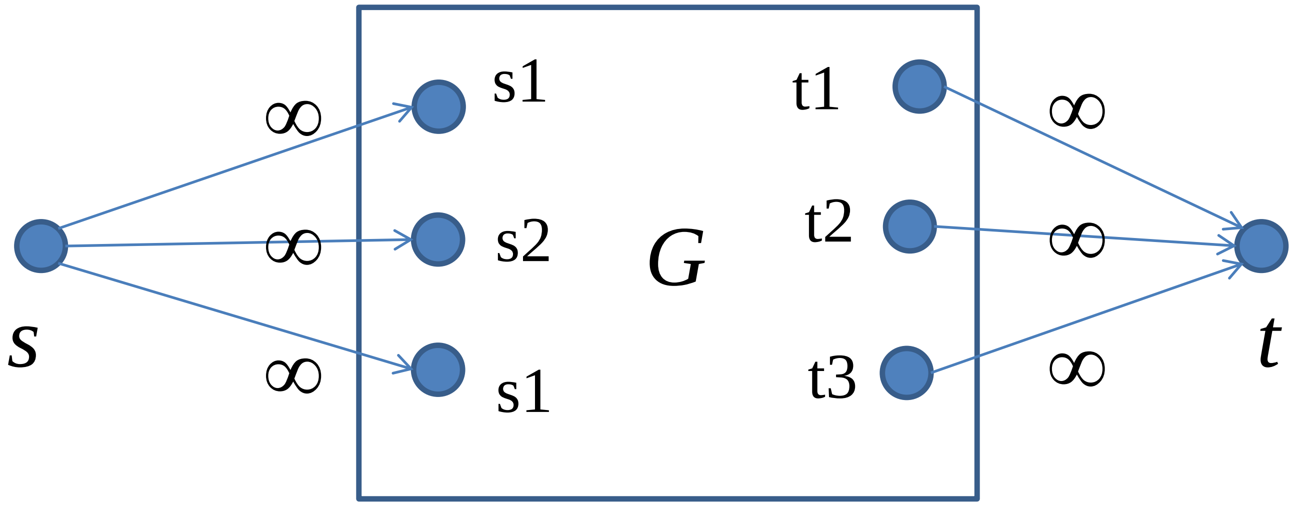
- Thus, if (u,v) becomes critical more than once, then for each additional time (u,v) becomes critical, $\delta(u)$ increases by at least 2.
- When (u,v) becomes critical for the last time, $\delta(u) \le |V| - 2.$
- Thus, (u,v) can become critical no more than $|V|/2$ times. This proves the claim and the theorem.

Networks with multiple sources and sinks

- $G=(V,E)$: flow network with

  m sources: $\{s_1, s_2, \ldots, s_m\}$

  n sinks: $\{t_1, t_2, \ldots, t_m\}$



Maximum Bipartite Matching

- $G=(V,E)$: undirected graph
- Bipartite graph: if V can be partitioned into L and R such that all edges in E go between L and R.
- Theorem: G is bipartite iff it has no cycles of odd length.
- Matching: a set of edges $M \subseteq E$ such that every vertex in V is an endpoint of at most one edge in M.
- Maximum matching: a matching with the max cardinality.
- A maximum matching of a bipartite graph can be found using the Ford-Fulkerson method.

Edge-Disjoint Paths

- $G=(V,E)$: a graph
- Edge-disjoint paths: two paths are edge-disjoint if they do not share any edges.
- Problem: Given a directed graph $G=(V,E)$ and two nodes s, t, find a maximum number of edge-disjoint paths from s to t.
- Problem: Given an undirected graph $G=(V,E)$ and two nodes s, t, find a maximum number of edge-disjoint paths from s to t.

Node-Disjoint Paths

- $G=(V,E)$: a graph
- Node-disjoint paths: two paths from s to t are node-disjoint if they do not share any intermediate nodes.
- Problem: Given a directed graph $G=(V,E)$ and two nodes s, t, find a maximum number of node-disjoint paths from s to t.
- Problem: Given an undirected graph $G=(V,E)$ and two nodes s, t, find a maximum number of node-disjoint paths from s to t.

Image Segmentation

- A fundamental problem in computer vision.
- Given a digital image (a set of pixels), we want to partition it into multiple segments.
- In a simple case, we just want to divide the image into two segments: the foreground and the background.
- Represent the image by an undirected graph $G = (V, E)$, where V is the set of pixels and there is an edge between two pixels iff there are neighbors.

- Each pixel i has a likelihood (goodness) $a_i > 0$ to belong to the foreground and a likelihood $b_i > 0$ to belong to the background.

- Each edge $(i, j) \in E$ is associated with a separation penalty $p_{ij} = p_{ji} > 0,$ which is incurred if pixels i and j are placed in different segments.

- Problem: Given a pixel graph $G=(V,E)$, likelihood functions $a,b:E\rightarrow\mathbb{R}^{+}$ and penalty function $p:E\rightarrow\mathbb{R}^{+}$, we want to partition V into two sets A and B and maximize

  $Q(A,B)=$

  $$\sum_{i\in A}a_i+\sum_{i\in B}b_i-\sum\left\{p_{ij}:(i,j)\in E,\ i,j\text{ in different segments}\right\}$$

- Or, equivalently, minimize

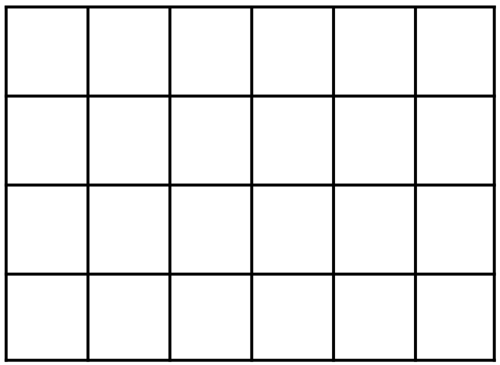
  $$Q'(A,B)=\left(\sum_{i\in V}a_i+\sum_{i\in V}b_i\right)-Q(A,B)$$

  $$=\sum_{i\in B}a_i+\sum_{i\in A}b_i+\sum\left\{p_{ij}:(i,j)\in E,\ i,j\text{ in different segments}\right\}$$

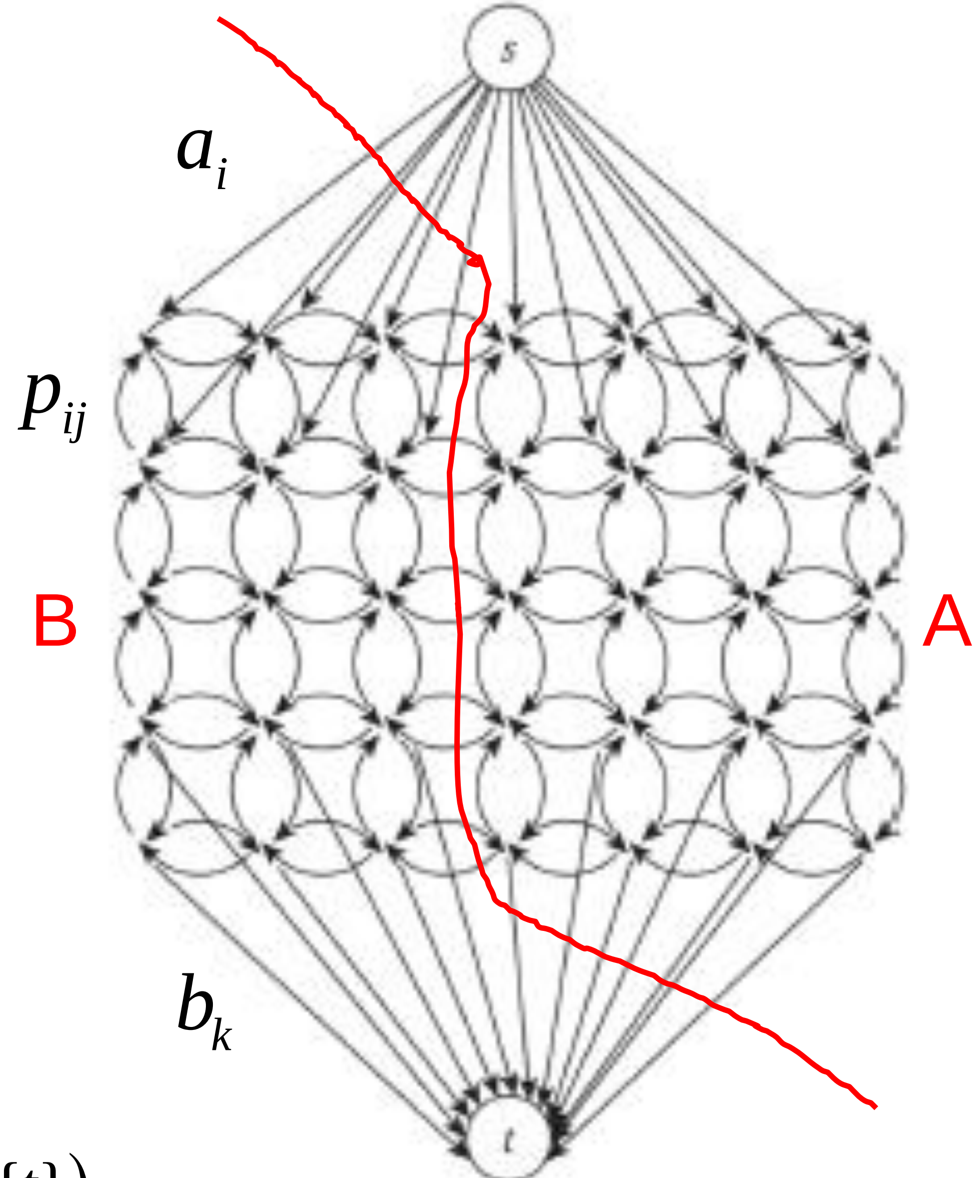
- We can solve the image segmentation problem by converting it to a flow network. Let $G=(V,E)$ be the pixel graph.
- Introduce two new vertices: a source s and a sink t.
- Connect s to each pixel $i \in V$ with capacity a_i.
- Connect t from each pixel $i \in V$ with capacity b_i.
- Replace each edge $(i, j) \in E$ with two directed edges (i, j) and (j,i) with capacities p_{ij} and p_{ji}.
- Relationship between the pixel graph $G=\left(V,E\right)$ and the constructed flow network $G'=\left(V',E'\right)$:

$$\text{Segmentations of } G \xleftrightarrow{\text{1-1 correspondence}} \text{Cuts of } G'$$

$$Q'(A,B) \ = \ c\left(A \cup \{s\}, B \cup \{t\}\right)$$



Pixel graph $G=(V,E)$



$$Q'(A,B) \ = \ c\left(A\cup\{s\},\ B\cup\{t\}\right)$$

Generic Push-Relabel Algorithms for Maximum Flows

Running time: $O\left(V^2 E\right)$

Preflows

- Flow net $G=(V,E)$, capacity function c, source s, sink t.
- A preflow is a function $f:V\times V\to\mathbb{R}$, satisfying
  - Capacity constraint: $\forall u,v\in V,\ f(u,v)\le c(u,v)$.
  - Skew symmetry: $\forall u,v\in V,\ f(u,v)=-f(v,u)$.
  - Relaxed flow conservation: $\forall u\in V-\{s\}$,

$$f(V,u)\ge 0$$

- The quantity $e(u)=f(V,u)$ is called the excess flow into u.
- Vertex $u\ne t$ is overflowing if $f(V,u)>0$.

- Height function: a function $h: V \to \mathbb{N}_0$, satisfying
  - $h(s) = |V|$
  - $h(t) = 0$
  - $h(u) \le h(v) + 1$ for every residual edge $(u,v) \in E_f$.
- Note: a height function is defined relative to a preflow.
- Lemma: If $h(u) > h(v) + 1$ then $(u,v) \notin E_f$.

Operation Push(u, v)

- Applicable when:

  u is overflowing, $c_f(u,v) > 0$, and $h(u) = h(v) + 1$.

- Action: push $\Delta_f(u,v) = \min\{e(u),\ c_f(u,v)\}$ units of flow from u to v.
  - $f(u,v) \leftarrow f(u,v) + \Delta_f(u,v)$.
  - $f(v,u) \leftarrow -f(u,v)$.
  - $e(u) \leftarrow e(u) - \Delta_f(u,v)$.
  - $e(v) \leftarrow e(v) + \Delta_f(u,v)$.

- The operation Push(u, v) is called a push from u to v.
- Saturating push: edge (u, v) becomes saturated (i.e., $c_f(u, v) = 0$) after the push.
- Nonsaturating push: $c_f(u, v) > 0$ after the push.
- Lemma: After a nonsaturating push from u to v, vertex u is no longer overflowing.

  Proof: After the push, either $e(u) = 0$ or $c_f(u, v) = 0$.

Operation Relabel(u)

- Applicable when:

  $u \notin \{s, t\}$ is overflowing and

  $h(u) \le h(v)$ for all edges $(u, v) \in E_f$.

- Action: increase the height of u.
  - $h(u) \leftarrow 1 + \min\left\{h(v) : (u, v) \in E_f\right\}$.

- Note: since u is overflowing, there is at least one edge $(u, v) \in E_f$, so the above min is not over an empty set.

Initialize-Preflow(G, s, t)

- Initial preflow: For all $u, v \in V$,

$$f(u,v) = \begin{cases} c(u,v) & \text{if } u = s \\ -c(u,v) & \text{if } v = s \\ 0 & \text{otherwise} \end{cases}$$

- Corresponding excess flow function:

$$e(v) = \begin{cases} c(s,v) & \text{if } (s,v) \in E \\ -\sum \{c(s,x) : (s,x) \in E\} & \text{if } v = s \\ 0 & \text{otherwise} \end{cases}$$

- Initial height function:

$$h(u) = \begin{cases} |V| & \text{if } u = s \\ 0 & \text{otherwise} \end{cases}$$

Generic-Push-Relabel Algorithm

1. Initialize-Preflow(G, s, t)   // initialize a preflow f //

2. **while** there is an applicable push or relabel operation

   do select an applicable operation and perform it

Correctness of Generic-Push-Relabel

Lemma 1. If $u \neq t$ is an overflowing vertex, then either a push or relabel operation can be applied to it.

Lemma 2. Whenever a relabel operation is applied to a vertex u, its height $h(u)$ increases by at least 1.

Lemma 3. During the execution of the algorithm, h is always a height function.

Lemma 4. During the execution of the algorithm, f is always a preflow.

Lemma 5. Let f be a preflow. If there is a height function h relative to f, then there is no path from s to t in the residual network G_f.

Proof. Otherwise, if there is a simple path p in G_f from s to t, then

$$h(s) - h(t) \le \text{length}(p) \le |V| - 1$$

contradicting the fact that $h(s) - h(t) = |V|$.

Theorem. If/when the algorithm terminates, the preflow it computes is a maximum flow.

Proof. When the algorithm terminates:

- f is a preflow (by Lemma 4).
- No vertex is overflowing (by Lemma 1).
- So, f is a flow.
- h is a height function (by Lemma 3).
- There is no augmenting path in G_f (by Lemma 5).
- So, f is a maxflow (by Max-flow Min-cut Theorem).

Analysis of Generic-Push-Relabel

Lemma 1. Let f be a preflow. Then, for any overflowing vertex u, there is a path from u to s in G_f.

Lemma 2. At any time during the execution of the algorithm, $h(u) \le 2|V|-1$ for any node $u \in V$.

Proof. When a vertex u is relabeled, it is overflowing and has a simple path to s (which is still true after the relabel). Since the path has at most $|V|-1$ edges, $h(u) \le 2|V|-1$.

Corollary (bound on relabel operations). The total number of relabel operations is at most $(2|V|-1)(|V|-2) < 2|V|^2$.

Lemma 3 (bound on saturating pushes). The total number of saturating pushes is at most $2|V||E|$.

Proof. Push(u,v) may occur only if $(u,v)\in E$ or $(v,u)\in E$. Between two consecutive saturating pushes from u to v, $h(u)$ increases by at least 2. Reasons:

- Between two saturating pushs from u to v, there must be a push from v to u.
- At the 1st Push(u,v): say $h(u)=\mathrm{a}$.
- At Push(v,u): $h(v)=h(u)+1\geq a+1$.
- At the 2nd Push(u,v): $h(u)=h(v)+1\geq a+2$.

So, for each $(u,v)\in E$ or $(v,u)\in E$, satulating Push(u,v) may occur no more than $|V|$ times.

Lemma 4 (bound on nonsaturating pushes). The number of nonsaturating pushes is less than $4|V|^2(|V|+|E|)$.

Proof. Define $\Phi = \sum_{e(u)>0} h(u)$. Initially, $\Phi = 0$.

- Relabeling a vertex u increases Φ by less than $2|V|$.
- A saturating push from u to v increases Φ by less than $2|V|$.
- Total amount of increase to Φ is less than $2|V| \cdot \left(2|V|^2 + 2|V||E|\right) = 4|V|^2(|V|+|E|)$.
- A nonsaturating push from u to v deccreases Φ by at least 1.
- Thus, the total number of nonsaturating pushes is less than $4|V|^2(|V|+|E|)$.

Lemma 5. Each relabel can be done in $O(V)$ time and each push can be done in $O(1)$ time.

Theorem. The running time of the generic push-relabel algorithm is $O(V^2E)$.

Proof.

- Total time for relabels: $O(V^3)$.
- Total time for satulating pushes: $O(VE)$.
- Total time for nonsatulating pushes: $O(V^2E)$.

The Relabel-to-Front Algorithm

Running time: $O\left(V^3\right)$

Admissible edges and networks

- An edge (u,v) is admissible if

  $$c_f(u,v) > 0 \text{ and } h(u) = h(v) + 1.$$

- Admissible network: $G_{f,h} = \left(V, E_{f,h}\right)$, where $E_{f,h}$ is the set of admissible edges. It is a subgraph of G_f.

Lemma 1. The admissible network $G_{f,h}$ is acyclic.

Proof. The height function $h(\cdot)$ is decreasing along any path in $G_{f,h}$.

When is Push(u,v) applicable? How does it affect $G_{f,h}$?

Lemma 2. If a vertex u is overflowing and edge (u,v) is admissible, then Push(u,v) is applicable. The operation does not create any new admissible edges, but it may cause (u,v) to become inadmissible.

Proof. The Push(u,v) operation reduces $c_f(u,v)$ and increases $c_f(v,u)$. If $c_f(u,v)$ becomes 0, (u,v) becomes inadmissible. Since $h(u) = h(v) + 1$, (v,u) cannot become admissible.

When is Relabel(u) applicable? How does it affect $G_{f,h}$?

Lemma 3. If a vertex $u \notin \{s,t\}$ is overflowing and there are no admissible edges leaving u, then Relabel(u) is applicable. **After the relabel operation, there is at least one admissible edge leaving u, but there are no admissible edges entering u.**

Proof. Only the last claim needs a proof.

If, after the relabel, (v,u) is an admissible edge entering u, then $h(v) = h(u)+1$. Before the relabel of u, $h(v) > h(u)+1$ and thus $(v,u) \notin E_f$. $\Rightarrow\Leftarrow$

Neighbor lists

- Same as the adjacency lists of the flow network $G=(V,E)$, except that the list of u contains v iff $(u,v)\in E$ or $(v,u)\in E$.
- $N(u)$: the neighbor list of u. It contains those vertices v for which there may be a residual edge (u,v).
- $head\left(N(u)\right)$: pointing to the first element in $N(u)$.
- $current(u)$: pointing to the vertex currently under consideration in $N(u)$. Initially, $current(u)\leftarrow head\left(N(u)\right)$.
- $next\text{-}neighbor(\bullet)$:

Discharging an overflowing vertex

- Discharge(u): push all excess flow of u thru admissible edges leaving u, relabeling u as necessary.

- Procedure Discharge(u) //after Discharge(u), $e(u)=0$//

  while $e(u)>0$ **do**

  $v \leftarrow current(u)$

  if $v=\text{NIL}$ **then**

  Relabel(u)

  $current(u) \leftarrow head\left(N(u)\right)$

  elseif (u,v) is admissible **then**

  Push(u,v)

  else $current(u) \leftarrow next\text{-}neighbor(v)$

Algorithm Relabel-to-Front(G, s, t)

Initialize-Preflow(G, s, t)
$L \leftarrow V[G] - \{s, t\}$ in any order
Initialize $current(u)$ for each $u \in V[G] - \{s, t\}$
$u \leftarrow head(L)$
while $u \neq$ NIL **do**
    Discharge(u)
    if u has been relabeled during Discharge(u)
        then move u to the front of L
    $u \leftarrow next\text{-}neighbor(u)$

Lemma 4. Relabel-to-Front performs push and relabel operations only when they are applicable.

Lemma 5. At each test in line 5 of Relabel-to-Front, L is a topological sort of $G_{f,h} - \{s, t\}$ and no vertex before u in the list has excess flow.

Corollary. When Relabel-to-Front terminates, there are no applicable push or relabel operations (since by Lemma 5 there is no overflowing node).

Theorem. Relabel-to-Front is an implementation of the generic push and relable algorithm.

Proof of Lemma 5 (part 1). By induction, we show L is a topological sort of $G_{f,h} - \{s, t\}$.

- For iteration 1, it is true, since initially $E_{f,h} = \phi$.
- Assume that L is in topological order at the beginning of an iteration.
- During the iteration, we perform pushes and relables.
  - Pushes do not create any admissible edges (Lemma 2).
  - By Lemma 3, Relabel(u) may create admissible edges leaving u, but after the relabel there will be no admissible edge entering u. By moving u to the front of L, L remains in topological order.

Proof of Lemma 5 (part 2). By induction, we show that vertices before u have no excess flow.

- Initially, it is true since u is at the front of L.
- Assume the property holds at the beginning of an iteration.
- Let u' be the vertex that will be the u in the next iteration.
- We will show that no vertex before u' has excess flow.
- If u is moved to front, it has no excess flow (since it has been discharged), and it is the only vertex before u'.
- If u is not moved to front, vertices before u received no additional flow and thus still have no excess, and u itself now has no excess.

Theorem. The running time of Relabel-to-Front is $O\left(V^3\right)$.

Proof.

- The running time $=$

$$O\left(\begin{array}{l}\text{the total number of iterations (discharges)} \\ +\text{ the time spent on executing the discharges}\end{array}\right)$$

- We first determine the number of discharges:
  - There are at most $O(V^2)$ relabels.
  - Preceding each relabel there may be $O(V)$ calls to Discharge. Similarly, $O(V)$ discharges after the last relabel.
  - Thus, the total number of calls to Discharge is $O(V^3)$.

- Now we determine the total time spent within Discharge.
  - Total time for moving the pointer *current*: $O(V^3)$.
    - Preceding each Relabel(u), it takes $O(V)$ time to move $current(u)$.
    - There are at most $O(V^2)$ relabels.
  - Total time for relabels: $O(V^3)$.
  - Total time for satulating pushes: $O(VE) \subseteq O(V^3)$.
  - Total time for nonsatulating pushes: $O(V^3)$.
    - Each discharge has at most 1 nonsatulating push.