# CSE 421 Section 7

## Network Flows

# Administrivia

# Announcements & Reminders

- **Midterm exam**
  - Congrats on finishing half of the course!
  - We'll be grading it over the next several days.

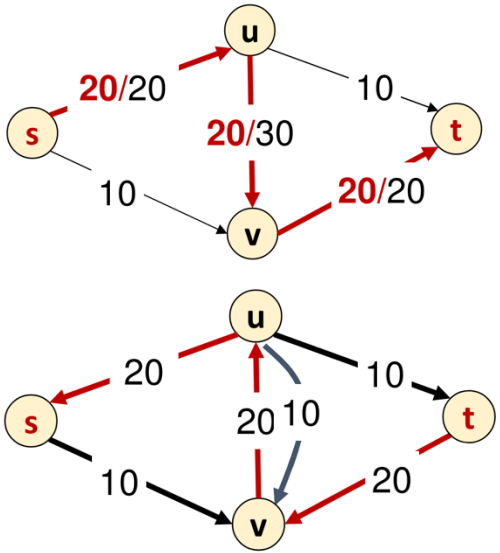
- **HW6**
  - Due Wednesday 11/13 @ 11:59pm

# Algorithms for network flows

# Algorithms for network flows

**Ford–Fulkerson** is a class of algorithms to compute maximum flow.

1. Let the residual graph $G_f$ be initialized to $G$.
2. While there exists an $s$-$t$ path $P$ in $G_f$,
   a. Let $c$ be the minimum capacity along this path.
   b. Update $f$ to push $c$ flow along $P$.
   c. Update edges in $G_f$ along $P$.




Warmup: How does the update work?

# Algorithms for network flows

**Ford–Fulkerson** is a class of algorithms to compute maximum flow.

- Edmonds–Karp implementation: BFS (unweighted shortest path) to select $s$-$t$ path

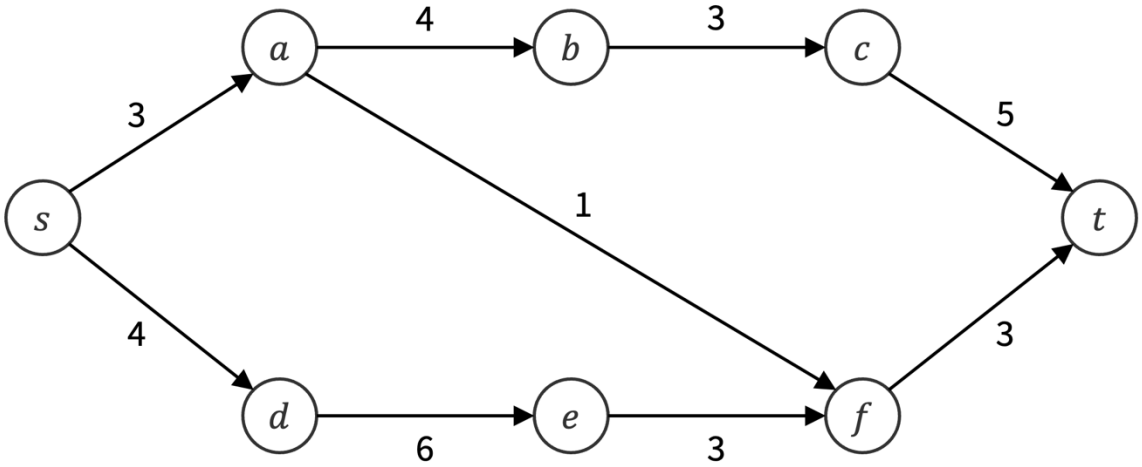
**Capacity scaling algorithm**: Process capacities one bit at a time

| **Ford–Fulkerson with BFS** | | **Capacity scaling** |
|---|---|---|
| **Ford–Fulkerson bound** | **Edmonds–Karp bound** | |
| $O(mnC)$ | $O(m^2n)$ | $O(m^2 \log C)$ |
| good when all capacities are small | good with many large capacities | good when there are a few large capacities |

# Flow algorithms practice

# Problem 1 – Flow algorithms practice

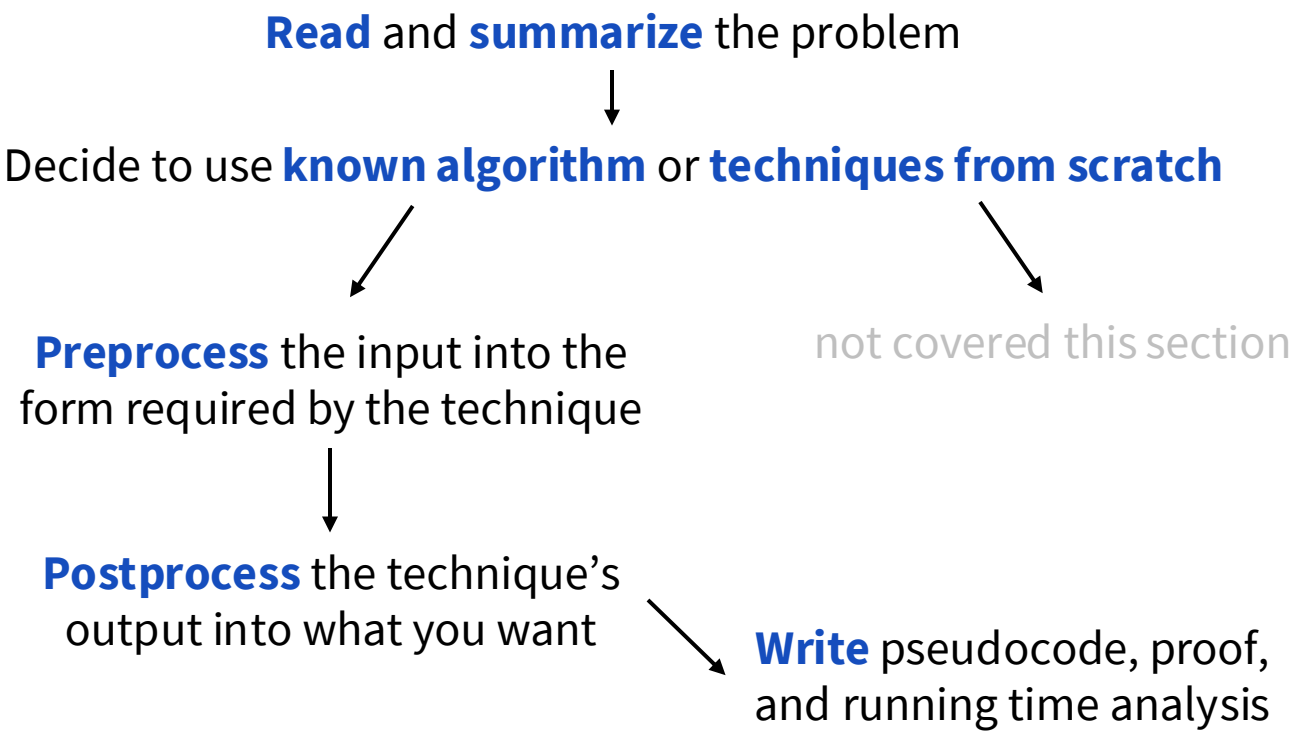
Using Ford–Fulkerson with BFS, find the maximum $s$-$t$ flow in the graph $G$ below, the corresponding residual graph, and minimum cut.




Work on this with the people around you, then we'll check!

# Problem solving with flows

# Problem solving strategy overview



# Three common preprocessing tricks

To preprocess for network flows:

- If you want “multiple sources/sinks,” add **dummy vertices**.
- If you want “vertex capacities,” **split vertices** into two.
- If you want “unconstrained capacity,” just **set capacity to infinity**.

We’ll see examples of each.

# Problem 2 – Reservoir balancing

You have a set of overfilled reservoirs $O = \{o_1, \ldots, o_m\}$ and a set of underfilled reservoirs $U = \{u_1, \ldots, u_n\}$, and want to move 10,000 gallons of water from reservoirs in $O$ to reservoirs in $U$. You only care about the total amount of water moved, not each individual reservoir. You have a directed graph $G = (V, E)$ describing the one-way pipes connecting the reservoirs, where $O \subseteq V$ and $U \subseteq V$. This graph may include intermediate reservoirs, whose water levels should not change through your solution. Each pipe $e \in E$ has an integer maximum rate of flow $c(e)$ in gallons per minute. Find a method to move the water in the shortest amount of time.

a) Write a summary of the problem.

Work on this with the people around you, then we'll check!

# Problem 2 – Reservoir balancing

a) Write a summary of the problem.

b) Think about the three tricks. Use them to preprocess our input into something suitable as input for a network flows algorithm.

- If you want “multiple sources/sinks,” add dummy vertices.
- If you want “vertex capacities,” split vertices into two.
- If you want “unconstrained capacity,” just set capacity to infinity.

# Problem 2 – Reservoir balancing

c) After running a max flow algorithm, what do you get? What postprocessing is needed to get the solution?

Work on this with the people around you, then we'll check!

# Network flows proofs

In a network flow problem, the main claim in your proof will probably be:

"The maximum flow in the graph that I constructed
is equal to the maximum solution in the original problem."

It should be intuitive, but to write thing formally, the strategy is to prove two things:

1. We can turn any **solution of our network** into a **solution of the original problem** of same quality.
2. We can turn any **solution of the original problem** into a **solution of our network** of same quality.

Then, your output works because (1), and no other solution of the original problem is better, by applying (2) and the fact that we found the max flow of our network.

# Problem 2 – Reservoir balancing

d) Prove your solution is correct.

Work on this with the people around you, then we'll check!

# Problem 2 – Reservoir balancing

e) Which flow algorithm is best for this problem? Then analyze your running time. (Assume $|V| = n$, $|E| = m$, and $n \leq m$.)

Work on this with the people around you, then we'll check!

# Problem 3 – Traffic modeling

In most cities, traffic congestion happens only at intersections – segments without intersections are free-flowing. An extremely rough model is that the capacity of an intersection (the total number of vehicles per hour that flow through the intersection in any direction) is proportional to the number of traffic lanes at the intersection. You are given the road network of a city as a graph $G = (V, E)$ (consisting of directed edges, i.e. one-way streets), as well as the number of lanes $c(v)$ at each intersection. Suppose each lane adds a capacity of 300 vehicles per hour, and there are no intersections with more than 12 lanes. Given an origin $s$ and destination $t$ (which also do have limited capacity), compute how many vehicles per hour can move from $s$ to $t$.

a) Write a summary of the problem.

Work on this with the people around you, then we'll check!

# Problem 3 – Traffic modeling

a) Write a summary of the problem.

b) Think about the three tricks. Use them to preprocess our input into something suitable as input for a network flows algorithm.

- If you want “multiple sources/sinks,” add dummy vertices.
- If you want “vertex capacities,” split vertices into two.
- If you want “unconstrained capacity,” just set capacity to infinity.

# Problem 3 – Traffic modeling

c) After running a max flow algorithm, what do you get? What postprocessing is needed to get the solution?

This should be quick – what can we do?

# Problem 3 – Traffic modeling

d) Prove your solution is correct.

Work on this with the people around you, then we'll check!

# Problem 3 – Traffic modeling

e) Which flow algorithm is best for this problem? Then analyze your running time. (Assume $|V| = n$, $|E| = m$, and $n \leq m$.)

Work on this with the people around you, then we'll check!

# Summary

- **Three tricks:** dummy $s/t$, split vertices for vertex capacity, and infinite capacity
- When using Ford–Fulkerson with BFS, pick **original FF bound if small capacities**, and pick **Edmonds–Karp bound if large capacities**.
- **Proof by converting solutions both ways**: between solutions to your constructed flow network and solutions to the original problem.

**Thanks for coming to section this week!**