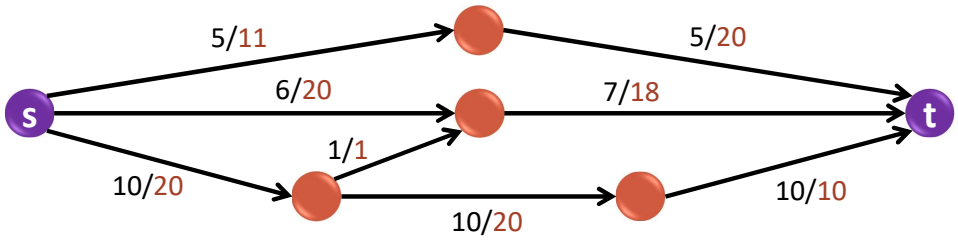
Network Flow

EECS 336: Design and Analysis of Algorithms.
Konstantin Makarychev

Network

- Network:
  - Directed graph $G = (V, E)$.
  - Two designated nodes (terminals): s-source and t-sink.
  - Every edge e has a capacity $c(e)$.

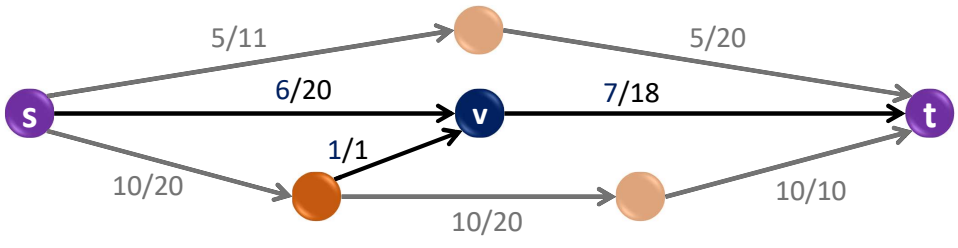


Feasible Flow

Flow $f: E \to \mathbb{R}^+$. We route $f(e)$ units of flow over each edge e.

a. Flow conservation constraints: for all vertices v but s and t,

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$



Feasible Flow

Flow $f: E \to \mathbb{R}^+$. We route $f(e)$ units of flow over each edge e.

a. Flow conservation constraints: for all vertices v but s and t,

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

b. Capacity constraints: for all edges e:

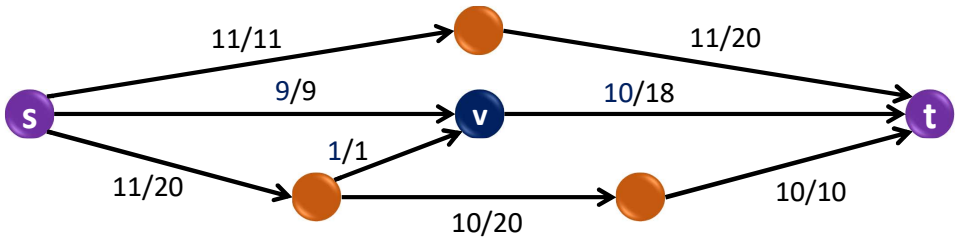
$$f(e) \leq c(e)$$

The value of the flow f equals

$$\sum_{s \to v \in E} f(s \to v) - \sum_{s \to v \in E} f(s \to v)$$

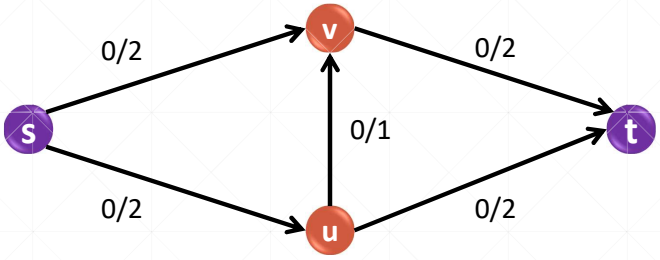
Maximum Flow Problem

Given a network $G = (V, E)$, find the maximum feasible flow from s to t.

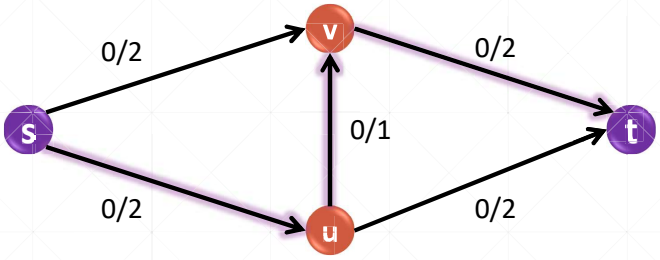


Ford–Fulkerson algorithm

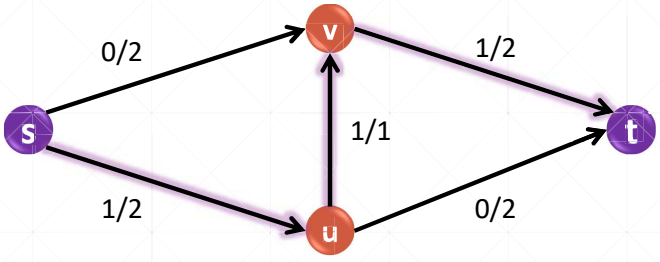
Naïve Greedy Algorithm Fails



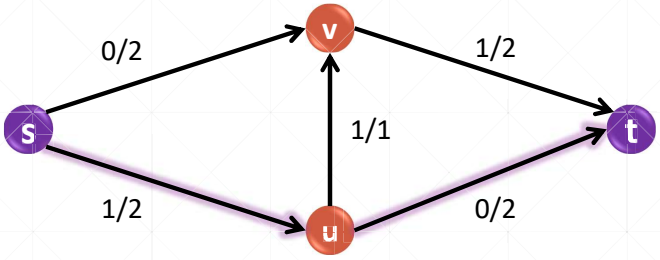
Naïve Greedy Algorithm Fails



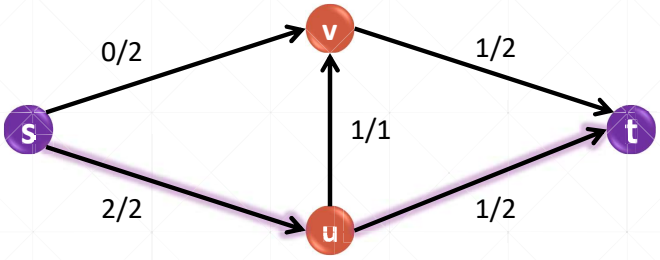
Naïve Greedy Algorithm Fails



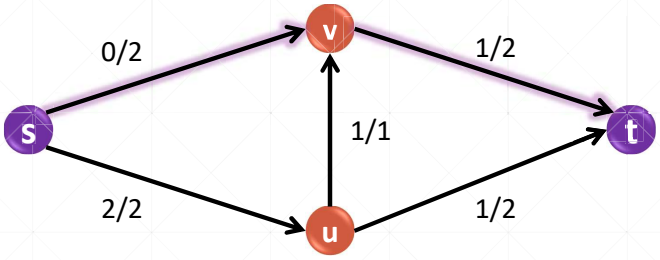
Naïve Greedy Algorithm Fails



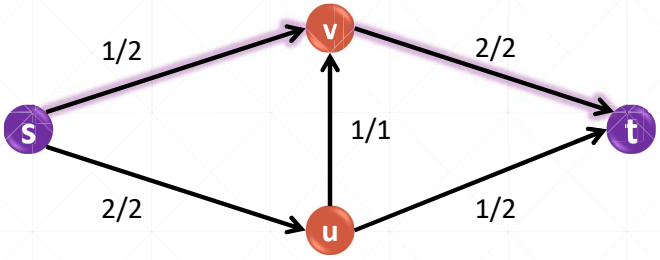
Naïve Greedy Algorithm Fails



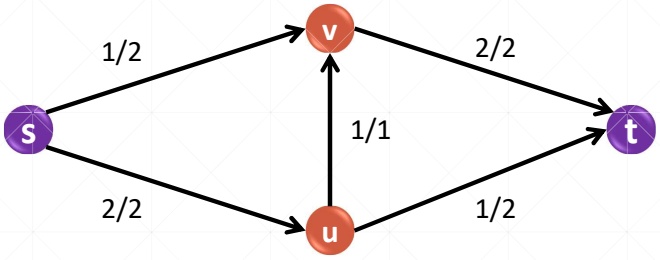
Naïve Greedy Algorithm Fails



Naïve Greedy Algorithm Fails

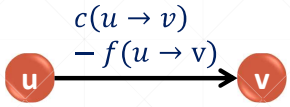


Naïve Greedy Algorithm Fails

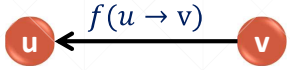


Residual network G_f

- Residual network is a graph on the same set of vertices as the original network G.
- For every original edge $u \to v$, G_f has edge $u \to v$ if $f(u \to v) < c(u \to v)$.
  - $c_f(u \to v) = c(u \to v) - f(u \to v)$

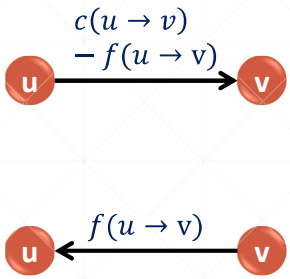


- For every original edge $u \to v$, G_f has edge $v \to u$ if $f(u \to v) > 0$.
  - $c_f(v \to u) = f(u \to v)$



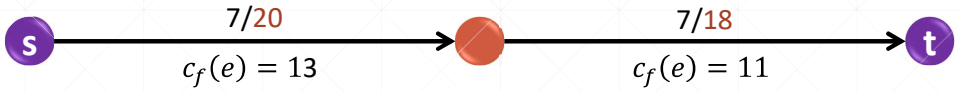
Residual network G_f

- Residual network is a graph on the same set of vertices as the original network G.
- For every original edge $u \to v$, G_f has edge $u \to v$ if $f(u \to v) < c(u \to v)$.
  - $c_f(u \to v) = c(u \to v) - f(u \to v)$
- For every original edge $u \to v$, G_f has edge $v \to u$ if $f(u \to v) > 0$.
  - $c_f(v \to u) = f(u \to v)$



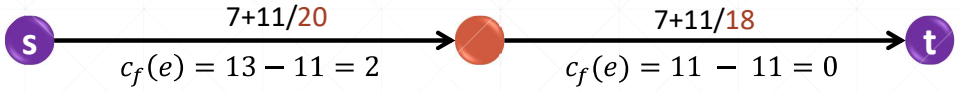
Bottleneck Edge

- Bottleneck edge in P is the edge with the min residual capacity.
- After augmenting the flow along P, the bottleneck edge disappears.



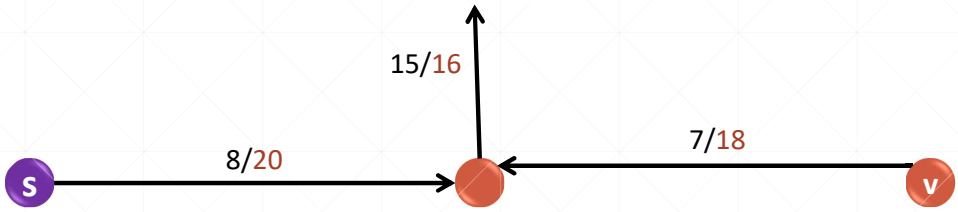
Bottleneck Edge

- Bottleneck edge in P is the edge with the min residual capacity.
- After augmenting the flow along P, the bottleneck edge disappears.



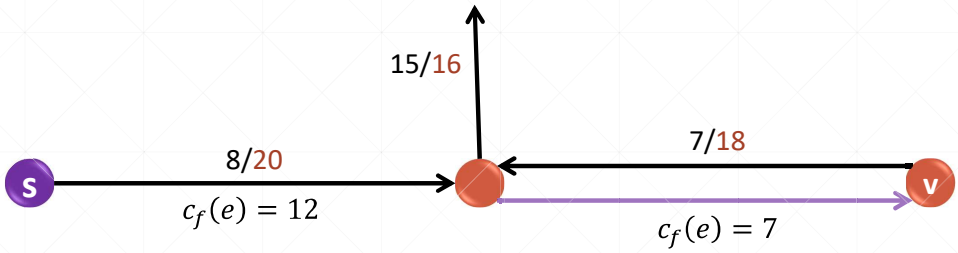
Bottleneck Edge

- Bottleneck edge in P is the edge with the min residual capacity.
- After augmenting the flow along P, the bottleneck edge disappears.



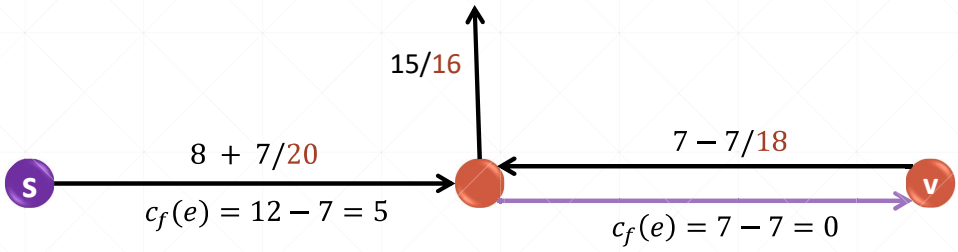
Bottleneck Edge

- Bottleneck edge in P is the edge with the min residual capacity.
- After augmenting the flow along P, the bottleneck edge disappears.



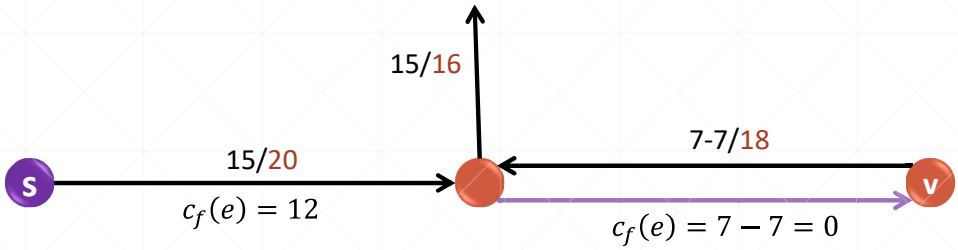
Bottleneck Edge

- Bottleneck edge in P is the edge with the min residual capacity.
- After augmenting the flow along P, the bottleneck edge disappears.



Bottleneck Edge

- Bottleneck edge in P is the edge with the min residual capacity.
- After augmenting the flow along P, the bottleneck edge disappears.



Ford–Fulkerson algorithm

Input: network $G = (V, E)$ with capacities c, source node s, sink node t

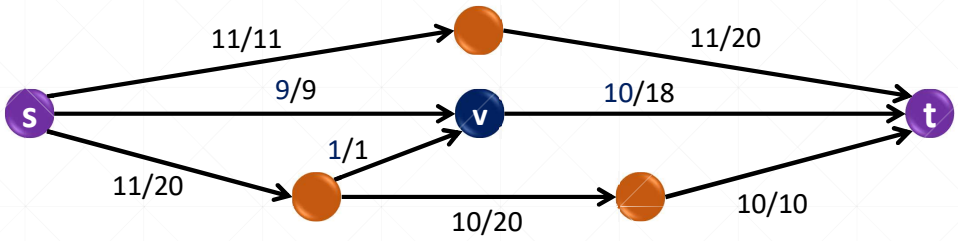
- Initialize flow: Let $f(u, v) = 0$ for all $(u, v) \in E$
- **Repeat:**
  - Compute residual flow G_f.
  - Find* a path P from s to t in G_f.
  - Augment the flow along P by the bottleneck capacity.
  - If there is no path from s to t in G_f return f.

Ford–Fulkerson algorithm

- **Claim:** If all capacities are integral, then Ford-Fulkerson algorithm eventually terminates and outputs a flow in G.
- Is this flow maximum possible?

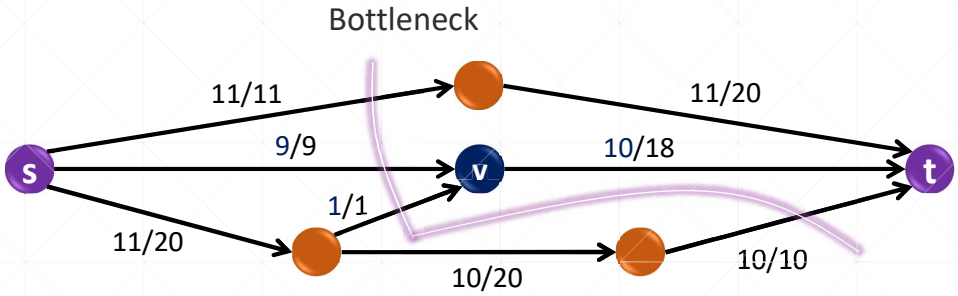
Maximum Flow Problem

Given a network $G = (V, E)$, find the maximum feasible flow from s to t.



Maximum Flow Problem

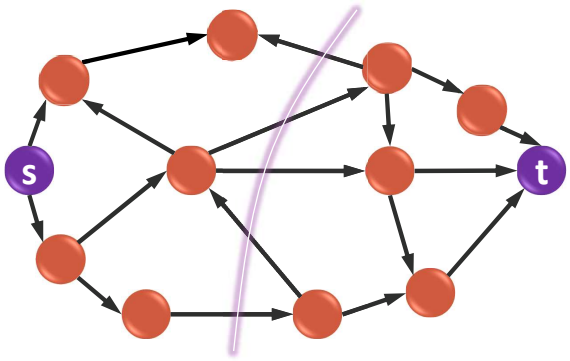
Given a network $G = (V, E)$, find the maximum feasible flow from s to t.



st-Cuts

- Cut (A, B) is partitioning of the set of all vertices into two sets A and B s.t. $s \in A$ and $t \in B$.
- Capacity of the cut:

$$cut(A, B) = \sum_{e \in A \to B} c(e)$$



Min Cut = Max Flow = Ford-Fulkerson

- We prove that

$$\textbf{Min Cut} \geq \textbf{Max Flow} \geq \textbf{Ford} - \textbf{Fulkerson} \geq \textbf{Min Cut}$$

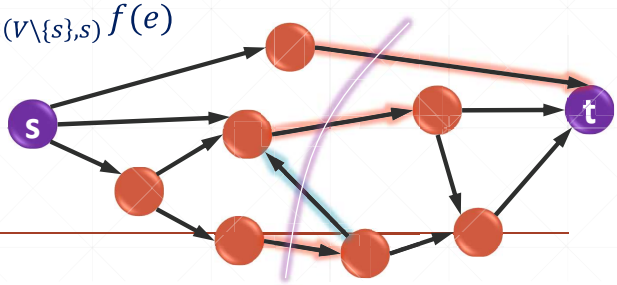
Flow-value Lemma

Network $G = (V, E)$. Feasible flow F from s to t. st-Cut $(S, \bar{S})$.

$$\text{val}(f) = \sum_{e \in (S,\bar{S})} f(e) - \sum_{e \in (\bar{S},S)} f(e)$$

where

- $\text{val}(f) = \sum_{e \in (s,V\setminus\{s\})} f(e) - \sum_{e \in (V\setminus\{s\},s)} f(e)$
- $f(e) = 0$, if $u \notin E$



Flow-value Lemma

Network $G = (V, E)$. Feasible flow F from s to t. st-Cut $(S, \bar{S})$.

$$\text{val}(f) = \sum_{e \in (S,\bar{S})} f(e) - \sum_{e \in (\bar{S},S)} f(e) = (*)$$

Proof:

$$val(f) = \sum_{u \in S} \left[\sum_{v:(u,v) \in E} f(e) - \sum_{v:(v,u) \in E} f(e) \right] = (*)$$

Details are on the board.

Min Cut = Max Flow = Ford-Fulkerson

We prove that

a. **Min Cut ≥ Max Flow:** By the flow-value lemma, the value of the flow equals the flow across the min cut, which is upper bounded by **Min Cut**.

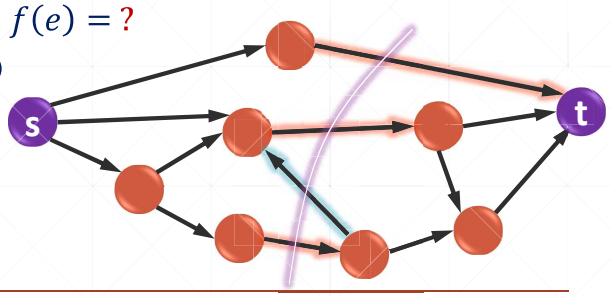
b. **Max Flow ≥ Ford − Fulkerson:** Ford-Fulkerson gives a feasible solution to the Max Flow Problem.

c. **Ford − Fulkerson ≥ Min Cut** (next slide)

Ford–Fulkerson ≥ Min Cut

Let f be the flow returned by Ford-Fulkerson and S be the set of nodes reachable from s in G_f. By the flow-value lemma:

$$\text{val}(f) = \sum_{e \in (S,\bar{S})} f(e) - \sum_{e \in (\bar{S},S)} f(e) = ?$$

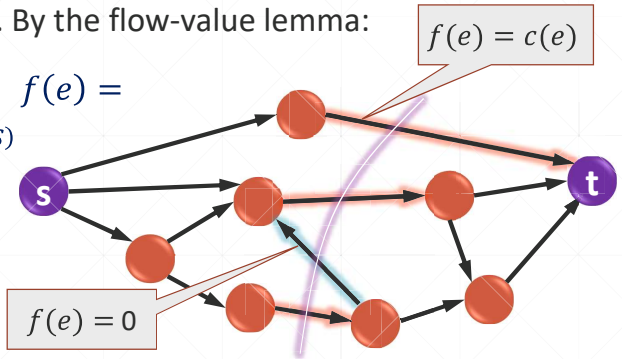


Ford–Fulkerson ≥ Min Cut

Let f be the flow returned by Ford-Fulkerson and S be the set of nodes reachable from s in G_f. By the flow-value lemma:

$$\text{val}(f) = \sum_{e \in (S,\bar{S})} f(e) - \sum_{e \in (\bar{S},S)} f(e) =$$

$$= \sum_{e \in (S,\bar{S})} c(e) \geq \textbf{Min Cut}$$



- Min cut – the minimum st-cut.
- Max flow – max flow from s to t.

Max-flow min-cut theorem:

The value of max flow = The capacity of min cut

Integral flow theorem:

If each edge in a flow network has integral capacity, then there exists an integral maximum flow.

Edmonds-Karp Algorithm

- Start with a feasible flow $f = 0$.
- while (f is not max flow)
  - Construct residual network G_f wrt flow f.
  - Find the *shortest path* P from s to t in G_f.
  - Push/augment flow along path P.
- return f

Lemma 1

- Throughout the algorithm, the length of a shortest augmenting path never decreases.

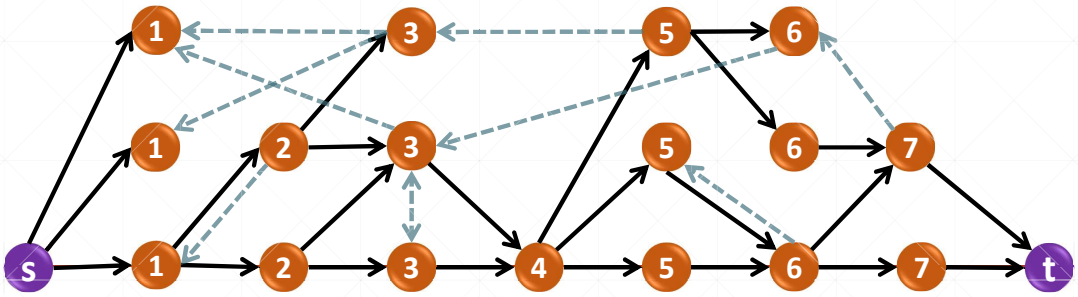
Lemma 2

- After at most m shortest-path augmentations, the length of a shortest augmenting path strictly increases.

Lemma 1 + Lemma 2 imply the Edmonds-Karp algorithm terminates in mn steps.

Level Graph

- Let $d(u)$ be the shortest path distance from s to u. Let $L = d(t)$.
- Place vertex u in row $d(u)$.
- Forward edges go from i to $j \geq i$. Backward edges go from j to $i < j$.

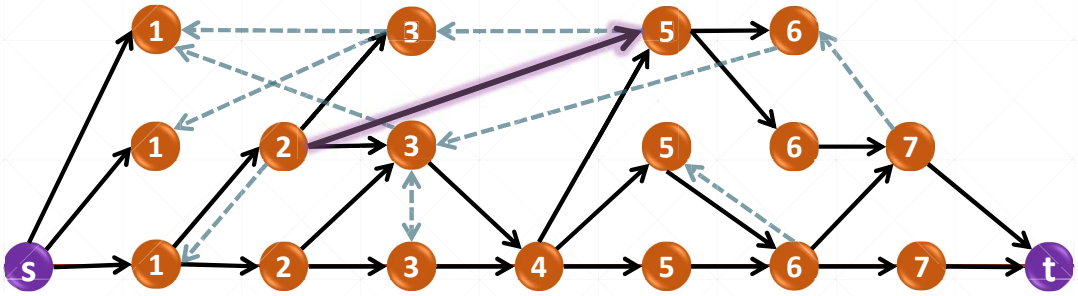


Level Graph

Claim A: If $u \rightarrow v$ is a forward edge, then

level of v = level of u + 1

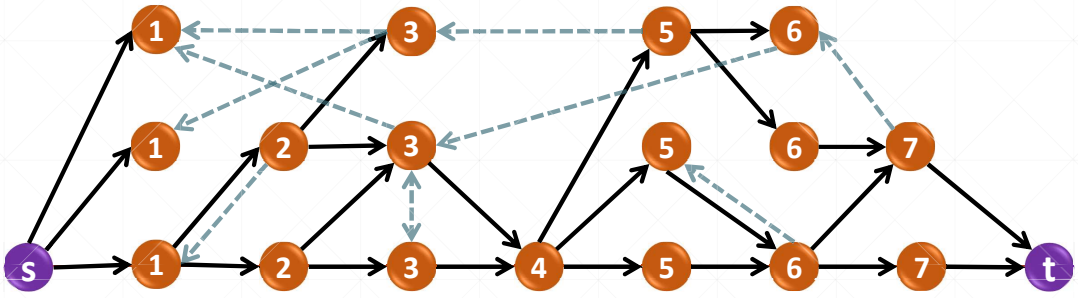
If graph satisfies (A) then: Every path $s \rightarrow t$ of length L consists only of forward edges, where L is the level of t. Distance from s to t is $\geq L$.



Level Graph

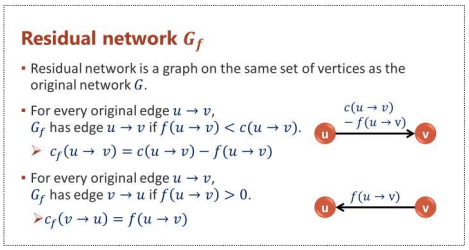
Claim B: If we add arbitrary ***backward*** edges and remove arbitrary edges, then the distance from s to t will still be $\geq L$.

Proof: After the update, the graph still satisfies **(A)**
(note: we don't change the positions of vertices).



Lemma 1

- Throughout the algorithm, the length of the shortest augmenting path never decreases.



Proof:

- f and f'— flow before and after a shortest-path augmentation along path P.
- G_f and $G_{f'}$ — corresponding residual networks.

Consider the level graph for G_f. We update capacities in f along the path P. Hence, only edges from $\text{reverse}(P)$ may get added to $G_{f'}$. Hence, we don't add forward edges. Lemma 1 follows from Claim B.

Lemma 2

- After at most m shortest-path augmentations, the length of the shortest augmenting path strictly increases.

Proof: Consider the phase of ALG when $d(s,t) = L$.

- Fix the level graph at the first step when $d(s,t) = L$.
- At every step, in this phase, all shortest paths $s \to t$ consist only of forward edges.
- Thus, no forward edges are added, but at least one forward edge – the bottleneck edge – is deleted.
- After m steps all forward edges are deleted or $d(s,t)$ increases.

Lemma 1

- Throughout the algorithm, the length of a shortest augmenting path never decreases.

Lemma 2

- After at most m shortest-path augmentations, the length of a shortest augmenting path strictly increases.

Lemma 1 + Lemma 2 imply the Edmonds-Karp algorithm terminates in mn steps.

Applications of Max Flow and Min Cut

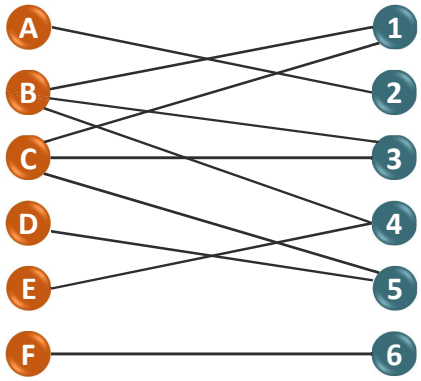
Routing

- Routing vehicles on the road.
- Routing electricity.
- Routing internet traffic.

Bipartite Matching

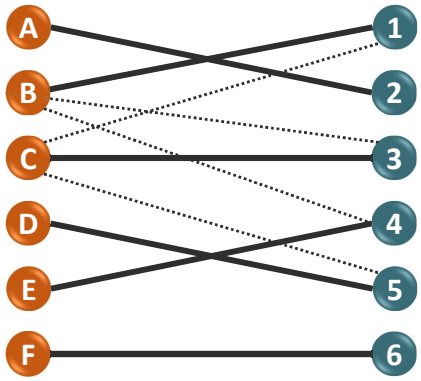
Bipartite Matching

- Alice, Bob, Carol, Dan, Eve, and Frank want to organize a party. They would like to prepare dishes 1,…,6. Each of them knows how to cook some subset of these dishes. What dish should each person prepare?
- Need to find a matching.



Bipartite Matching

- Alice, Bob, Carol, Dan, Eve, and Frank want to organize a party. They would like to prepare dishes 1,…,6. Each of them knows how to cook some subset of these dishes. What dish should each person prepare?
- Need to find a matching.

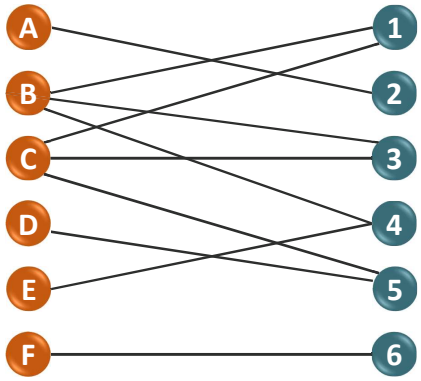


Bipartite Matching

Given: Bipartite graph

$$G = (L, R, E)$$

Goal: Find a perfect matching i.e., a subset of edges $M \subset E$ s.t. each vertex is incident to exactly one edge in M.

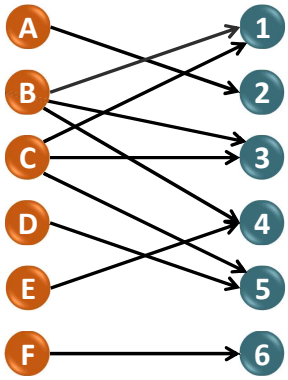


Bipartite Matching

Given: Bipartite graph

$$G = (L, R, E)$$

Goal: Find a perfect matching i.e., a subset of edges $M \subset E$ s.t. each vertex is incident to exactly one edge in M.



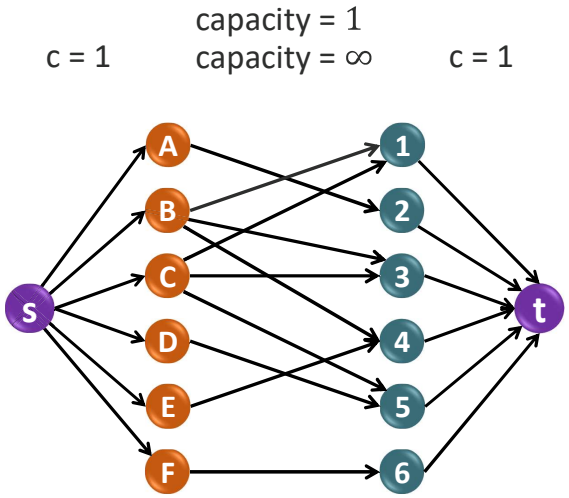
Bipartite Matching

Given: Bipartite graph

$$G = (L, R, E)$$

Goal: Find a perfect matching i.e., a subset of edges $M \subset E$ s.t. each vertex is incident to exactly one edge in M.

Consider a flow network with edges of capacity 1 and ∞.



Bipartite Matching

Given: Bipartite graph G.

Goal: Find a perfect matching.

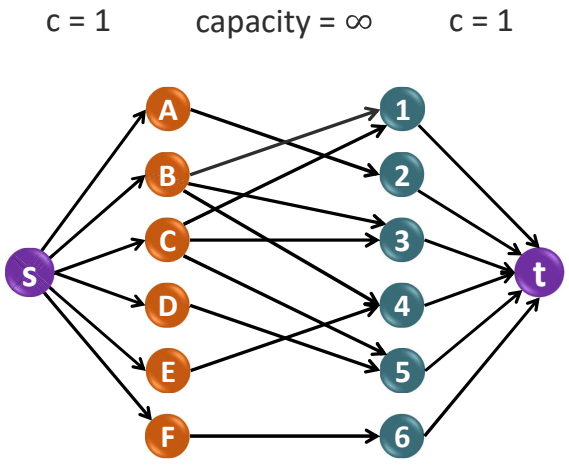
n = #vertices on each side

Thm:

There is perfect matching in G

⇕

the max flow equals n

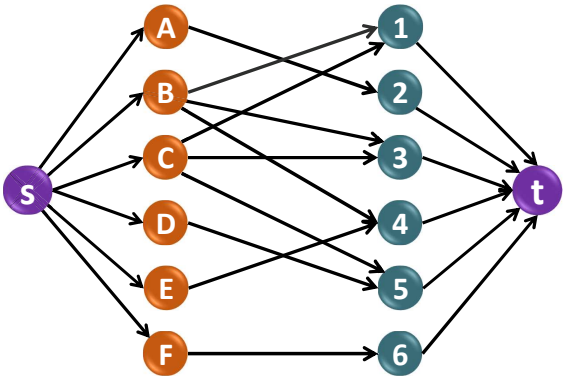


Bipartite Matching

Proof:

⇒ If there is a perfect matching in G, then we can route the flow via this matching.

c = 1 capacity = ∞ c = 1



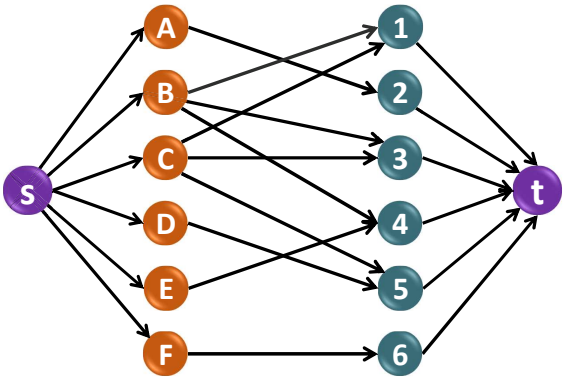
Bipartite Matching

Proof:

⇒ If there is a perfect matching in G, then we can route the flow via this matching.

⇐ Since all capacities are integral, there exists an integral max flow ($f(e) \in \mathbb{Z}$). Drop all edges with $f(e) = 0$.

c = 1 capacity = ∞ c = 1



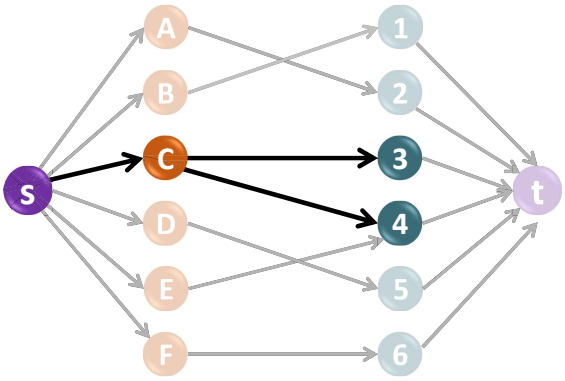
Bipartite Matching

Proof:

⇐ Since all capacities are integral, there exists an integral max flow ($f(e) \in \mathbb{Z}$). Drop all edges with $f(e) = 0$.

The flow to each vertex on the left ≤ 1. It is 1 because max flow $= n$. Hence, flow out of each vertex must be also 1.

c = 1 capacity = ∞ c = 1

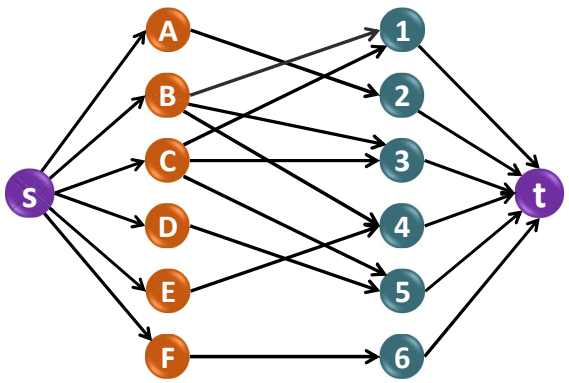


Algorithm

- Solve the Max Flow problem.
- Return all edges e between L and R with $f(e) = 1$.

When does the algorithm succeed?

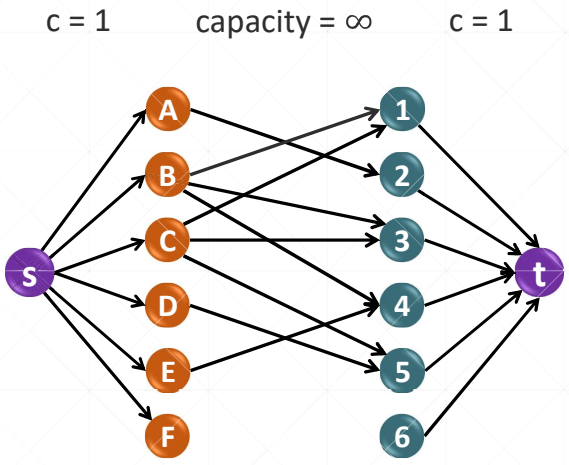
Min cut = Max flow $= n$.



Maximum Matching

Algorithm for Maximum Matching

- Solve the Max Flow problem.
- Return all edges e between L and R with $f(e) = 1$.



Hall's Theorem

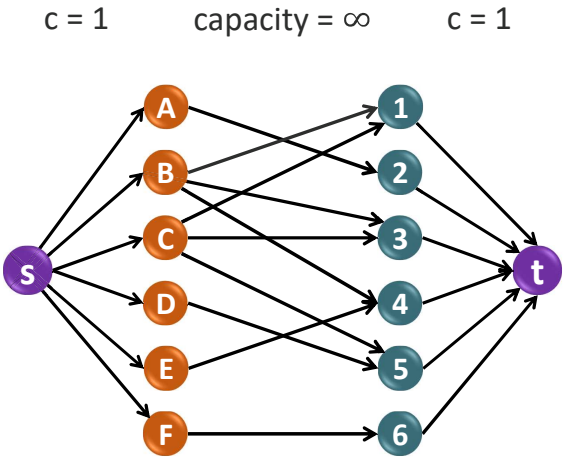
Hall's Theorem

There is a perfect matching in G

$\Updownarrow$

for every $S \subset L$, we have
$|N(S)| \geq |S|$

Here, $N(S)$ is the set of neighbors of S.



Hall's Theorem

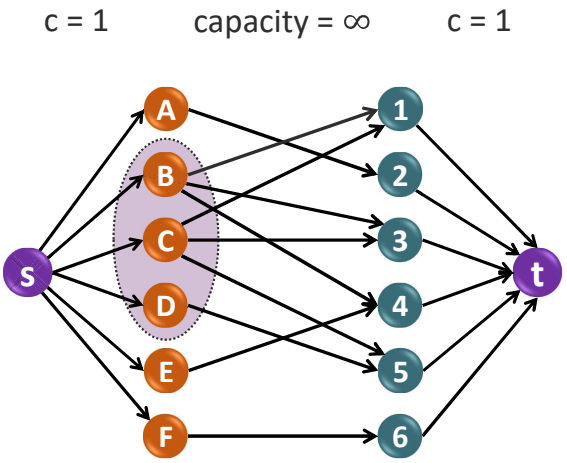
There is a perfect matching in G

$\Updownarrow$

for every $S \subset L$, we have
$|N(S)| \geq |S|$

Here, $N(S)$ is the set of neighbors of S.

$\Rightarrow$ If there is a perfect matching in G, then "the image of S" contains at least $|S|$ elements.



Hall's Theorem

There is a perfect matching in G

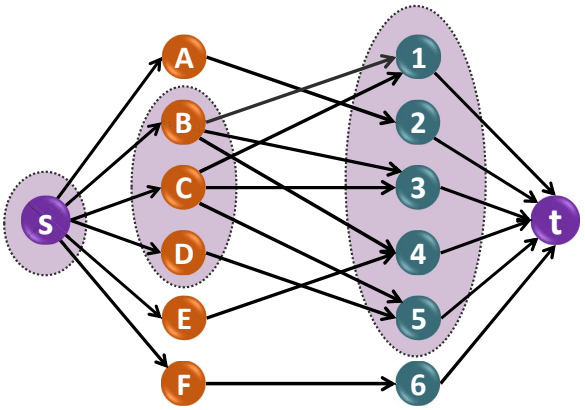
$\Updownarrow$

for every $S \subset L$, we have $|N(S)| \geq |S|$

$\Leftarrow$ Consider min cut (A, B). Its capacity $< \infty$. Thus, edges between L and R cannot cross it. Let $S = A \cap L$; $T = A \cap R$.

Then, $N(S) \subset T$. We get

$$cut(A, B) = n - |S| + |T|$$
$$\geq n - |S| + |N(S)| \geq n$$

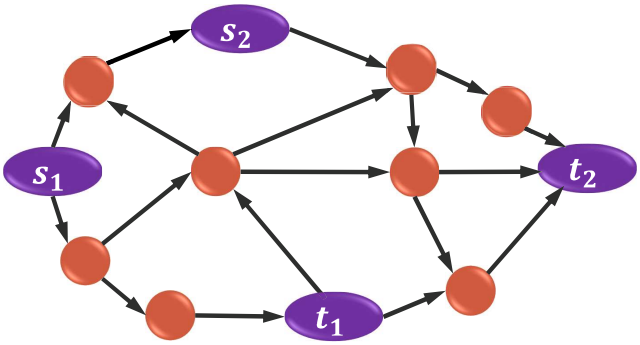


Multiple Sources & Sinks

Multiple Sources & Sinks

Given: Flow network w/ many sources: $s_1, \ldots, s_{k'}$ and many sinks: $t_1, \ldots, t_{k''}$.

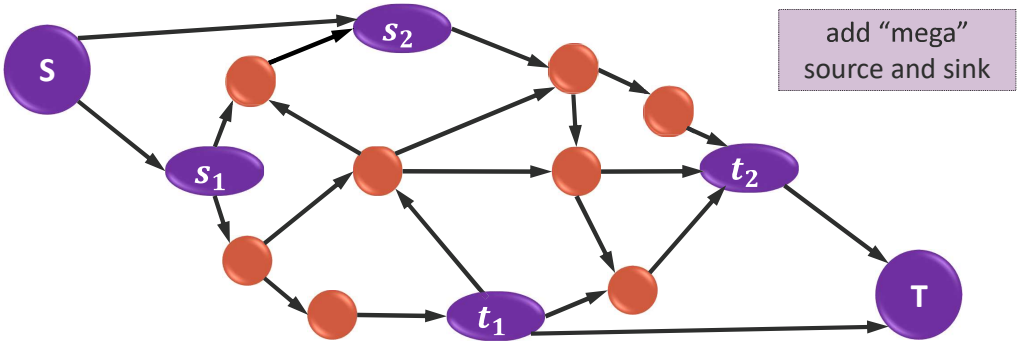
Goal: Maximize the total amount of flow from all sources to all sinks.



Multiple Sources & Sinks

Given: Flow network w/ many sources: $s_1, \ldots, s_{k'}$ and many sinks: $t_1, \ldots, t_{k''}$.

Goal: Maximize the total amount of flow from all sources to all sinks.



Questions