CS3000: Algorithms & Data
Jonathan Ullman

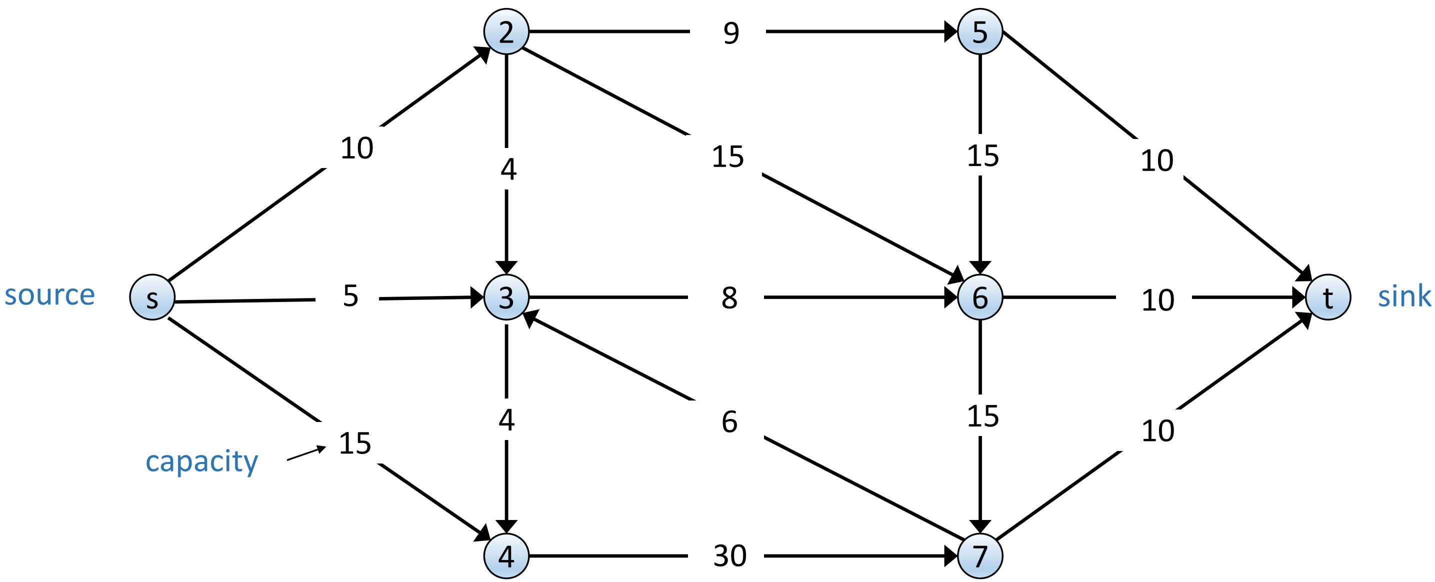
Lecture 18:

- Network Flow: choosing good paths

Nov 9, 2018

Flow Networks

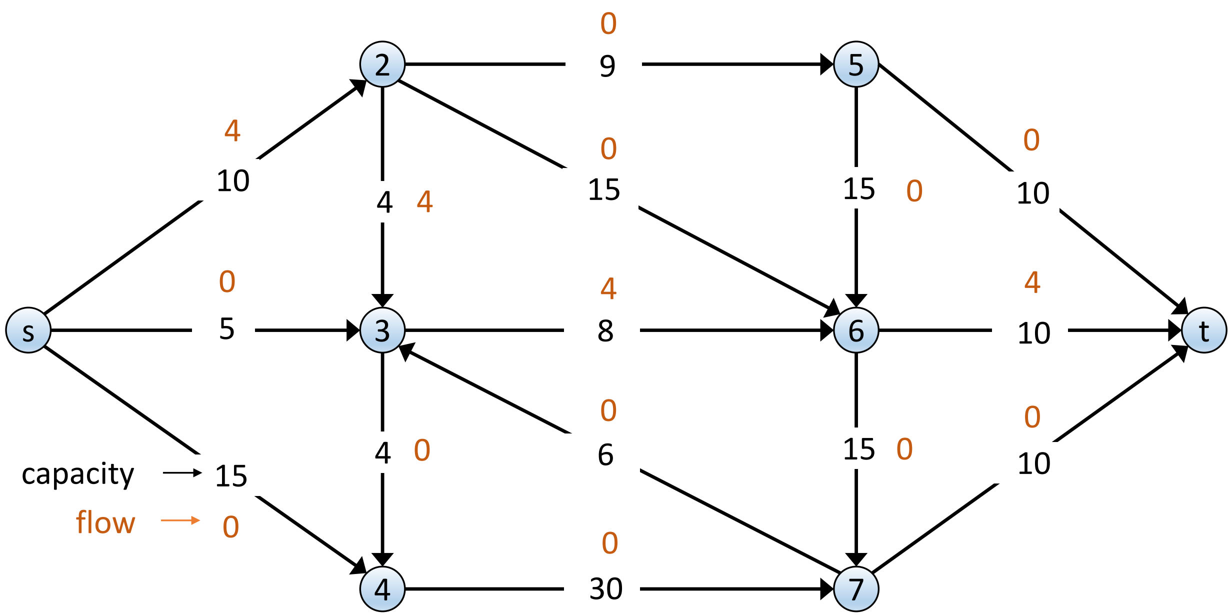
- Directed graph $G = (V, E)$
- Two special nodes: source s and sink t
- Edge capacities $c(e)$



Flows

- An s-t flow is a function $f(e)$ such that
  - For every $e \in E$, $0 \le f(e) \le c(e)$ (capacity)
  - For every , $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)

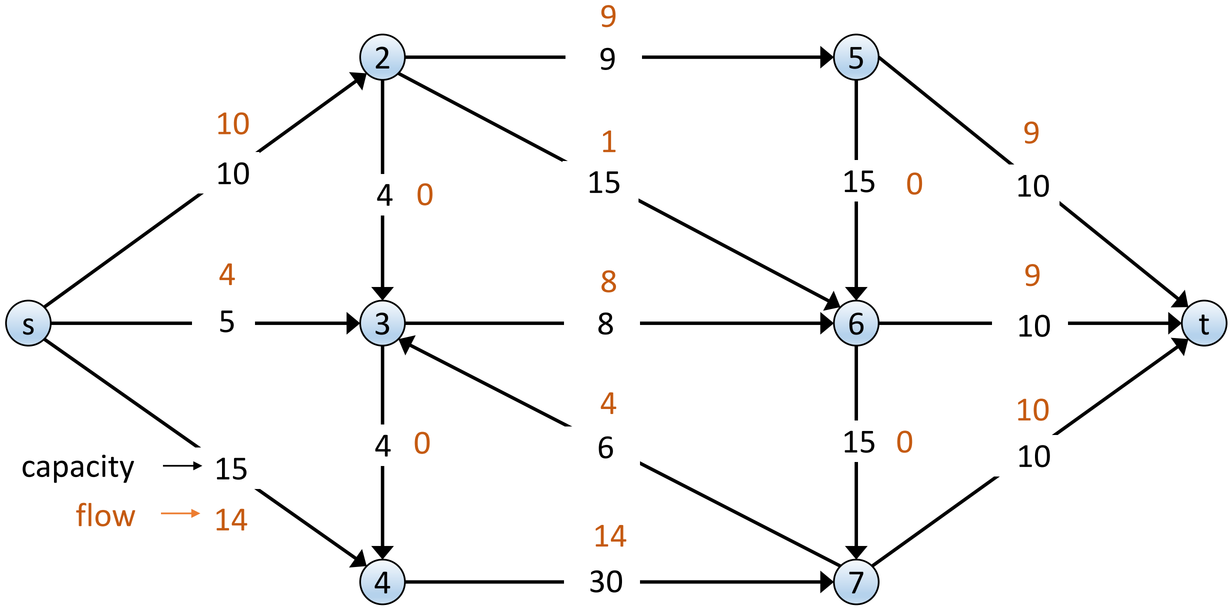
    $v \in V$ $v \neq s, t$
- The value of a flow is $val(f) = \sum_{e \text{ out of } s} f(e)$



Maximum Flow Problem

- Given G = (V,E,s,t,{c(e)}), find an s-t flow of maximum value

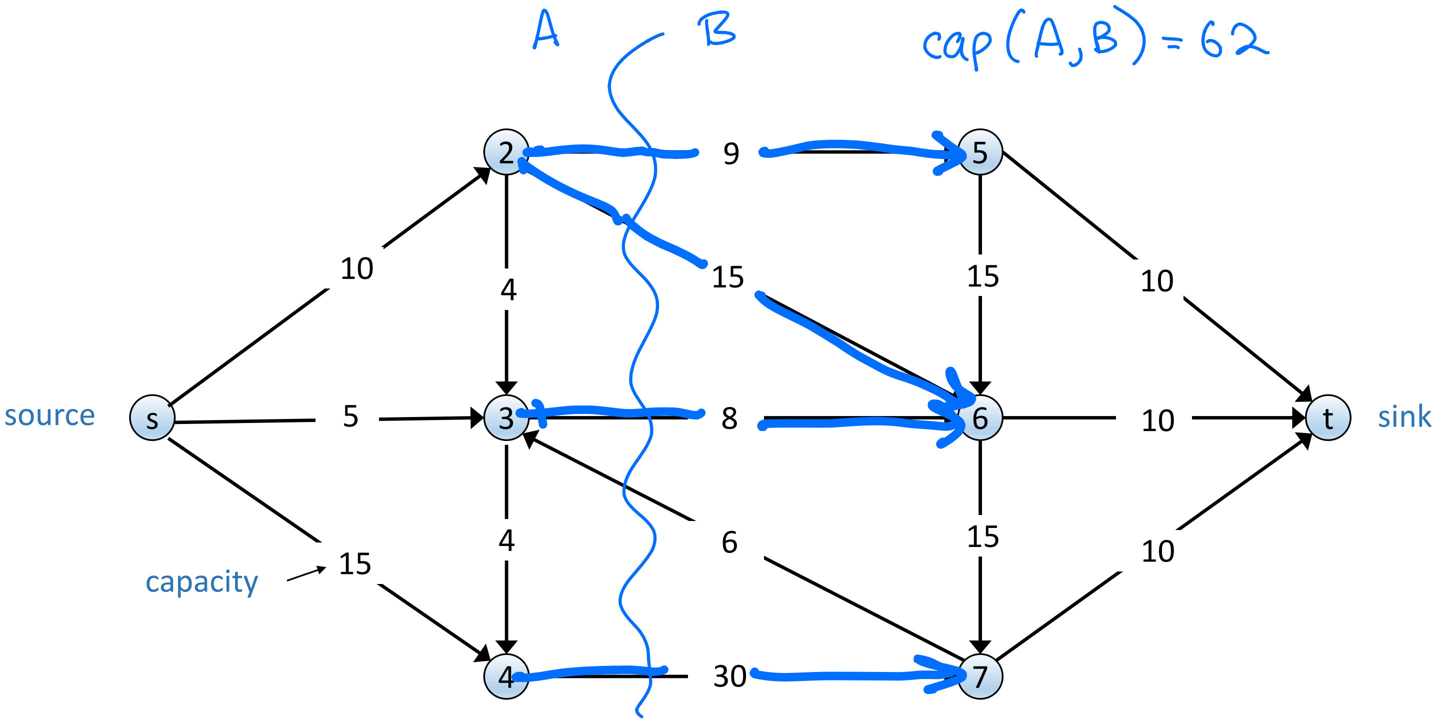
val(f) = 28



Cuts

- An s-t cut is a partition (A, B) of V with $s \in A$ and $t \in B$

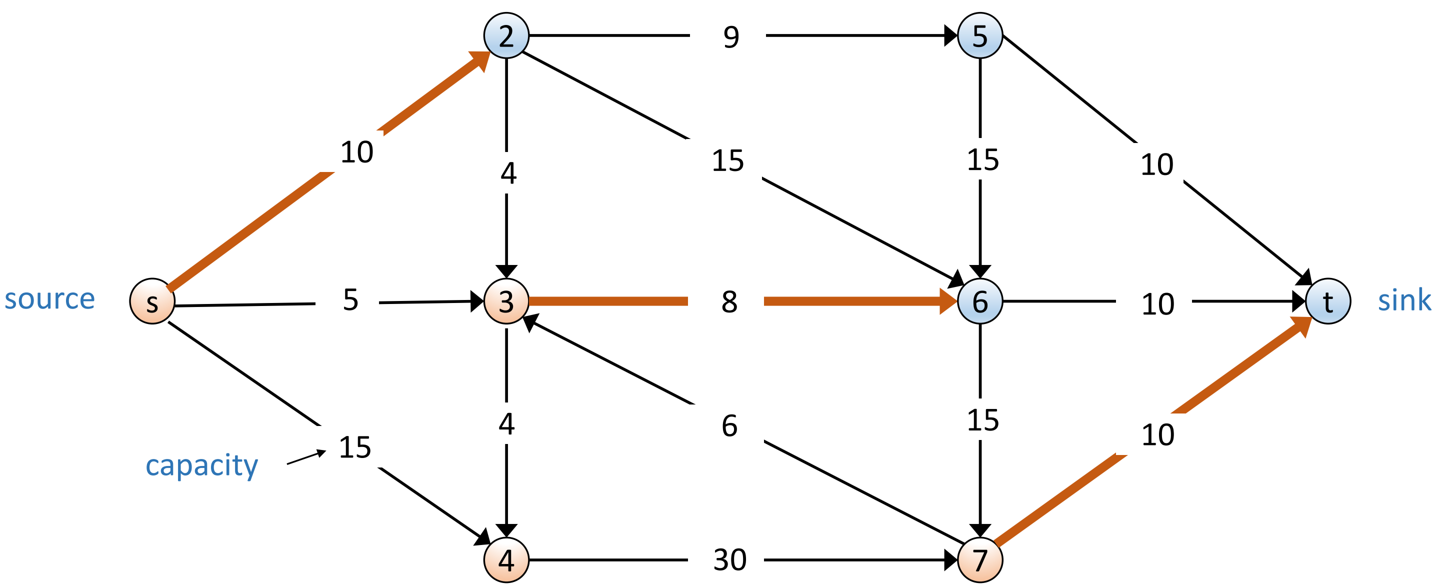
- The capacity of a cut (A,B) is $cap(A, B) = \sum_{e \text{ out of } A} c(e)$



Minimum Cut problem

- Given G = (V,E,s,t,{c(e)}), find an s-t cut of minimum capacity

cap(A, B) = 28

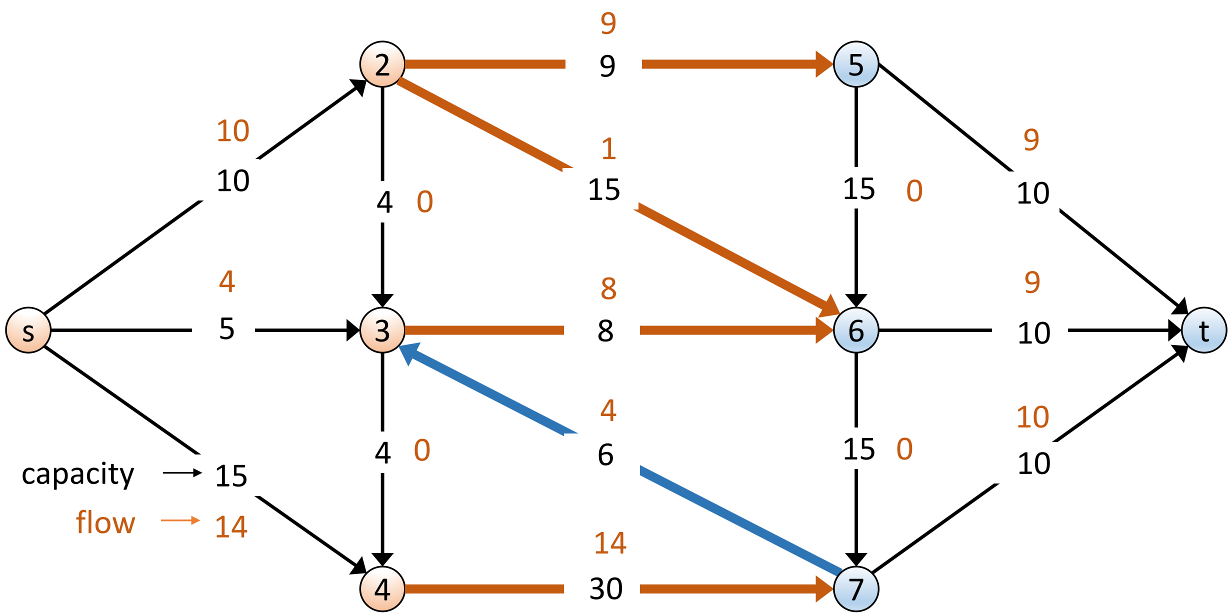


Flows vs. Cuts

$\Rightarrow val(f^*) \le \min cap(A, B)$
(weak duality)

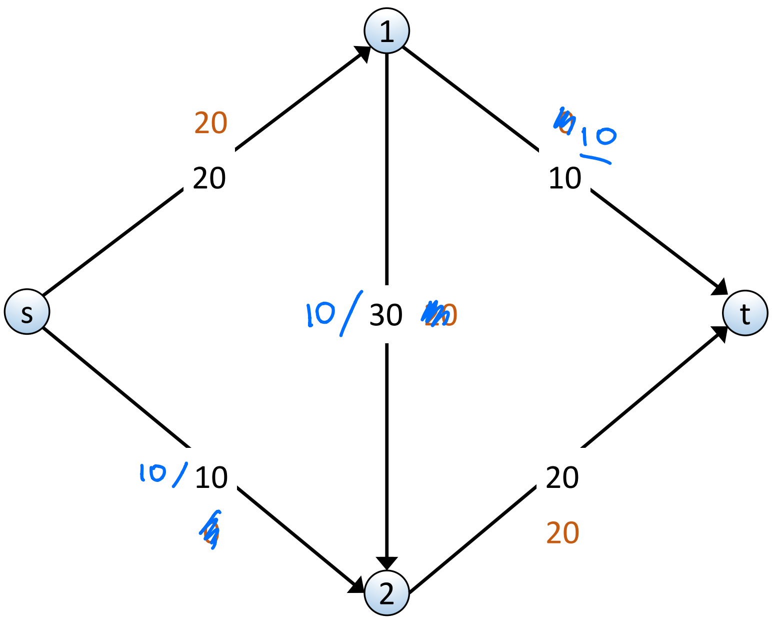
- **Fact:** If f is any s-t flow and (A, B) is any s-t cut, then the net flow across (A, B) is equal to the amount leaving s

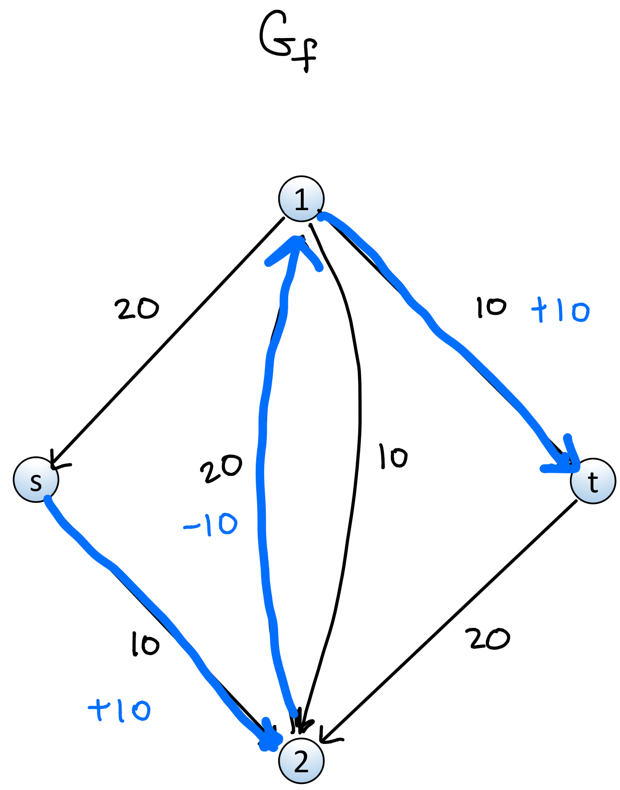
$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = val(f)$$



Ford-Fulkerson Algorithm

- Start with $f(e) = 0$ for all edges $e \in E$
- Find an **augmenting path** P in the **residual graph**
- Repeat until you get stuck





original graph

$$u \xrightarrow{f(e)\,/\,c(e)} v$$

residual graph

$$\begin{tikzcd} u \arrow[r, bend left, "c(e)-f(e)"] & v \arrow[l, bend left, "f(e)"] \end{tikzcd}$$

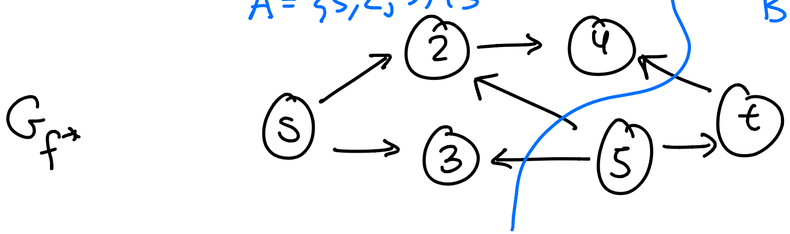
(remove 0 capacity edges)

Summary

- The **Ford-Fulkerson Algorithm** solves maximum s-t flow
  - Running time is $O(m)$ per augmentation step
  - $O(val(f^*))$ augmentinations in any graph with integer capacities

- **MaxFlow-MinCut Theorem:** The value of the max s-t flow equals the capacity of the min s-t cut
  - If f^* is a max flow, the nodes reachable from s in G_{f^*} are a min cut
  - Given a max flow, can find a min cut in time $O(n + m)$ via BFS

$A^* = \{s, 2, 3, 4\}$ $B^* = \{5, t\}$



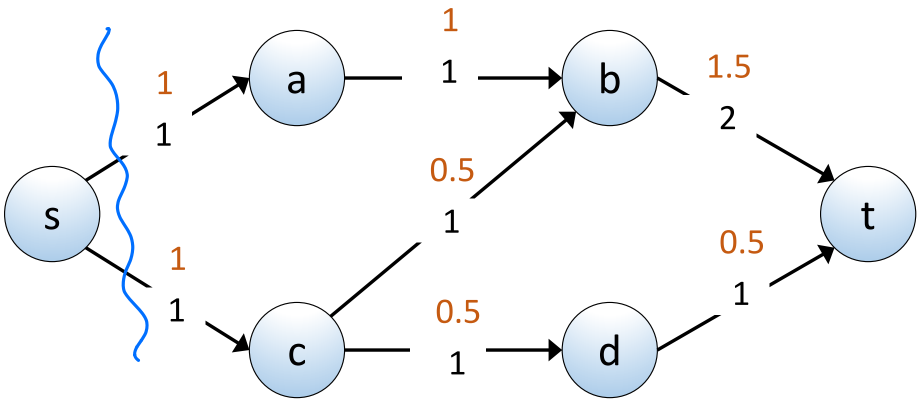
If f^* is a max flow, A^*, B^* is a minimum cut.

- For any f and any G, the max flow in G can be expressed as $\boxed{f + (\text{max flow in } G_f)}$

Ask the Audience

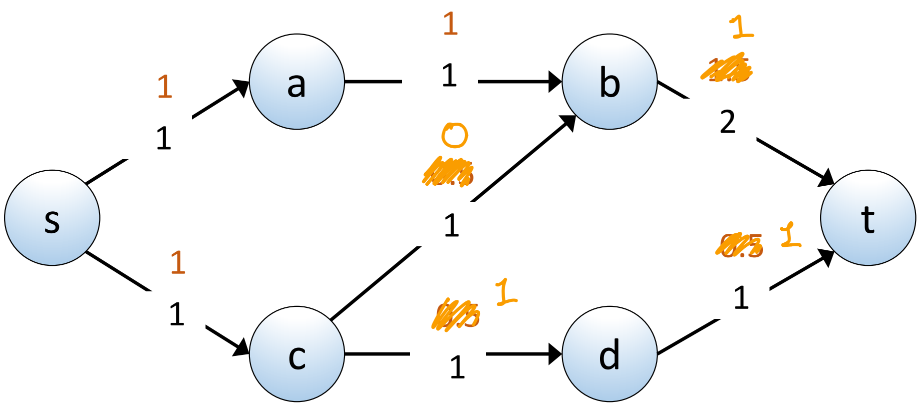
- Is this a maximum flow?

val=2
cap=2



Ask the Audience

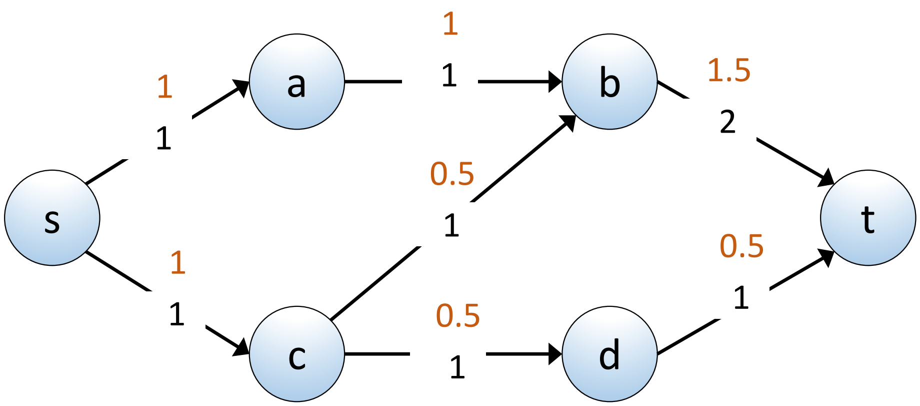
- Is this a maximum flow?



- Is there an integer maximum flow?

Ask the Audience

- Is this a maximum flow?



- Is there an integer maximum flow?
- Does every graph with integer capacities have an integer maximum flow?
- Yes. FF will find an integer flow.

Summary

- The **Ford-Fulkerson Algorithm** solves maximum s-t flow
  - Running time is $O(m)$ per augmentation step
  - $O(val(f^*))$ augmentinations in any graph with integer capacities

- **MaxFlow-MinCut Theorem:** The value of the max s-t flow equals the capacity of the min s-t cut
  - If f^* is a max flow, the nodes reachable from s in G_{f^*} are a min cut
  - Given a max flow, can find a min cut in time $O(n + m)$ via BFS

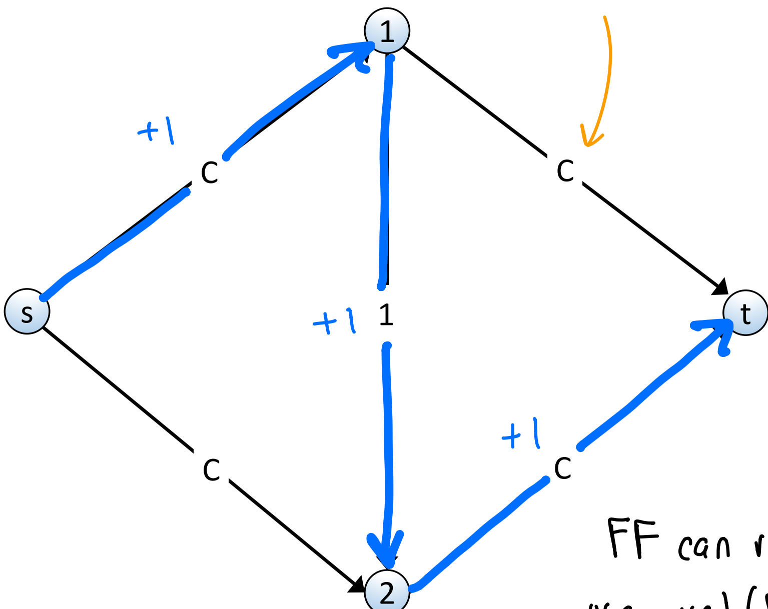
- Every graph with integer capacities has an integer max flow
  - And Ford-Fulkerson finds an integer max flow

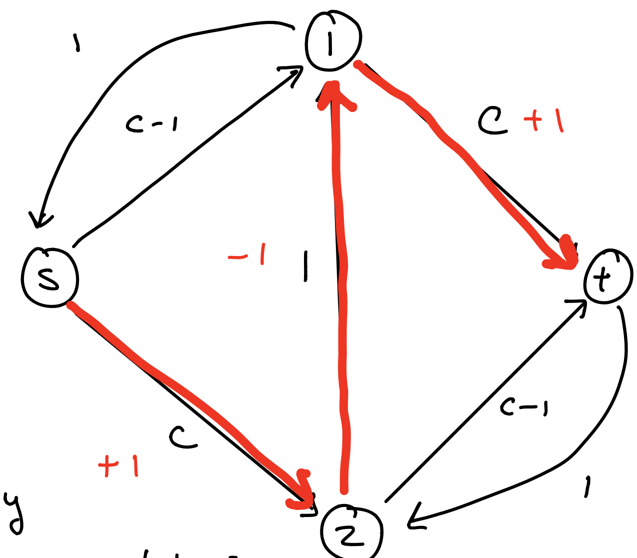
Ford-Fulkerson Algorithm

- Start with $f(e) = 0$ for all edges $e \in E$
- Find an **augmenting path** P in the **residual graph**
- Repeat until you get stuck

val$(f^*) = 2c$

c is some really big number





FF can really use val(f^*) augmentations

Choosing Good Augmenting Paths

- **Last time:** arbitrary augmenting paths
  - If FF terminates, it outputs a maximum flow
  - Might not terminate, or might require many augmentations
- **Today:** clever augmenting paths
  - Maximum-capacity augmenting path (“fattest augmenting path”)
  - Shortest augmenting paths (“shortest augmenting path”)

Fattest Augmenting Path

Fattest Augmenting Path

- Maximum-capacity augmenting path

$$\max_{\substack{s\text{-}t \text{ paths } P \\ \text{in } G_f}} \underbrace{\left(\min_{e \in P} c(e) \right)}_{\text{"bottleneck capacity"}}$$

- Can find the fattest augmenting path in time $O(m \log n)$ in several different ways
  - Variants of Prim's or Kruskal's MST algorithms
  - BFS + binary search

Only slightly slower than finding any path.

Fattest Augmenting Path

$v^* =$ value of max flow f^*

Arbitrary Paths

- **Assume integer capacities**
- Value of maxflow: v^*
- Value of aug path: ≥ 1
- Flow remaining in G_f: $\leq v^* - 1$
- # of aug paths: $\leq v^*$

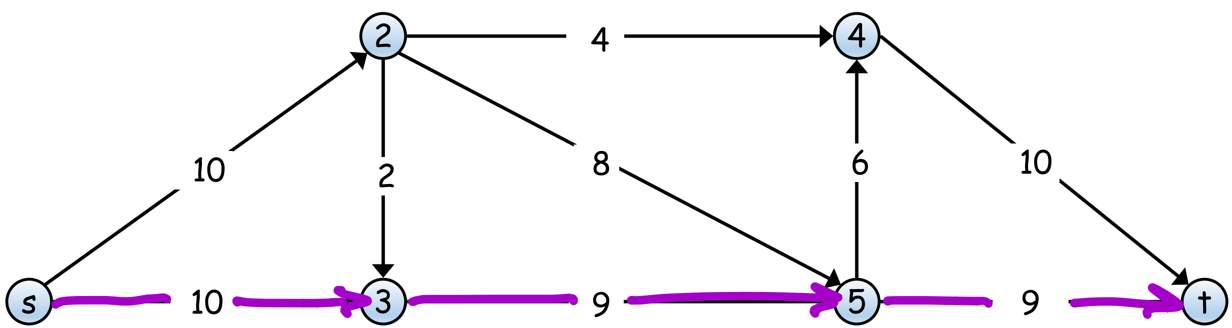
Maximum-Capacity Path

- **Assume integer capacities**
- Value of maxflow: v^*
- Value of aug path:
- Flow remaining in G_f:
- # of aug paths:

Fattest Augmenting Path

- f^* is a maximum flow with value $v^* = val(f^*)$
- P is a fattest augmenting s-t path with capacity b
- **Key Claim:** $b \geq \frac{v^*}{m}$

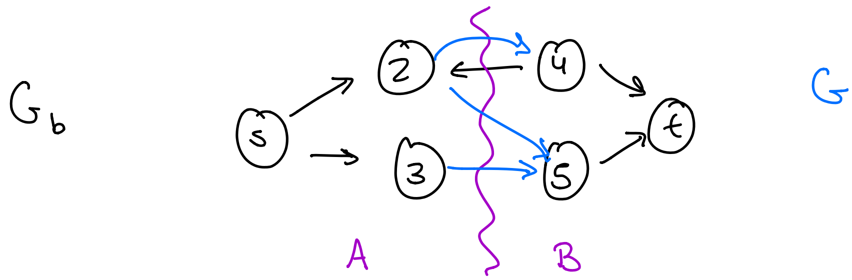
val(f*) = 19   m = 9



b = 9

Fattest Augmenting Path

- f^* is a maximum flow with value $v^* = val(f^*)$
- P is a fattest augmenting s-t path with capacity b
- **Key Claim:** $b \geq \frac{v^*}{m}$
- **Proof:**
  - Let G_b be the graph with nodes V and edges whose capacity is $> b$ in E.
  - G_b has no path from s to t
  - Let $A = \{ v$ reachable from s in $G_b \}$



- $\text{cap}(A,B) = \sum_{e \text{ out of } A} c(e) \le b \cdot (\#\text{ of edges from } A \text{ to } B)$

$$\le b \cdot m$$

- $v^* \le \text{cap}(A,B) \le b \cdot m \iff b \geq \frac{v^*}{m}$

Fattest Augmenting Path

Arbitrary Paths

- **Assume integer capacities**

- Value of maxflow: v^*
- Value of aug path: ≥ 1
- Flow remaining in G_f: $\leq v^* - 1$
- # of aug paths: $\leq v^*$

Maximum-Capacity Path

- **Assume integer capacities**

- Value of maxflow: v^*
- Value of aug path: $\geq \frac{v^*}{m}$
- Flow remaining in G_f: $\leq \left(1 - \frac{1}{m}\right) v^*$
- # of aug paths: $\leq m \cdot \ln(v^*) + 1$

- After k augmentations, the remaining flow is $\leq \left(1-\frac{1}{m}\right)^k v^*$
- If there are $k+1$ augmentations, then

$$\left(1-\frac{1}{m}\right)^k v^* \geq 1$$

$$\left[\left(1-\frac{1}{m}\right)^m\right]^{\frac{k}{m}} \cdot v^* \geq 1$$

$$\left(\frac{1}{e}\right)^{\frac{k}{m}} \cdot v^* \geq 1$$

$$\frac{k}{m} \cdot (-1) + \ln(v^*) \geq 0$$

$$k \leq m \cdot \ln(v^*)$$

$\Rightarrow$ At most $m \cdot \ln(v^*) + 1$ augmentations

Choosing Good Paths

$\leq \min\{m \ln(v^*), v^*\}$

- **Last time:** arbitrary augmenting paths
  - If FF terminates, it outputs a maximum flow

- **Today:** clever augmenting paths
  - Maximum-capacity augmenting path ("fattest augmenting path")
    - $\leq m \ln v^*$ augmenting paths (assuming integer capacities)
    - $O(m^2 \ln n \ln v^*)$ total running time
    - See KT for a slightly faster variant ("fat-ish augmenting path"?)

  - Shortest augmenting paths ("shortest augmenting path")

Shortest Augmenting Path

Shortest Augmenting Path

- Find the augmenting path with the fewest hops
  - Can find shortest augmenting path in $O(m)$ time using BFS

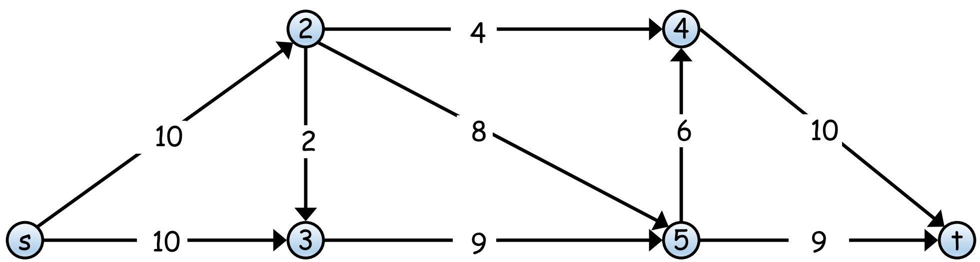
- **Theorem:** for any capacities $\frac{nm}{2}$ augmentations suffice
  - Overall running time $O(m^2 n)$
  - Works for any capacities!

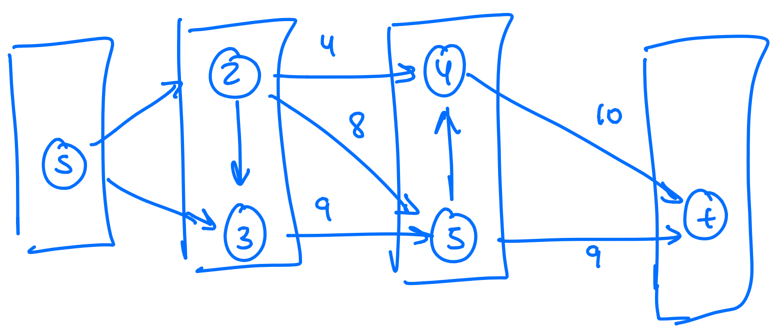
- **Warning:** proof is challenging (you will not be tested on it)

Shortest Augmenting Path

- Let f_i be the flow after the i-th augmenting path
- Let $G_i = G_{f_i}$ be the i-th residual graph
- Let $L_i(v)$ be the distance from s to v in G_i
  - Recall that the shortest path in G_i moves layer-by-layer

$G = G_0$





Shortest Augmenting Path

- Every augmentation causes at least one edge to disappear from the residual graph, may also cause an edge to appear

- **Key Property:** each edge disappears at most $\frac{n}{2}$ times
  - Means that there are at most $\frac{mn}{2}$ augmentaitons

Shortest Augmenting Path

$L_i(v)$ = # of hops from s to v in G_i

- **Claim 1:** for every $v \in V$ and every i, $L_{i+1}(v) \geq L_i(v)$
  - Obvious for $v = s$ because $L_i(s) = 0$
  - Suppose for the sake of contradiction that $L_{i+1}(v) < L_i(v)$
    - Let v be the smallest such node
  - Let $s \leadsto u \rightarrow v$ be a shortest path in G_{i+1}
    - By optimality of the path, $L_{i+1}(v) = L_{i+1}(u) + 1$
    - By assumption, $L_{i+1}(u) \geq L_i(u)$
  - Two Cases:
    - $(u, v) \in G_i$, so $L_i(v) \leq L_i(u) + 1$
    - $(u, v) \notin G_i$, so (v, u) was in the i-th path, so $L_i(v) = L_i(u) - 1$

Shortest Augmenting Path

- **Claim 2:** If an edge $u \to v$ disappears from G_i and reappears in G_{j+1} then $L_j(u) \geq L_i(u) + 2$
  - $u \to v$ is on the i-th augmenting path, $L_i(v) = L_i(u) + 1$
  - $v \to u$ is on the j-th augmenting path, $L_j(u) = L_j(v) + 1$
  - By Claim 1: $L_j(v) \geq L_i(v)$

- **Claim 3:** An edge (u, v) cannot reappear more than $\frac{n}{2}$ times
  - $0 \leq L_i(u) \leq n$
  - By Claim 2: length increases by 2 for each reappearance

Choosing Good Paths

- **Last time:** arbitrary augmenting paths
  - If FF terminates, it outputs a maximum flow

- **Today:** clever augmenting paths
  - Maximum-capacity augmenting path ("fattest augmenting path")
    - $\leq m \ln v^*$ augmenting paths (assuming integer capacities)
    - $O(m^2 \ln n \ln v^*)$ total running time
    - See KT for a slightly faster variant ("fat-ish augmenting path"?)

  - Shortest augmenting paths ("shortest augmenting path")
    - $\leq \frac{mn}{2}$ augmenting paths (for any capacities)
    - $O(m^2 n)$ total running time
- There are algorithms for max flow running in $O(mn)$ time.