CPSC 490 – Problem Solving in Computer Science

Lecture 22: Maximum Flow

Jason Chiu and Raunak Kumar

2017/03/17

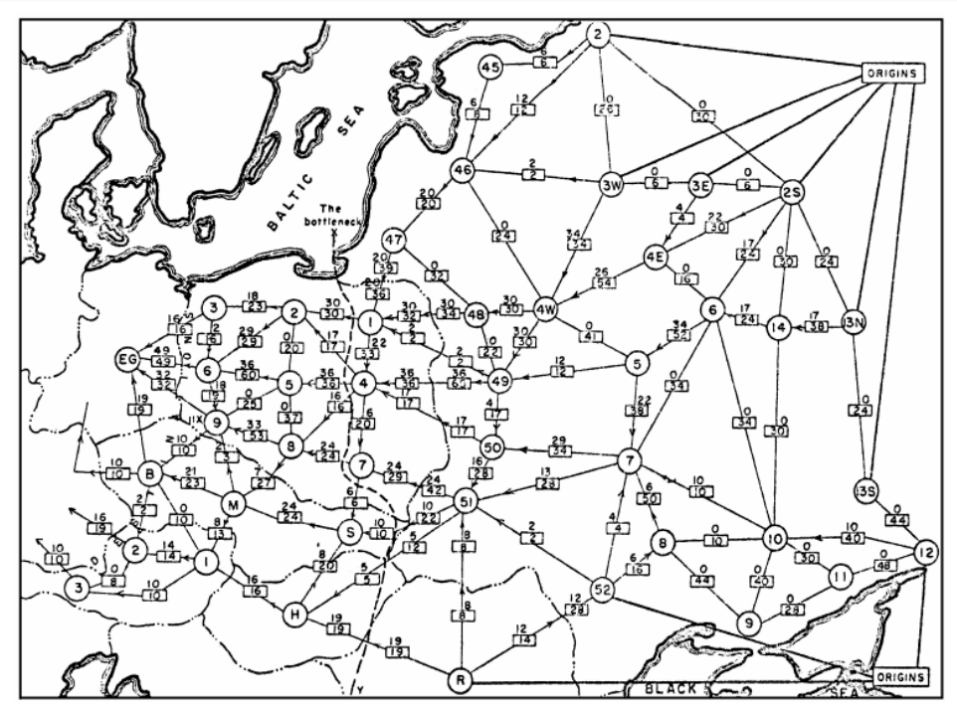
University of British Columbia

Last Time

We discussed Binary / Ternary search, and discussed assignment 2,3,4 solutions.

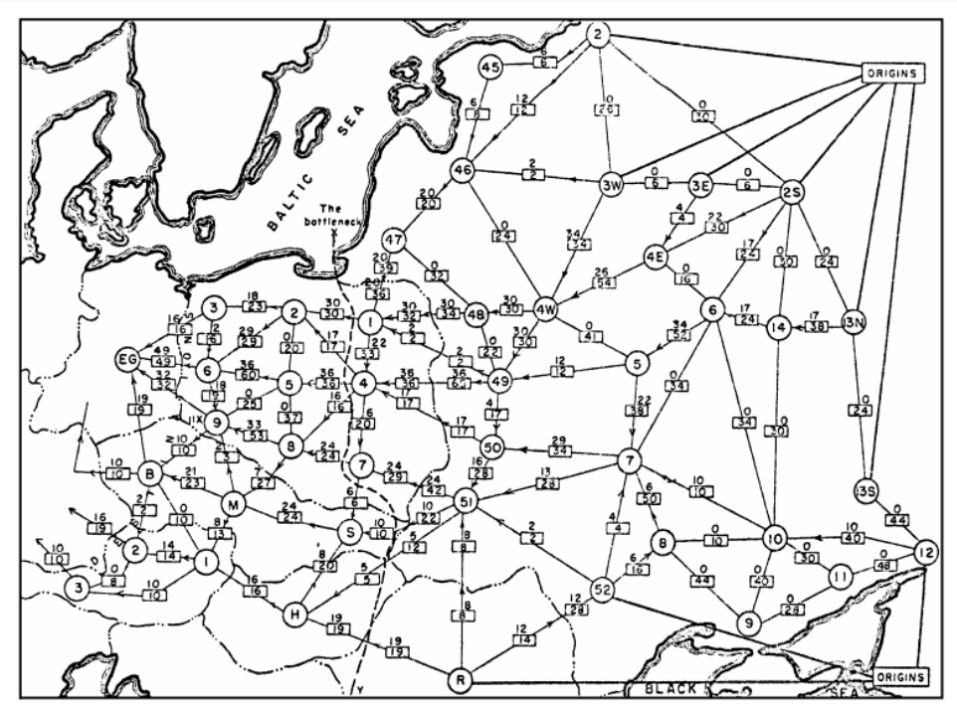
Today we'll move on to a new topic.

Cold War



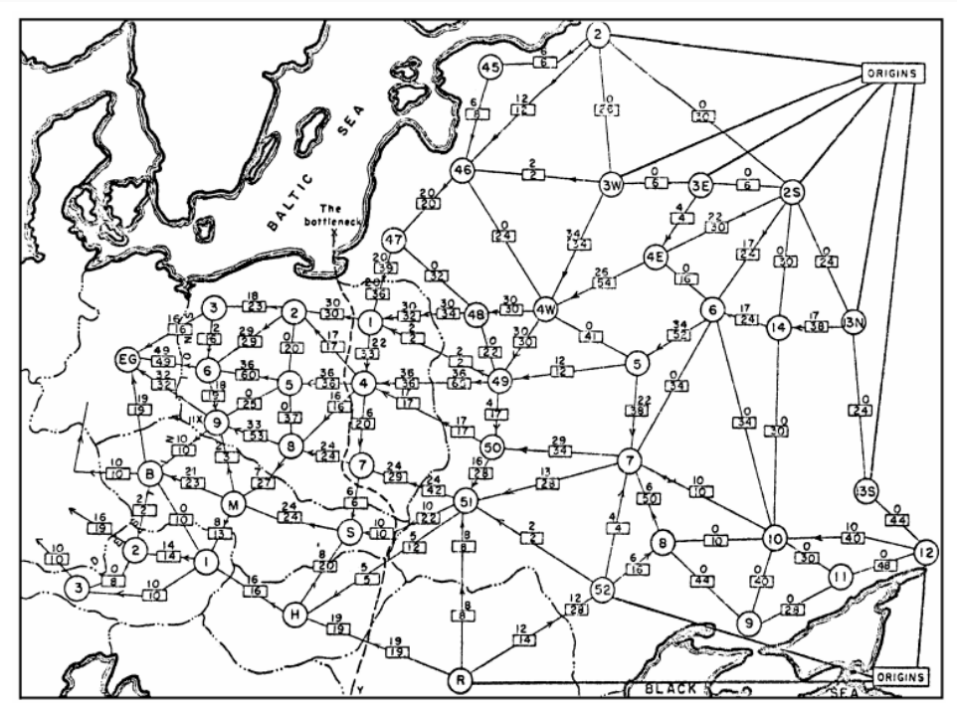
- You are given a map of the railroads in Europe.

Cold War



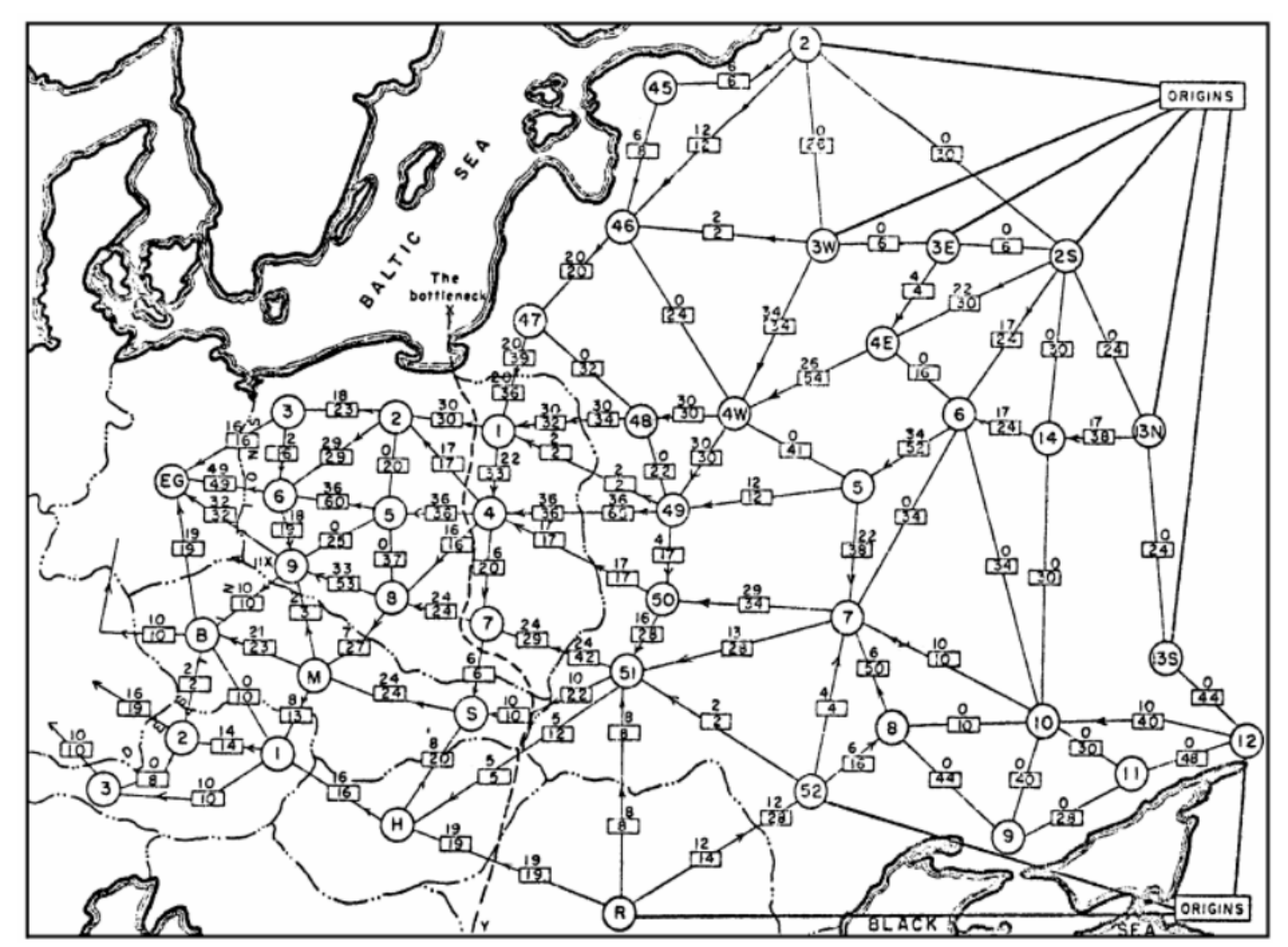
- You are given a map of the railroads in Europe.
- The CIA wants wants to disconnect the Soviet Union from the rest of Europe (even if Trump doesn't).

Cold War



- You are given a map of the railroads in Europe.
- The CIA wants wants to disconnect the Soviet Union from the rest of Europe (even if Trump doesn't).
- Each railroad has a capacity and the cost to blow up a railroad is proportional to its capacity.
- What's the cheapest way of blowing up railroads?

Cold War



Why ask computer scientists and mathematicians???

Maximum Flow Problem

Input:

- A directed graph $G = (V, E)$

Maximum Flow Problem

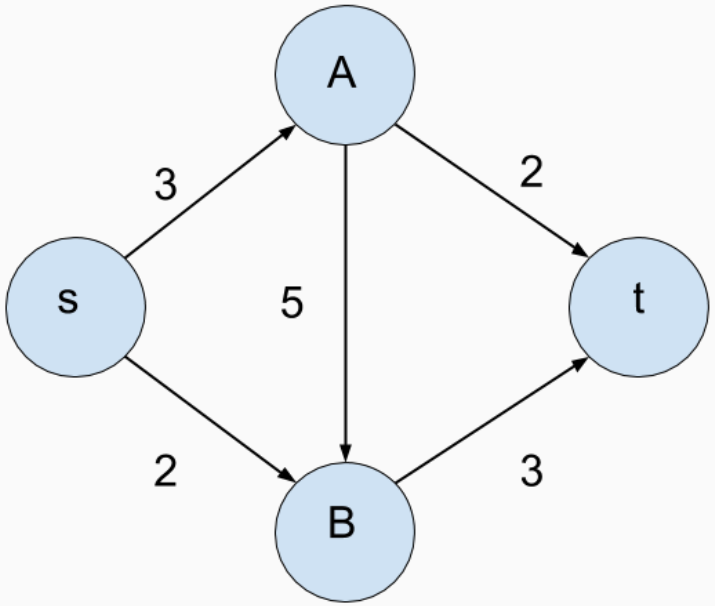
Input:

- A directed graph $G = (V, E)$
- A **source** vertex $s \in V$ - no incoming edges
- A **sink** vertex $t \in V$ - no outgoing edges

Maximum Flow Problem

Input:

- A directed graph $G = (V, E)$
- A **source** vertex $s \in V$ - no incoming edges
- A **sink** vertex $t \in V$ - no outgoing edges
- A **capacity** c_e for each edge $e = (u, v)$, with $c_e \in \mathbb{Z}_+$



Maximum Flow Problem

Flow: Each edge $e = (u, v)$ has a non-negative flow f_e that satisfies:

- Capacity Constraints: $f_e \leq c_e$

Maximum Flow Problem

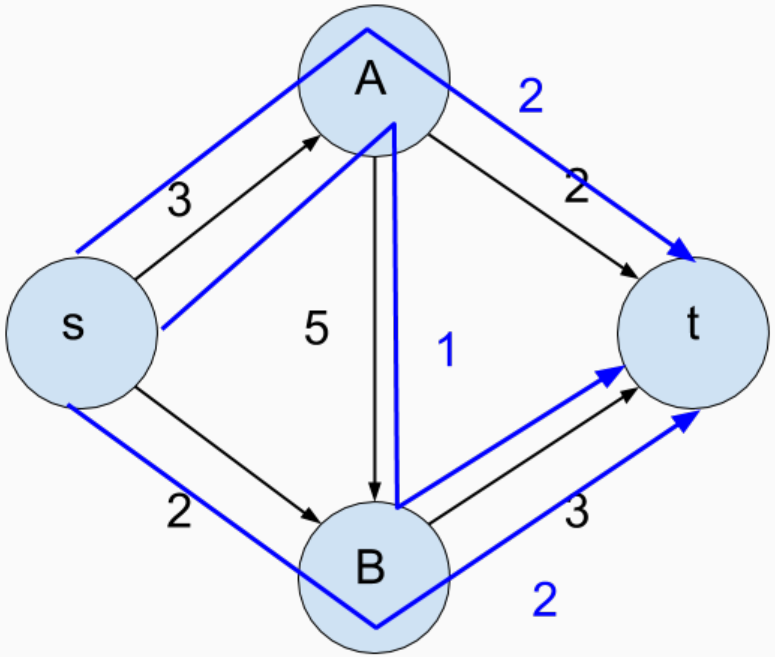
Flow: Each edge $e = (u, v)$ has a non-negative flow f_e that satisfies:

- Capacity Constraints: $f_e \leq c_e$
- Conservation Constraints: For all vertices v except s and t, amount of flow entering v = amount of flow exiting v

Output: Maximum flow in the graph G.

Maximum Flow Problem

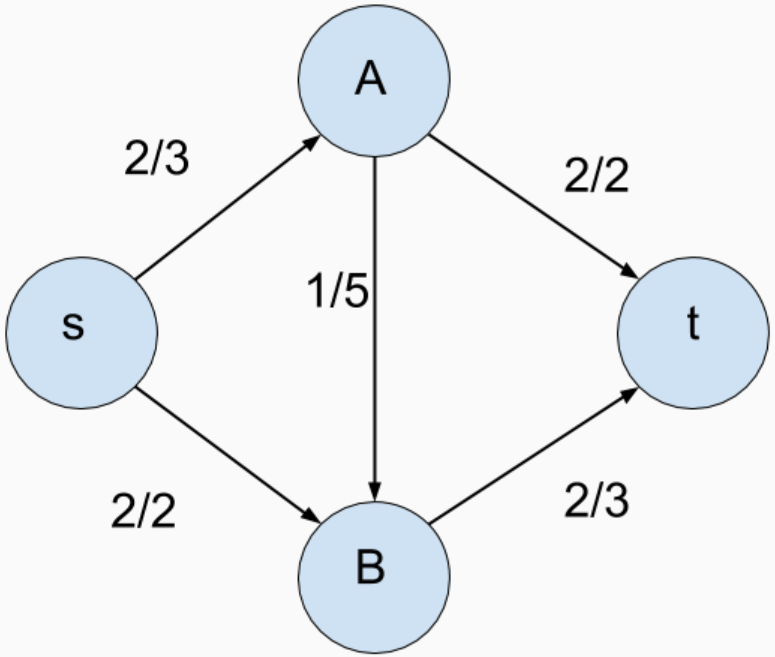
The blue lines represent the flows.



Maximum Flow = size(f) = 5

Maximum Flow Problem

x/*y* indicates *x* units of flow on an edge with original capacity *y*.



Greedy Attempt

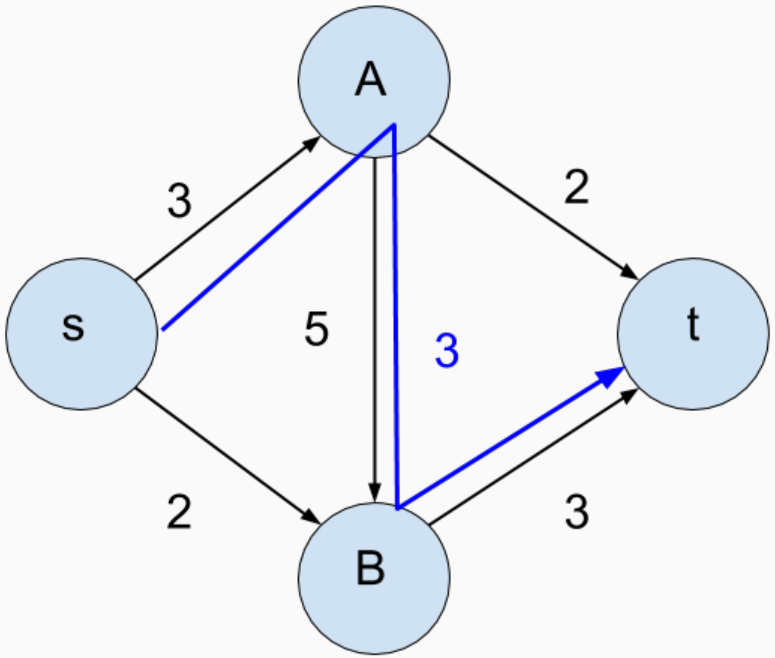
Let's try the following greedy algorithm:

```
while there is an augmenting path
  pick an augmenting path
  augment the flow:
    increase the flow on edges of augmenting path
```

where an **augmenting path** is an $s - t$ path with positive remaining capacity on its edges.

Does Greedy Work?

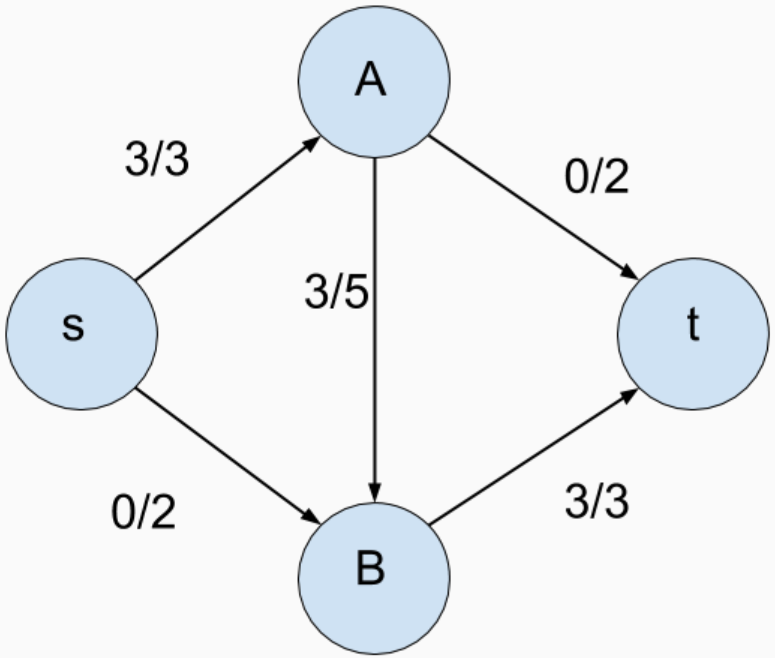
No! We know that the max flow in the following network is 5, but greedy could produce 3.



size(f) = 3

Does Greedy Work?

No! We know that the max flow in the following network is 5, but greedy could produce 3.



Residual Graph

- When we pick an augmenting path, we might pick a “bad” one.
- What if we somehow had a way of “undoing” these bad operations?
- This leads us to the concept of **residual networks**.

Residual Graph

A residual graph $G^f = (V, E^f)$ is the following:

- A graph that depends on the current flow f in the graph.
- Has the same vertices as the original graph (V).

Residual Graph

A residual graph $G^f = (V, E^f)$ is the following:

- A graph that depends on the current flow f in the graph.
- Has the same vertices as the original graph (V).
- Has the edges:
  1. **Forward edge**: $(u, v) \in E$ with capacity $c^f(u, v) = c(u, v) - f(u, v) > 0$

Residual Graph

A residual graph $G^f = (V, E^f)$ is the following:

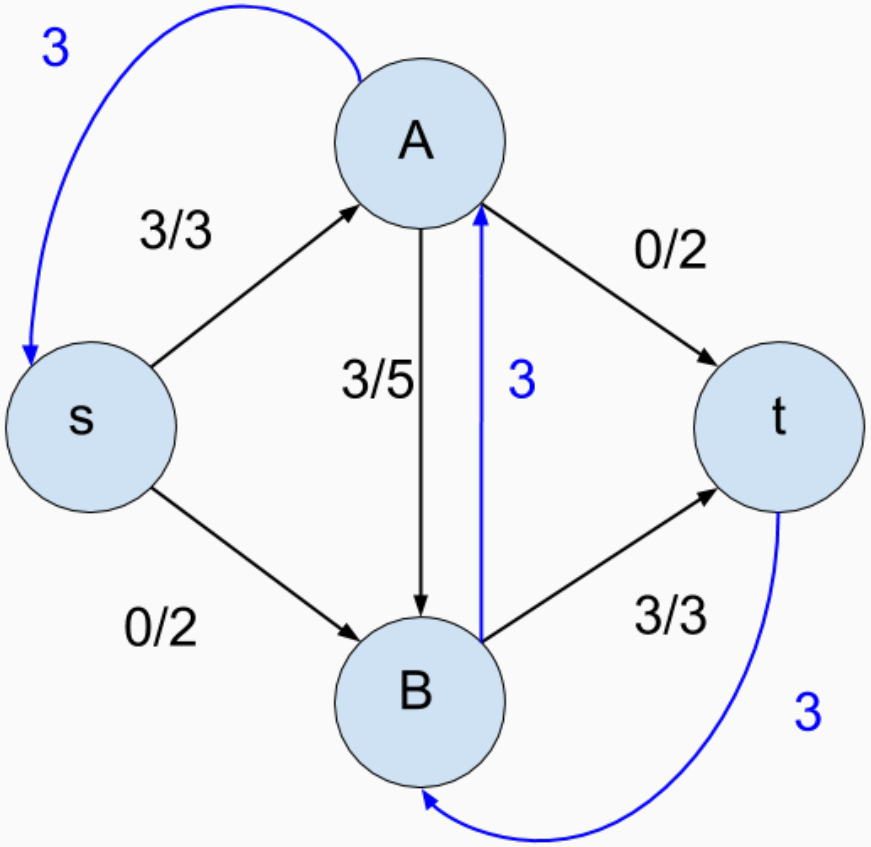
- A graph that depends on the current flow f in the graph.
- Has the same vertices as the original graph (V).
- Has the edges:
  1. **Forward edge**: $(u, v) \in E$ with capacity $c^f(u, v) = c(u, v) - f(u, v) > 0$
  2. **Backward edge**: (v, u) for $(u, v) \in E$ with capacity $c^f(v, u) = f(u, v)$

Residual Graph

A residual graph $G^f = (V, E^f)$ is the following:

- A graph that depends on the current flow f in the graph.
- Has the same vertices as the original graph (V).
- Has the edges:
  1. **Forward edge**: $(u, v) \in E$ with capacity $c^f(u, v) = c(u, v) - f(u, v) > 0$
  2. **Backward edge**: (v, u) for $(u, v) \in E$ with capacity $c^f(v, u) = f(u, v)$
- $c^f(u, v)$ is called the **residual (remaining) capacity** of an edge.

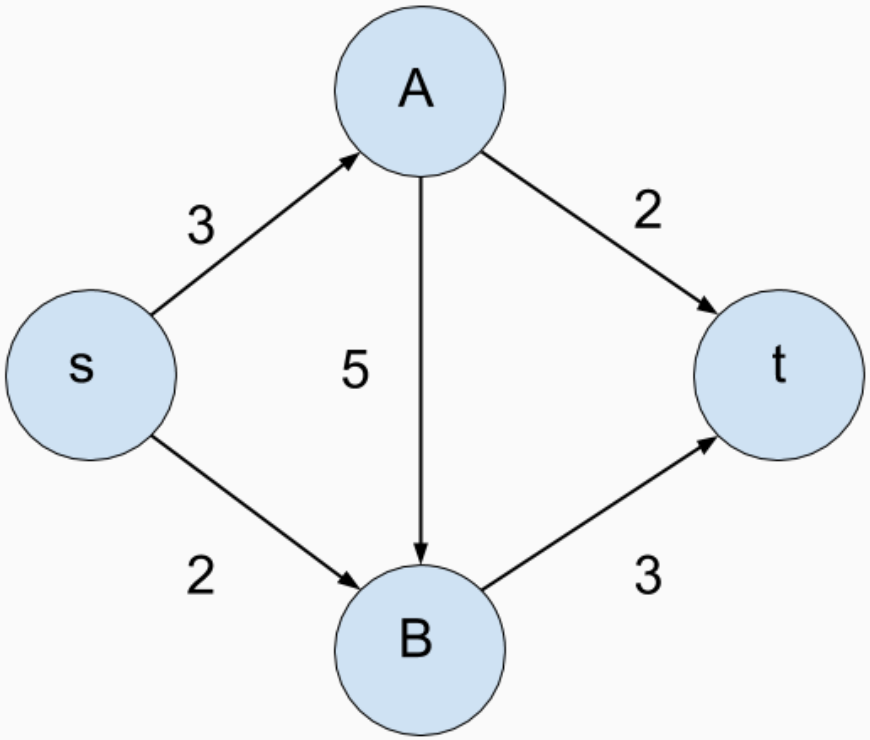
Residual Graph



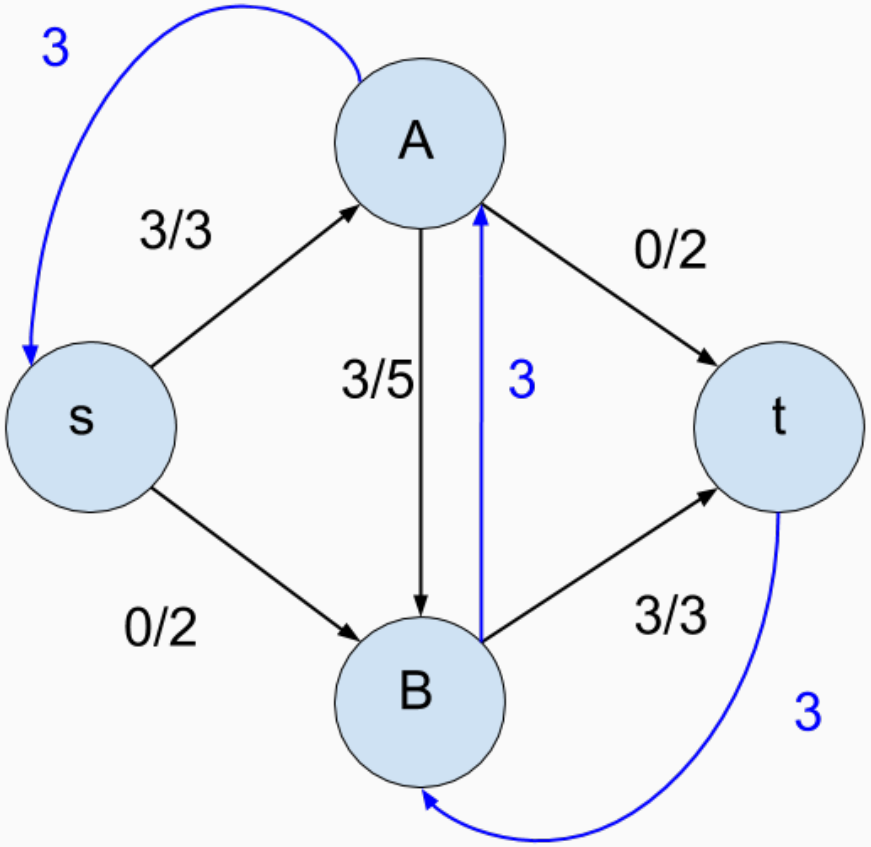
Ford-Fulkerson Algorithm

```
1 while there is an augmenting path in the residual
      graph
2   pick an augmenting path
3   augment the flow:
4     for each edge e in the path:
5     if e is a forward edge:
6       increase the flow on e
7     else:
8       decrease the flow on the corresponding
            forward edge
```

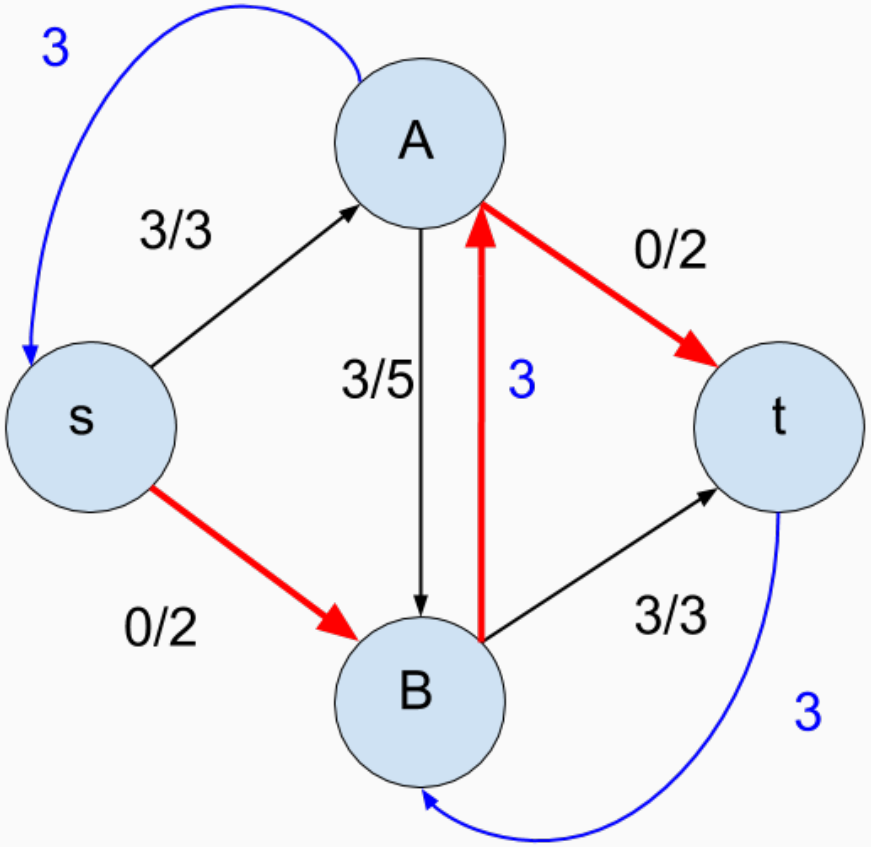
Ford-Fulkerson Example



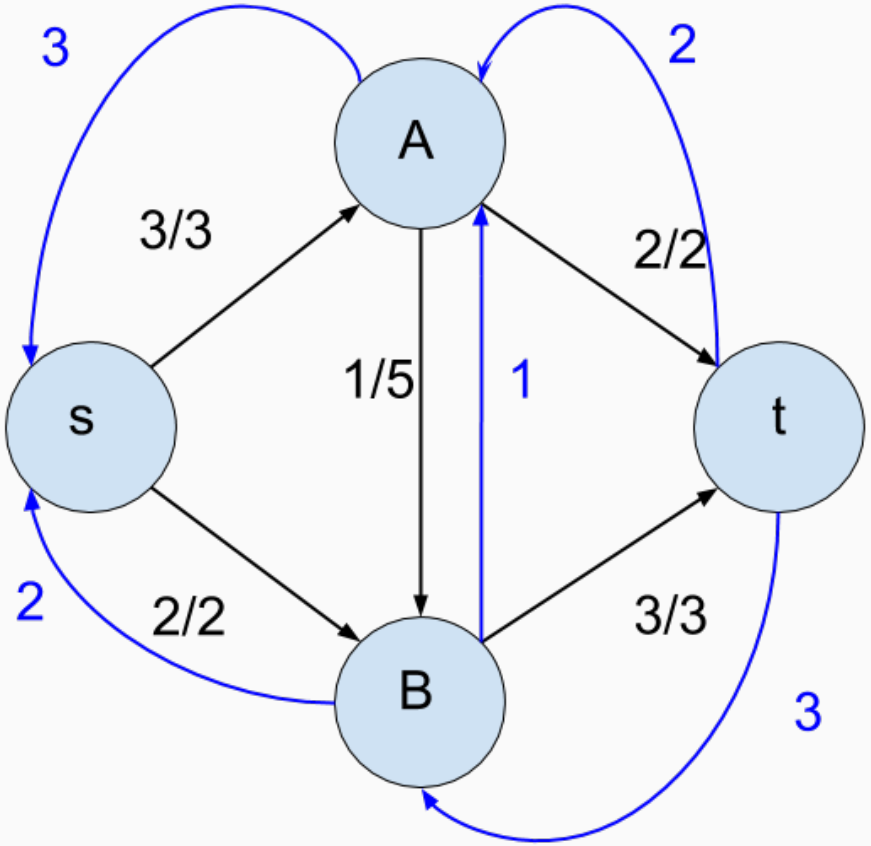
Ford-Fulkerson Example



Ford-Fulkerson Example



Ford-Fulkerson Example



Runtime

What's the runtime?

- Each iteration takes $O(E)$ time to find an augmenting path.
- On integer capacity graphs, each iteration increases the flow by at least 1 unit.
- $O(f)$ iterations in total, where f is the max flow.

So, Ford-Fulkerson takes $O(Ef)$ time.

Runtime

Note:

- On real valued capacity graphs, Ford-Fulkerson is not guaranteed to terminate.

Runtime

Note:

- On real valued capacity graphs, Ford-Fulkerson is not guaranteed to terminate.
- Ford-Fulkerson does not specify how to find the augmenting path.
- It is chosen arbitrarily.
- In code, you'd use DFS or BFS.
- Using BFS to find the augmenting path is called the Edmonds-Karp algorithm, and has a runtime of $O(VE^2)$.
- There are lots of other algorithms to solve max flow!

Cold War Again

How does max flow help us solve our cold war problem?

We want to find the cheapest way of disconnecting Soviet Union from the rest of Europe.

To answer this, we need to know about cuts in a graph.

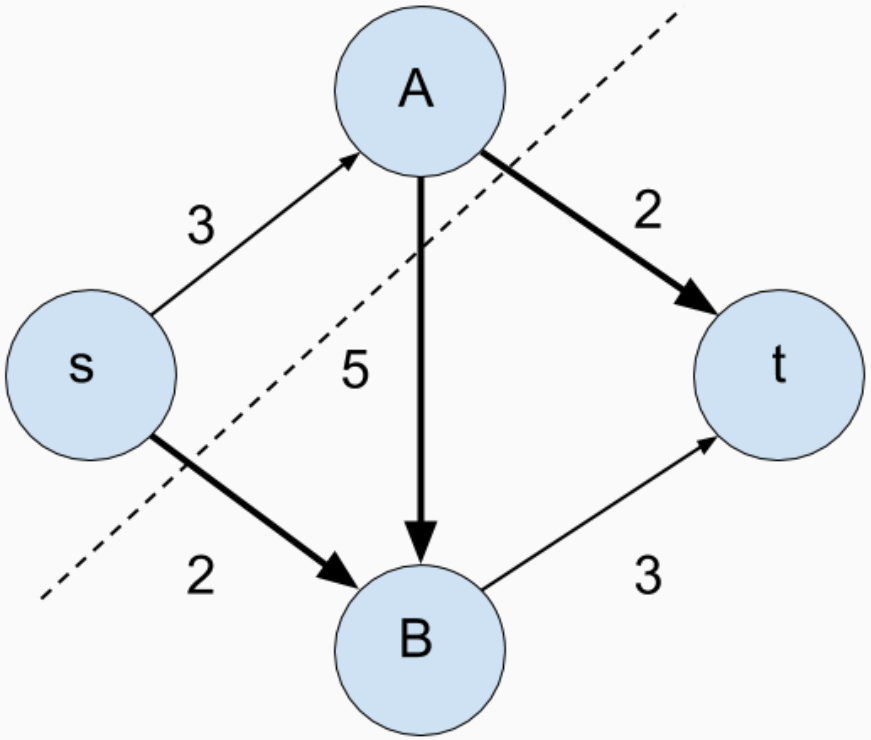
Cuts

- A **cut** is a partition of the vertices of a graph.
- $V = S \cup T$, and $S \cap T = \emptyset$.

Cuts

- In the context of flow networks, we usually refer to an **s-t cut** in which $s \in S$ and $t \in T$.
- Edges only go from the source side to the sink side.
- **Capacity of a cut**: The sum of the capacities of the edges going from vertices in S to vertices in T.
- Edges from T to S do not contribute to the capacity of a cut.

Cuts



S = {s, A} and T = {t,B}
Cap(S,T) = 2 + 5 + 2 = 7

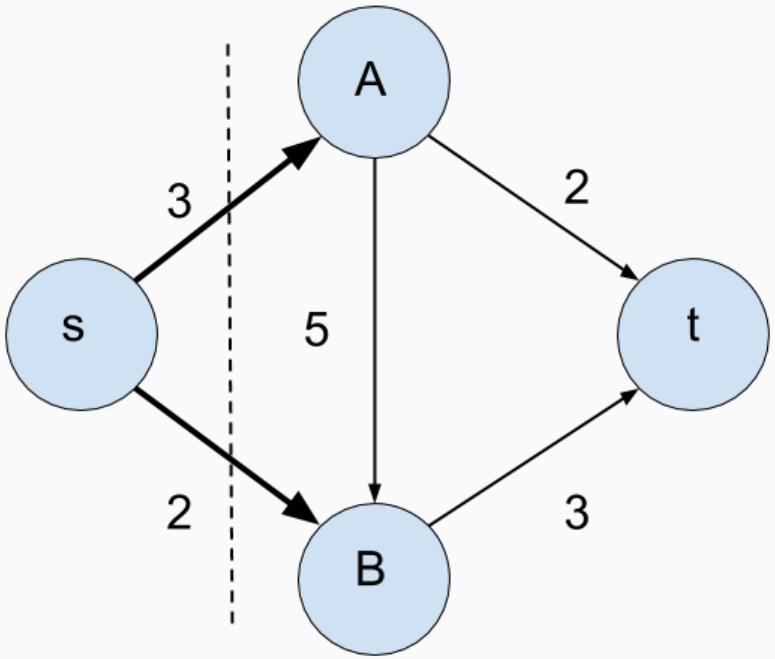
Max Flow - Min Cut Theorem

The maximum flow in a graph is equal to the capacity of the minimum cut.

Max Flow - Min Cut Theorem

The maximum flow in a graph is equal to the capacity of the minimum cut.

Here's a minimum cut from our running example:



S = {s} and T = {t,A,B}
Cap(S,T) = 2 + 3 = 5

Max Flow - Min Cut Theorem

The maximum flow in a graph is equal to the capacity of the minimum cut.

We will not go over the proof in detail but here are a few key things to keep in mind. At the end of the Ford-Fulkerson algorithm:

- There is no augmenting path from s to t.

Max Flow - Min Cut Theorem

The maximum flow in a graph is equal to the capacity of the minimum cut.

We will not go over the proof in detail but here are a few key things to keep in mind. At the end of the Ford-Fulkerson algorithm:

- There is no augmenting path from s to t.
- We have an $s - t$ cut which is a minimum capacity cut.

Max Flow - Min Cut Theorem

The maximum flow in a graph is equal to the capacity of the minimum cut.

We will not go over the proof in detail but here are a few key things to keep in mind. At the end of the Ford-Fulkerson algorithm:

- There is no augmenting path from s to t.
- We have an $s - t$ cut which is a minimum capacity cut.
- All edges going from S to T are saturated - they have 0 residual capacity.

Problem 1

You are given a flow network with a sources $s_1, s_2, \ldots, s_a$ and b sinks $t_1, t_2, \ldots, t_b$. Find the maximum flow.

Problem 1 - Solution

Just add a super source s and super sink t!

- Add edge (s, s_i) for all sources s_i with infinite capacity.
- Add edge (t_i, t) for all sinks t_i with infinite capacity.

Runtime: Still the same!

Next Class

Maximum Bipartite Matching