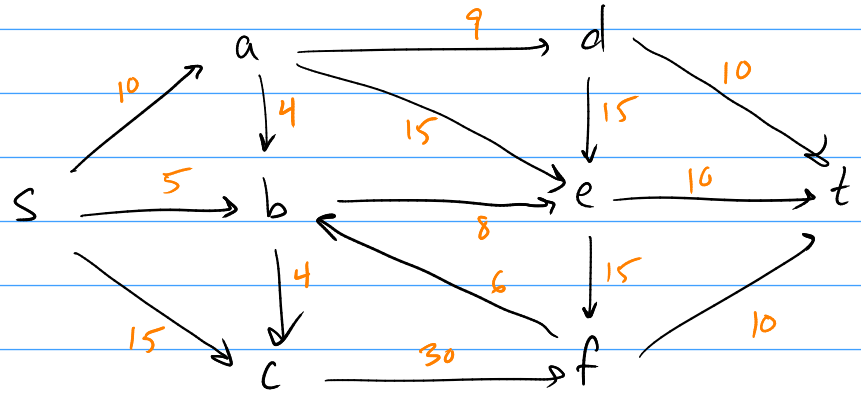



GreedyFlow(G, s, t):

repeat: ("augmenting path")

P = find a path from s→t with positive capacity remaining.
(is shortest? max capacity? min capacity?)

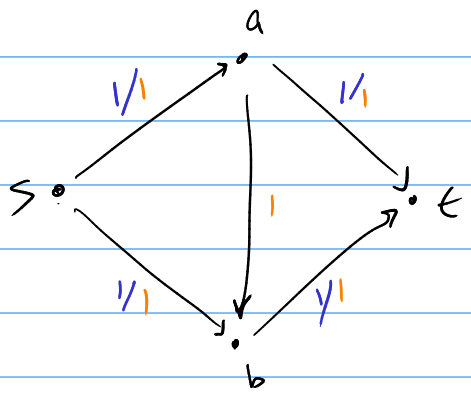
α ← bottleneck of P, ie. min remaining capacity of any edge in P.

for each edge e∈P:

f(e) ← f(e) + α.

while any augmenting path remains.

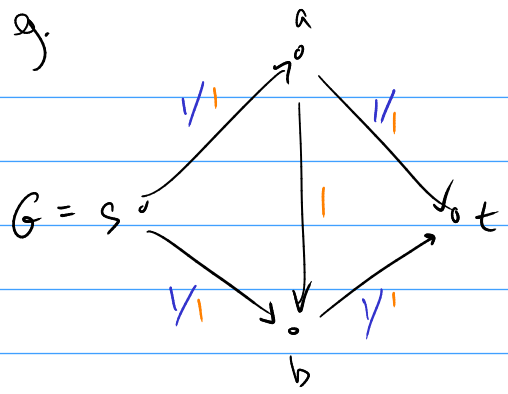
Does not work!

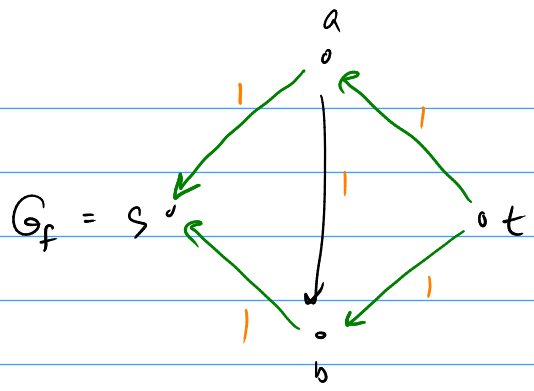



Defn Given a flow network G with flow f, define the residual network $G_f$ as follows:

- $G_f$ has the same vertices as G.
- For each edge e with c(e) > f(e), add a corresponding edge to $G_f$ with weight c(e) − f(e).
- For each edge e with f(e) > 0, add a corresponding backwards edge to $G_f$ with weight f(e).

9.




Ford-Fulkerson $(G, s, t)$:

Copy $G$ into $G_f$

while there exists a path $P$ from $s$ to $t$ in $G_f$:

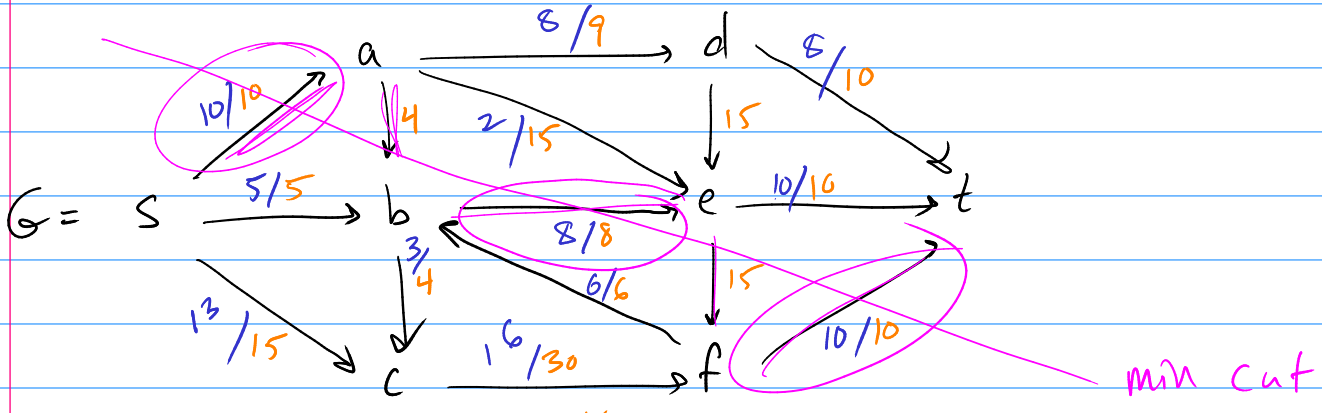
$\alpha \leftarrow$ min weight along $P$

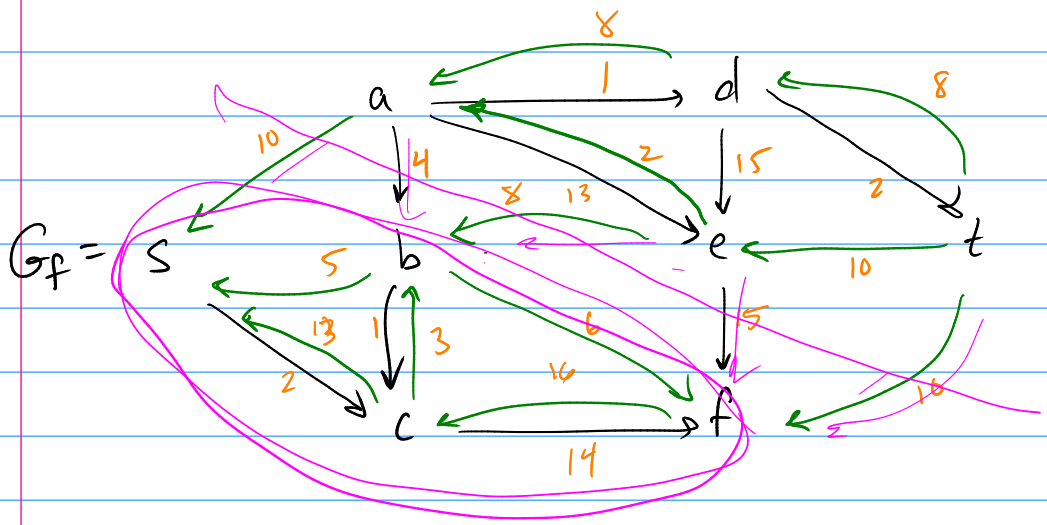
for each edge $e \in P$:

if $e$ is a forwards edge, $f(e) \leftarrow f(e) + \alpha$

if $e$ is backwards edge, $f(e) \leftarrow f(e) - \alpha$

Update $G_f$



Observation. Augmenting along a path in the residual network always preserves the flow properties.

↳ zero flow is valid, so by induction the F-F algorithm will always produce a valid flow.

Observation. Augmenting along an s-t path in the residual network will always increase the value of a flow.

Reason: s only has outgoing edges, so any s-t path must start w/ forwards edge.

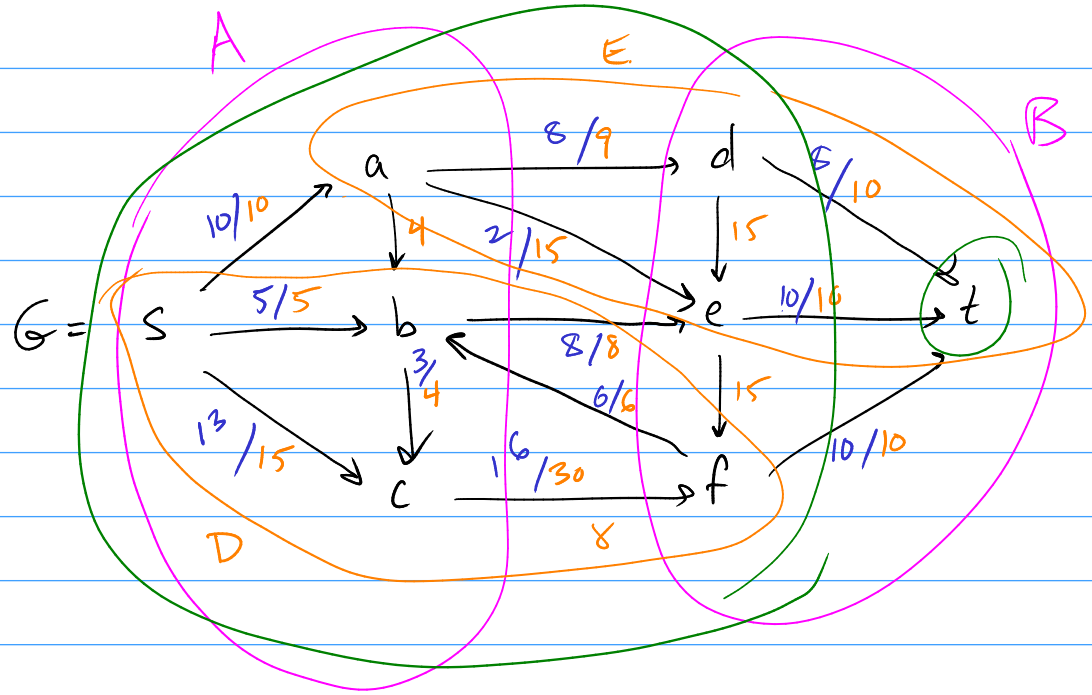
↳ Therefore the F-F algorithm will terminate.

Each loop must increase flow by at least 1, so max # of iterations is capacity coming out of s.

Defn An s-t cut is a partition of V into two sets $(A, B)$ where $s \in A$ and $t \in B$.

Defn The capacity of an s-t cut is the total capacity of all edges from A to B, i.e.

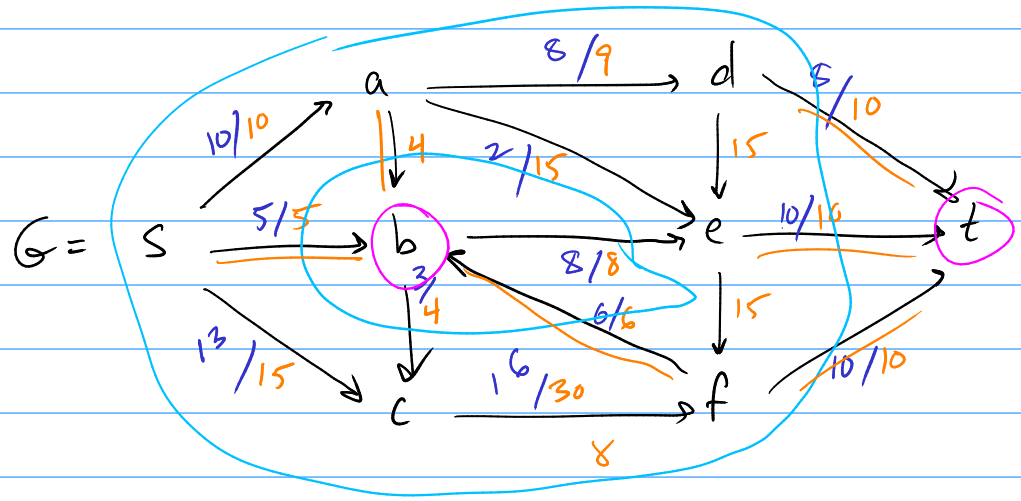
$$c(A,B) = \sum_{e : A \to B} c(e).$$




$c(A, B) = 9 + 15 + 8 + 30 = 24 + 38 = 62.$

$c(D, E) = 10 + 8 + 16 = 28.$

$c(A, B) = 30$



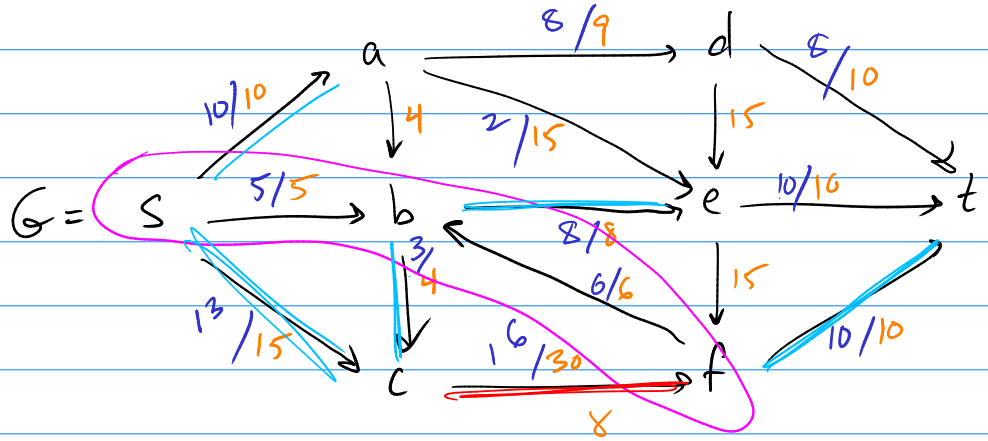

$c(A, B) = 30 + 5 + 4 + 6 = 45.$

**Def'n** The min cut is an s-t cut with minimum possible capacity.

**Defn** The **net flow** of an $s$-$t$ cut $(A,B)$ with respect to a flow $f$ is

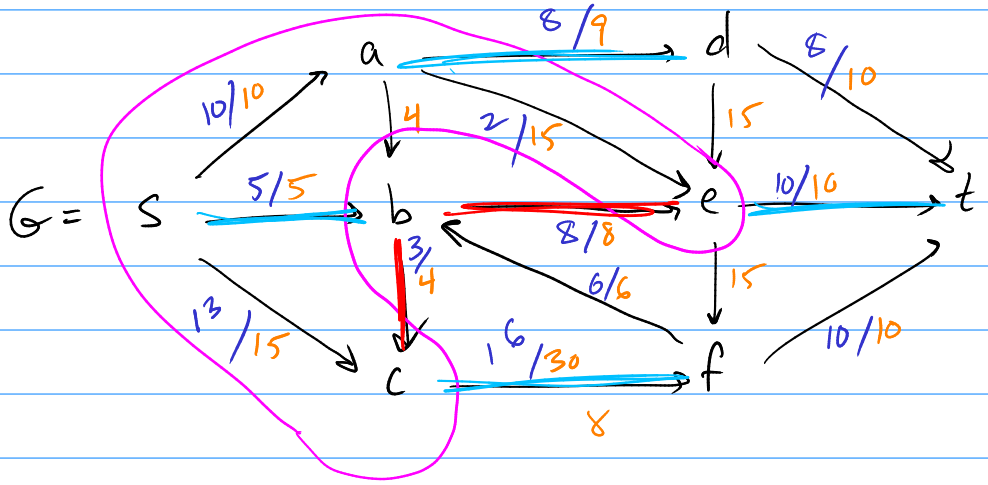
$$\sum_{e:A\to B} f(e) - \sum_{e:B\to A} f(e).$$

i.e. "net exports" from $A$ to $B$.




net flow = $10 + 13 + 3 + 8 + 10 - 16$

$= 44 - 16 = 28.$




net flow = $5 + 8 + 10 + 16 - 8 - 3$

$= 13 + 26 - 11 = 26 + 2 = 28.$

**Lemma.** (Flow value lemma) Let $f$ be any flow and $(A,B)$ any $s$-$t$ cut. Then the net flow of $(A,B)$ with respect to $f$ equals the value of $f$, $v(f)$.

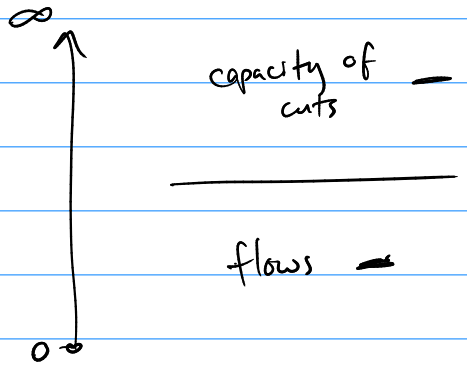
**Proof.** $v(f) = f_{out}(s)$ ← amt of flow going out of $s$.

$$= \sum_{v \in A} \left( f_{out}(v) - f_{in}(v) \right).$$ ← $f_{out} - f_{in} = 0$ for all vertices other than $s,t$.

$$= \text{net flow of } (A,B).$$

∎

**Lemma** (Bottleneck lemma). Let $f$ be any flow and $(A,B)$ any $s$-$t$ cut. Then $v(f) \leq c(A,B)$.




Corollary: if $v(f) = c(A,B)$ for some flow $f$ and $s$-$t$ cut $(A,B)$, then $f$ is a max flow and $(A,B)$ is a min cut.

**Theorem.** (Max flow / min cut). Let $f$ be any flow on a network $G$. The following are equivalent:

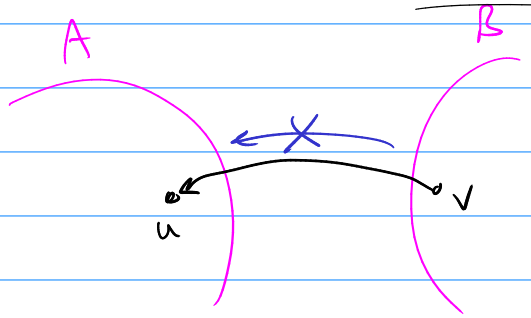
1. $v(f) = c(A,B)$ for some $s$-$t$ cut $(A,B)$.
2. $f$ is a max flow. (and $(A,B)$ is a min cut).
3. There are no $s$-$t$ paths in the residual network $G_f$.

Proof. We will show (1) ⇒ (2) ⇒ (3) ⇒ (1).

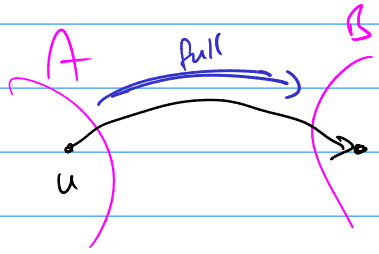
- (1) ⇒ (2): This is the corollary from above: since all flows are ≤ all cuts, if $v(f) = c(A,B)$ then $f$ is a max flow (and $(A,B)$ is a min cut).
- (2) ⇒ (3). By contrapositive: suppose there is a path from $s$ to $t$ in $G_f$. Then we can augment along the path, increasing the flow, hence $f$ was not a max flow.
- (3) ⇒ (1). Suppose there is no path $s \to t$ in $G_f$. Let $A$ = set of vertices reachable from $s$, $B$ = all others. Note $t \in B$ since we assumed $t$ is not reachable from $s$. So $(A,B)$ is an $s$-$t$ cut. $(V - A)$.

By the Flow Value Lemma,

$$v(f) = \text{net flow of } (A,B) = \sum_{e: A \to B} f(e) - \sum_{e: B \to A} f(e).$$




For edges $e: B \to A$, they cannot have any flow: if they did, there would be a backwards edge in $G_f$, but that contradicts our definition of $B$ as vertices unreachable from $s$.




Edges $e: A \to B$ must be full, since otherwise there would be a forwards edge in $G_f$.

So

$$v(f) = \text{net flow of } (A,B) = \sum_{e:A\to B} f(e) - \sum_{e:B\to A} f(e)$$

$$= \sum_{e:A\to B} c(e) - 0$$

$$= c(A,B).$$

Corollary. Ford-Fulkerson alg. correctly finds max flows (and min cuts).

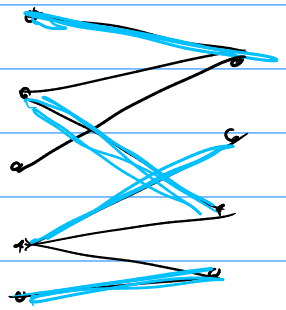
Proof. We know the algorithm terminates, and it does so when there are no $s-t$ paths in $G_f$, hence by the above thm it has found a max flow.

---

Example Applications.

① Maximum bipartite matching.

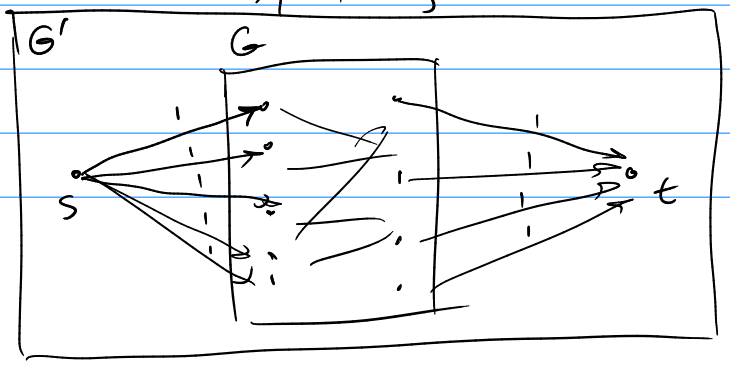
Let $G$ be a bipartite graph with $V = (L, R)$.



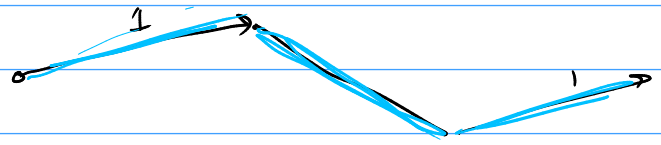
A matching on $G$ is a subset of edges such that no two edges share an endpoint.

A max matching is a matching w/ biggest possible # of edges.

We can solve this by turning it into max flow problem!



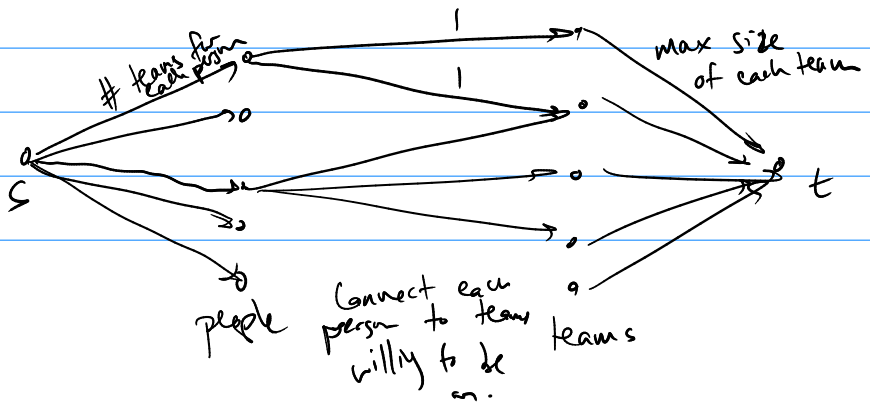
Claim: given a valid matching of size $k$ on $G$, we can make it into a valid flow on $G'$ of value $k$, and vice versa.




Valid matching $\Rightarrow$ valid flow since if no 2 edges share an endpoint, we can just send 1 unit of flow along each disjoint path

Valid flow $\rightarrow$ valid matching since at most 1 unit of flow can reach a vertex in $L$ or leave a vertex in $R$, so we can't possibly have flow along 2 edges sharing an endpoint.

Hence max flow $\Rightarrow$ max matching.

More examples: airline scheduling, baseball elimination, image segmentation.

Assigning people to teams:
- Each person has a set of teams they are willing to be on
- Each team has a max size
- Assign as many people as possible.
- Each person has a max # of teams they are willing to be on.