Network Flows
Admin/Announcements

• CS Preregistration Info Session during Colloquium. Learn about:
  • courses offered next semester
  • major applications & forms
  • thesis applications & timelines
  • study abroad guidelines
• TAs and becoming a TA
  • Fill out TA feedback forms by Monday
  • Submit TA applications by April 21
• Williams Entrepreneurship Summit this Saturday!
Story So Far

• Algorithmic design paradigms:
  
  • **Greedy**: often simplest algorithms to design, but only work for certain limited class of optimization problems
    
    • A good initial thought for most problems but rarely optimal
  
  • **Divide and Conquer**
    
    • Solving a problem by breaking it down into smaller subproblems and (often) combining results
  
  • **Dynamic programming**
    
    • Recursion with memoization: avoiding repeated work
    
    • Trade space (memoization structure representation) for time (reuse stored results of repeated computations)
New Algorithmic Paradigm

• **Network flows** model a variety of optimization problems

• These optimization problems look complicated with lots of constraints
  • At first they may seem to have nothing to do with networks or flows!

• Very powerful problem solving frameworks

• We'll focus on the concept of **problem reductions**
  • Problem A reduces to B if a solution to B leads to a solution to A

• We'll learn how to prove that our reductions are correct
What’s a Flow Network?

- A flow network is a directed graph $G = (V, E)$ with a
  - A **source** is a vertex $s$ with in-degree 0
  - A **sink** is a vertex $t$ with out-degree 0
  - Each edge $e \in E$ has **edge capacity** $c(e) > 0$
Assumptions

• Assume that each node $v$ is on some $s$-$t$ path, that is, $s \rightsquigarrow v \rightsquigarrow t$ exists, for any vertex $v \in V$

  • Implies $G$ is connected and $m \geq n - 1$

• Assume capacities are positive integers

  • Will revisit this assumption & what happens otherwise

• Directed edge $(u, v)$ written as $u \rightarrow v$

• For simplifying expositions, we will sometimes write $c(u \rightarrow v) = 0$ when $(u, v) \notin E$
What’s a Flow?

- Given a flow network, an \((s, t)\)-flow or just flow (if source \(s\) and sink \(t\) are clear from context) \(f : E \rightarrow \mathbb{Z}^+\) satisfies the following two constraints:

  - **[Flow conservation]** \(f_{in}(v) = f_{out}(v), \text{ for } v \neq s, t\) where

    \[
    f_{in}(v) = \sum_{u} f(u \rightarrow v)
    \]
    \[
    f_{out}(v) = \sum_{w} f(v \rightarrow w)
    \]

  - To simplify, \(f(u \rightarrow v) = 0\) if there is no edge from \(u\) to \(v\)
Feasible Flow

- And second, a feasible flow must satisfy the capacity constraints of the network, that is,

\[
\text{[Capacity constraint]} \quad \text{for each } e \in E, 0 \leq f(e) \leq c(e)
\]
• **Definition.** The **value** of a flow $f$, written $v(f)$, is $f_{out}(s)$.

$$v(f) = 5 + 10 + 10 = 25$$
Value of a Flow

- **Definition.** The value of a flow $f$, written $v(f)$, is $f_{out}(s)$.

- **Lemma.** $f_{out}(s) = f_{in}(t)$

Intuitively, why do you think this is true?

value $= 5 + 10 + 10 = 25$
Value of a Flow

**Lemma.** \( f_{out}(s) = f_{in}(t) \)

**Proof.** Let \( f(E) = \sum_{e \in E} f(e) \)

• Then, \( \sum_{v \in V} f_{in}(v) = f(E) = \sum_{v \in V} f_{out}(v) \)

• For every \( v \neq s, t \) flow conservation implies \( f_{in}(v) = f_{out}(v) \)

• Thus all terms cancel out on both sides except \( f_{in}(s) + f_{in}(t) = f_{out}(s) + f_{out}(t) \)

• But \( f_{in}(s) = f_{out}(t) = 0 \)  ■
Value of a Flow

- **Lemma.** $f_{out}(s) = f_{in}(t)$
- **Corollary.** $v(f) = f_{in}(t)$.

\[ \text{value} = 5 + 10 + 10 = 25 \]
Max-Flow Problem

- **Problem.** Given an $s$-$t$ flow network, find a feasible $s$-$t$ flow of maximum value.
Minimum Cut Problem
Cuts are Back!

- Cuts in graphs played a key role when we were designing algorithms for MSTs.
- What is the definition of a cut?
Cuts in Flow Networks

- **Recall.** A cut \((S, T)\) in a graph is a partition of vertices such that \(S \cup T = V\), \(S \cap T = \emptyset\) and \(S, T\) are non-empty.

- **Definition.** An \((s, t)\)-cut is a cut \((S, T)\) s.t. \(s \in S\) and \(t \in T\).
Cut Capacity

- **Recall.** A cut \((S, T)\) in a graph is a partition of vertices such that \(S \cup T = V\), \(S \cap T = \emptyset\) and \(S, T\) are non-empty.

- **Definition.** An \((s, t)\)-cut is a cut \((S, T)\) s.t. \(s \in S\) and \(t \in T\).

- **Capacity** of a \((s, t)\)-cut \((S, T)\) is the sum of the capacities of edges leaving \(S\):

\[
c(S, T) = \sum_{v \in S, w \in T} c(v \rightarrow w)
\]
Quick Quiz

**Question.** What is the capacity of the $s$-$t$ cut given by grey and white nodes?

A. 11 \((20 + 25 - 8 - 11 - 9 - 6)\)

B. 34 \((8 + 11 + 9 + 6)\)

C. 45 \((20 + 25)\)

D. 79 \((20 + 25 + 8 + 11 + 9 + 6)\)

\[ c(S, T) = \sum_{v \in S, w \in T} c(v \to w) \]
Min Cut Problem

- **Problem.** Given an \( s-t \) flow network, find an \( s-t \) cut of minimum capacity.
Relationship between Flows and Cuts
Flows and Cuts

- Cuts represent "bottlenecks" in a flow network
- For any \((s, t)\)-cut, all flow needs to "exit" \(S\) to get to \(t\)
- We will formalize this intuition
Claim. Let $f$ be any $s$-$t$ flow and $(S, T)$ be any $s$-$t$ cut then $\nu(f) \leq c(S, T)$

- There are two $s$-$t$ cuts for which this is easy to see (which ones?)
Claim. Let $f$ be any $s$-$t$ flow and $(S, T)$ be any $s$-$t$ cut then $\nu(f) \leq c(S, T)$

- There are two $s$-$t$ cuts for which this is easy to see (which ones?)
Flows and Cuts

To prove this for any cut, we first relate the flow value in a network to the net flow leaving a cut

- **Lemma.** For any feasible $(s, t)$-flow $f$ on $G = (V, E)$ and any $(s, t)$-cut, $v(f) = f_{out}(S) - f_{in}(S)$, where

  \[
  f_{out}(S) = \sum_{v \in S, w \in T} f(v \rightarrow w) \quad \text{(sum of flow ‘leaving’ $S$)}
  \]

  \[
  f_{in}(S) = \sum_{v \in S, w \in T} f(w \rightarrow v) \quad \text{(sum of flow ‘entering’ $S$)}
  \]

- **Note:** $f_{out}(S) = f_{in}(T)$ and $f_{in}(S) = f_{out}(T)$
Flows and Cuts

Proof. \( f_{out}(S) - f_{in}(S) \)

\[
= \sum_{v \in S, w \in T} f(v \to w) - \sum_{v \in S, u \in T} f(u \to v) \quad \text{[by definition]}
\]

\[
= \left[ \sum_{v, w \in S} f(v \to w) - \sum_{v, u \in S} f(u \to v) \right] + \sum_{v \in S, w \in T} f(v \to w) - \sum_{v \in S, u \in T} f(u \to v)
\]

These are the same sum: they sum the flow of all edges with both vertices in \( S \)

Adding zero terms
Proof. \( f_{out}(S) - f_{in}(S) \)

\[
= \left[ \sum_{v,w \in S} f(v \to w) - \sum_{v,u \in S} f(u \to v) \right] + \sum_{v \in S, w \in T} f(v \to w) - \sum_{v \in S, u \in T} f(u \to v)
\]

\[
= \sum_{v,w \in S} f(v \to w) + \sum_{v \in S, w \in T} f(v \to w) - \sum_{v,u \in S} f(u \to v) - \sum_{v \in S, u \in T} f(u \to v)
\]

\[
= \sum_{v \in S} \left( \sum_{w} f(v \to w) - \sum_{u} f(u \to v) \right)
\]

\[
= \sum_{v \in S} f_{out}(v) - f_{in}(v)
\]

\[
= f_{out}(S) = v(f) \hspace{1cm} \blacksquare
\]
Flows and Cuts

- We use this result to prove that the value of a flow cannot exceed the capacity of any cut in the network.

- **Claim.** Let $f$ be any $s$-$t$ flow and $(S, T)$ be any $s$-$t$ cut then
  \[ v(f) \leq c(S, T) \]

- **Proof.**
  \[
  v(f) = f_{out}(S) - f_{in}(S) \\
  \leq f_{out}(S) = \sum_{v \in S, w \in T} f(v \to w) \\
  \leq \sum_{v \in S, w \in T} c(v, w) = c(S, T)
  \]

When is $v(f) = c(S, T)$?

- $f_{in}(S) = 0$, $f_{out}(S) = c(S, T)$
Max-Flow & Min-Cut

- Suppose the $c_{\text{min}}$ is the capacity of the minimum cut in a network
- What can we say about the feasible flow we can send through it
  - cannot be more than $c_{\text{min}}$
- In fact, whenever we find any $s$-$t$ flow $f$ and any $s$-$t$ cut $(S, T)$ such that, $\nu(f) = c(S, T)$ we can conclude that:
  - $f$ is the maximum flow, and,
  - $(S, T)$ is the minimum cut
- The question now is, given any flow network with min cut $c_{\text{min}}$, is it always possible to route a feasible $s$-$t$ flow $f$ with $\nu(f) = c_{\text{min}}$?
Max-Flow Min-Cut Theorem

There is a beautiful, powerful relationship between these two problems in given by the following theorem.

- **Theorem.** Given any flow network $G$, there exists a feasible $(s, t)$-flow $f$ and an $(s, t)$-cut $(S, T)$ such that,

$$\nu(f) = c(S, T)$$

- Informally, in a flow network, the **max-flow = min-cut**

- This will guide our algorithm design for finding max flow

- (Will prove this theorem by construction in a bit.)
Aside: Network Flow History

• In 1950s, US military researchers Harris and Ross wrote a classified report about the rail network linking Soviet Union and Eastern Europe
  • Vertices were the geographic regions
  • Edges were railway links between the regions
  • Edge weights were the rate at which material could be shipped from one region to next
• Ross and Harris determined:
  • Maximum amount of stuff that could be moved from Russia to Europe (max flow)
  • Cheapest way to disrupt the network by removing rail links (min cut)
Network Flow History

Fig. 7 — Traffic pattern: entire network available

Legend:
- International boundary
- Railway operating division

Capacity: 12 each way per day. Required flow of 9 per day toward destinations (in direction of arrow) with equivalent number of returning trains in opposite direction.

All capacities in √1000’s of tons each way per day.


Destinations: Divisions 3, 6, 9 (Poland); 10 (Czechoslovakia); and 2, 3 (Austria).

Alternative destinations: Germany or East Germany.

Note: IX at Division 9, Poland.
Ford-Fulkerson Algorithm
Towards a Max-Flow Algorithm

We will design a max-flow algorithm and show that there is a $s$-$t$ cut s.t. value of flow computed by algorithm $=$ capacity of cut

- Let's start with a greedy approach:
  - Pick an $s$-$t$ path and push as much flow as possible down it
  - Repeat until you get stuck

Note: This won't actually work, but it gives us a sense of what we need to keep track of to improve it
Towards a Max-Flow Algorithm

Greedy strategy:

- Start with $f(e) = 0$ for each edge
- Find an $s \sim t$ path $P$ where each edge has $f(e) < c(e)$
- “Augment” flow (as much as possible) along path $P$
- Repeat until you get stuck
- Let’s explore an example
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \sim t$ path $P$ where each edge has $f(e) < c(e)$
- “Augment” flow (as much as possible) along path $P$
- Repeat until you get stuck
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \leadsto t$ path $P$ where each edge has $f(e) < c(e)$
- “Augment” flow (as much as possible) along path $P$
- Repeat until you get stuck
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \rightsquigarrow t$ path $P$ where each edge has $f(e) < c(e)$
- “Augment” flow (as much as possible) along path $P$
- Repeat until you get stuck
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \sim t$ path $P$ where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path $P$
- Repeat until you get stuck
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \sim t$ path $P$ where each edge has $f(e) < c(e)$
- “Augment” flow (as much as possible) along path $P$
- Repeat until you get stuck
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \leadsto t$ path $P$ where each edge has $f(e) < c(e)$
- “Augment” flow (as much as possible) along path $P$
- Repeat until you get stuck
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \leadsto t$ path $P$ where each edge has $f(e) < c(e)$
- “Augment” flow (as much as possible) along path $P$
- Repeat until you get stuck

Is this the best we can do?

Ending flow value = 16
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \sim t$ path $P$ where each edge has $f(e) < c(e)$
- “Augment” flow (as much as possible) along path $P$
- Repeat until you get stuck

**ending flow value = 16**
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \sim t$ path $P$ where each edge has $f(e) < c(e)$
- “Augment” flow (as much as possible) along path $P$
- Repeat until you get stuck

ending flow value = 16
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \sim t$ path $P$ where each edge has $f(e) < c(e)$
- “Augment” flow (as much as possible) along path $P$
- Repeat until you get stuck

ending flow value = 16
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \sim t$ path $P$ where each edge has $f(e) < c(e)$
- “Augment” flow (as much as possible) along path $P$
- Repeat until you get stuck

**ending flow value = 16**
Towards a Max-Flow Algorithm

- Start with \( f(e) = 0 \) for each edge
- Find an \( s \sim t \) path \( P \) where each edge has \( f(e) < c(e) \)
- “Augment” flow (as much as possible) along path \( P \)
- Repeat until you get stuck

ending flow value = 16
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \sim t$ path $P$ where each edge has $f(e) < c(e)$
- “Augment” flow (as much as possible) along path $P$
- Repeat until you get stuck

**ending flow value = 16**
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \sim t$ path $P$ where each edge has $f(e) < c(e)$
- “Augment” flow (as much as possible) along path $P$
- Repeat until you get stuck

Ending flow value = 16
Towards a Max-Flow Algorithm

- Start with $f(e) = 0$ for each edge
- Find an $s \sim t$ path $P$ where each edge has $f(e) < c(e)$
- "Augment" flow (as much as possible) along path $P$
- Repeat until you get stuck

max-flow value = 19
Towards a Max-Flow Algorithm

- Start with \( f(e) = 0 \) for each edge
- Find an \( s \sim t \) path \( P \) where each edge has \( f(e) < c(e) \)
- “Augment” flow (as much as possible) along path \( P \)
- Repeat until you get stuck

**max-flow value = 19**
Why Greedy Fails

**Problem:** greedy can never “undo” a bad flow decision

- Consider the following flow network

```
+---+ 2  +---+ 2  +---+ 2
|   |     |   |     |   |
+---+ 2  +---+ 1  +---+
|   |     |   |     |   |
+---+     +---+     +---+
  s     2     w     t
```

- Greedy could choose $s \rightarrow v \rightarrow w \rightarrow t$ as first $P$

- **Takeaway:** Need a mechanism to “undo” bad flow decisions
Ford-Fulkerson Algorithm
Ford Fulkerson: Idea

**Goal**: Want to make “forward progress” while letting ourselves undo previous decisions if they’re getting in our way

- **Idea**: keep track of where we can push flow
  - Can push more flow along any edge with remaining capacity
  - Can also push flow “back” along any edge that already has flow down it (**undo** a previous flow push)
- We need a way to systematically track these decisions
Residual Graph

Given flow network $G = (V, E, c)$ and a feasible flow $f$ on $G$, the residual graph $G_f = (V, E_f, c_f)$ is defined as follows:

- Vertices in $G_f$ same as $G$

- (Forward edge) For $e \in E$ with residual capacity $c(e) - f(e) > 0$, create $e \in E_f$ with capacity $c(e) - f(e)$

- (Backward edge) For $e \in E$ with $f(e) > 0$, create $e_{reverse} \in E_f$ with capacity $f(e)$
Flow Algorithm Idea

• Now we have a residual graph that lets us make forward progress or push back existing flow

• We will look for $s \sim t$ paths in $G_f$ rather than $G$

• Once we have a path, we will "augment" flow along it similar to greedy
  
  • find bottleneck capacity edge on the path and push that much flow through it in $G_f$

• When we translate this back to $G$, this means:
  
  • We increment existing flow on a forward edge
  
  • Or we decrement flow on a backward edge
Augmenting Path & Flow

- An **augmenting path** $P$ is a **simple** $s \rightsquigarrow t$ path in the residual graph $G_f$

- The **bottleneck capacity** $b$ of an augmenting path $P$ is the minimum capacity of any edge in $P$.

```
AUGMENT($f$, $P$)

$b \leftarrow$ bottleneck capacity of augmenting path $P$.

FOREACH edge $e \in P$

    IF ($e \in E$, that is, $e$ is forward edge)

        Increase $f(e)$ in $G$ by $b$

    ELSE

        Decrease $f(e)$ in $G$ by $b$

RETURN $f$.
```
Ford-Fulkerson Algorithm

- Start with \( f(e) = 0 \) for each edge \( e \in E \)
- Find a simple \( s \rightarrow t \) path \( P \) in the residual network \( G_f \)
- Augment flow along path \( P \) by bottleneck capacity \( b \)
- Repeat until you get stuck

\[
\text{FORD–FULKERSON}(G)
\]

\begin{align*}
\text{FOREACH} & \quad \text{edge} \ e \in E : \ f(e) \leftarrow 0. \\
\text{G}_f & \leftarrow \text{residual network of} \ G \ \text{with respect to} \ \text{flow} \ f. \\
\text{WHILE} & \quad (\text{there exists an} \ s \rightarrow t \ \text{path} \ P \ \text{in} \ G_f) \\
& \qquad \quad f \leftarrow \text{AUGMENT}(f, P). \\
& \quad \text{Update} \ G_f. \\
\text{RETURN} & \quad f.
\end{align*}
Ford-Fulkerson Example

network $G$ and flow $f$

residual network $G_f$
Ford-Fulkerson Example

network $G$ and flow $f$

flow capacity

value of flow 0

$P$ in residual network $G_f$
Ford-Fulkerson Example

network $G$ and flow $f$

residual network $G_f$
Ford-Fulkerson Example

network $G$ and flow $f$

value of flow $8$

$P$ in residual network $G_f$
Ford-Fulkerson Example

network $G$ and flow $f$

residual network $G_f$
Ford-Fulkerson Example

network $G$ and flow $f$

value of flow $10$

$P$ in residual network $G_f$
Ford-Fulkerson Example

network $G$ and flow $f$

residual network $G_f$

value of flow

$10 + 6 = 16$
Ford-Fulkerson Example

network $G$ and flow $f$

P in residual network $G_f$

flow capacity

value of flow 16

fixes mistake from second augmenting path
Ford-Fulkerson Example

network $G$ and flow $f$

residual network $G_f$
Ford-Fulkerson Example

network $G$ and flow $f$

$P$ in residual network $G_f$
Ford-Fulkerson Example

network $G$ and flow $f$

residual network $G_f$

No s-t path left!
Ford-Fulkerson Example

network $G$ and flow $f$

capacity

flow

Capacity of cut?

residual network $G_f$

nodes reachable from $s$

value of flow $= 19$

No s-t path left!
Analysis: Ford-Fulkerson
Analysis Outline

- Feasibility and value of flow:
  - Show that each time we update the flow, we are routing a feasible $s$-$t$ flow through the network
  - And that value of this flow increases each time by that amount
- Optimality:
  - Final value of flow is the maximum possible
- Running time:
  - How long does it take for the algorithm to terminate?
- Space:
  - How much total space are we using?
Feasibility of Flow

- **Claim.** Let $f$ be a feasible flow in $G$ and let $P$ be an augmenting path in $G_f$ with bottleneck capacity $b$. Let $f' \leftarrow \text{AUGMENT}(f, P)$, then $f'$ is a feasible flow.

- **Proof.** Only need to verify constraints on the edges of $P$ (since $f' = f$ for other edges). Let $e = (u, v) \in P$
  
  - If $e$ is a forward edge: $f'(e) = f(e) + b$
    
    $\leq f(e) + (c(e) - f(e)) = c(e)$
  
  - If $e$ is a backward edge: $f'(e) = f(e) - b$
    
    $\geq f(e) - f(e) = 0$

- Conservation constraint hold on any node in $u \in P$:
  
  - $f_{in}(u) = f_{out}(u)$, therefore $f'_{in}(u) = f'_{out}(u)$ for both cases
Value of Flow: Making Progress

• **Claim.** Let \( f \) be a feasible flow in \( G \) and let \( P \) be an augmenting path in \( G_f \) with bottleneck capacity \( b \). Let \( f' \leftarrow \text{AUGMENT}(f, P) \), then \( v(f') = v(f) + b \).

• **Proof.**
  * First edge \( e \in P \) must be out of \( s \) in \( G_f \)
  * \((P \text{ is simple so never visits } s \text{ again})\
  * \( e \text{ must be a forward edge } (P \text{ is a path from } s \text{ to } t)\
  * Thus \( f(e) \text{ increases by } b \), increasing \( v(f) \text{ by } b \)

• **Note.** Means the algorithm makes forward progress each time!
Optimality
Ford-Fulkerson Optimality

- **Recall**: If \( f \) is any feasible \( s-t \) flow and \((S, T)\) is any \( s-t \) cut then \( \nu(f) \leq c(S, T) \).

- We will show that the Ford-Fulkerson algorithm terminates in a flow that achieves optimality, that is,

- Ford-Fulkerson finds a flow \( f^* \) and there exists a cut \((S^*, T^*)\) such that, \( \nu(f^*) = c(S^*, T^*) \)

- Proving this shows that it finds the maximum flow (and the min cut)

- This also **proves the max-flow min-cut theorem**
Ford-Fulkerson Optimality

- **Lemma.** Let $f$ be an $s$-$t$ flow in $G$ such that there is no augmenting path in the residual graph $G_f$, then there exists a cut $(S^*, T^*)$ such that $\nu(f) = c(S^*, T^*)$.

- **Proof.**
  - Let $S^* = \{v \mid v$ is reachable from $s$ in $G_f\}$, $T^* = V - S^*$
  - Is this an $s$-$t$ cut?
    - $s \in S$, $t \in T$, $S \cup T = V$ and $S \cap T = \emptyset$
  - Consider an edge $e = u \rightarrow v$ with $u \in S^*$, $v \in T^*$, then what can we say about $f(e)$?
Recall: Ford-Fulkerson Example

network $G$ and flow $f$

residual network $G_f$

Capacity of cut?

nodes reachable from $s$

No $s$-$t$ path left!
Ford-Fulkerson Optimality

- **Lemma.** Let \( f \) be a \( s-t \) flow in \( G \) such that there is no augmenting path in the residual graph \( G_f \), then there exists a cut \((S^*, T^*)\) such that \( \nu(f) = c(S^*, T^*) \).

- **Proof.**
  - Let \( S^* = \{ v \mid v \text{ is reachable from } s \text{ in } G_f \} \), \( T^* = V - S^* \)
  - Is this an \( s-t \) cut?
    - \( s \in S, t \in T, S \cup T = V \) and \( S \cap T = \emptyset \)
  - Consider an edge \( e = u \rightarrow v \) with \( u \in S^*, v \in T^* \), then what can we say about \( f(e) \)?
    - \( f(e) = c(e) \)
Ford-Fulkerson Optimality

- **Lemma.** Let $f$ be a $s$-$t$ flow in $G$ such that there is no augmenting path in the residual graph $G_f$, then there exists a cut $(S^*, T^*)$ such that $\nu(f) = c(S^*, T^*)$.

- **Proof. (Cont.)**
  - Let $S^* = \{v \mid v$ is reachable from $s$ in $G_f\}$, $T^* = V - S^*$
  - Is this an $s$-$t$ cut?
    - $s \in S, t \in T, S \cup T = V$ and $S \cap T = \emptyset$
  - Consider an edge $e = w \rightarrow v$ with $v \in S^*, w \in T^*$, then what can we say about $f(e)$?
Recall: Ford-Fulkerson Example

network $G$ and flow $f$

residual network $G_f$

Capacity of cut?

value of flow

No s-t path left!
Ford-Fulkerson Optimality

- **Lemma.** Let $f$ be a $s$-$t$ flow in $G$ such that there is no augmenting path in the residual graph $G_f$, then there exists a cut $(S^*, T^*)$ such that $\nu(f) = c(S^*, T^*)$.

- **Proof.** (Cont.)

- Let $S^* = \{v \mid v$ is reachable from $s$ in $G_f\}$, $T^* = V - S^*$

- Is this an $s$-$t$ cut?

  - $s \in S$, $t \in T$, $S \cup T = V$ and $S \cap T = \emptyset$

- Consider an edge $e = w \rightarrow v$ with $v \in S^*$, $w \in T^*$, then what can we say about $f(e)$?

  - $f(e) = 0$

  Otherwise, there would have been a backwards edge in the residual graph
Ford-Fulkerson Optimality

- **Lemma.** Let $f$ be a $s$-$t$ flow in $G$ such that there is no augmenting path in the residual graph $G_f$, then there exists a cut $(S^*, T^*)$ such that $v(f) = c(S^*, T^*)$.

- **Proof. (Cont.)**
  - Let $S^* = \{v \mid v$ is reachable from $s$ in $G_f\}$, $T^* = V - S^*$
  - Thus, all edges leaving $S^*$ are completely saturated and all edges entering $S^*$ have zero flow

- $v(f) = f_{out}(S^*) - f_{in}(S^*) = f_{out}(S^*) = c(S^*, T^*)$ $\blacksquare$

- **Corollary.** Ford-Fulkerson returns the maximum flow.
Ford-Fulkerson Algorithm

Running Time
Ford-Fulkerson Performance

\[\text{FORD–FULKERSON}(G)\]

\text{FOREACH edge } e \in E: f(e) \leftarrow 0.

\(G_f \leftarrow \text{residual network of } G \text{ with respect to flow } f.\)

\textbf{WHILE} (there exists an } s \rightarrow t \text{ path } P \text{ in } G_f)\textbf{ }

\(f \leftarrow \text{AUGMENT}(f, P).\)

\text{Update } G_f.

\textbf{RETURN } f.

• Does the algorithm terminate?

• Can we bound the number of iterations it does?

• Running time?
Ford-Fulkerson Running Time

- Recall we proved that with each call to AUGMENT, we increase the **value of flow** by $b = \text{bottleneck}(G_f, P)$

- **Assumption.** Suppose all capacities $c(e)$ are integers.

- **Integrality invariant.** Throughout Ford–Fulkerson, every edge flow $f(e)$ and corresponding residual capacity is an integer. Thus $b \geq 1$.

- Let $C = \max_u c(s \rightarrow u)$ be the maximum capacity among edges leaving the source $s$.

- It must be that $v(f) \leq (n - 1)C$

- Since, $v(f)$ increases by $b \geq 1$ in each iteration, it follows that FF algorithm terminates in at most $v(f) = O(nC)$ iterations.
Ford-Fulkerson Performance

\begin{itemize}
  \item Operations in each iteration?
    \begin{itemize}
    \item Find an augmenting path in $G_f$
    \item Augment flow on path
    \item Update $G_f$
    \end{itemize}
\end{itemize}
Ford-Fulkerson Running Time

- **Claim.** Ford-Fulkerson can be implemented to run in time $O(nmC)$, where $m = |E| \geq n - 1$ and $C = \max_{u} c(s \to u)$.

- **Proof.** Time taken by each iteration:
  - Finding an augmenting path in $G_f$
    - $G_f$ has at most $2m$ edges, using BFS/DFS takes $O(m + n) = O(m)$ time
  - Augmenting flow in $P$ takes $O(n)$ time
  - Given new flow, we can build new residual graph in $O(m)$ time
  - Overall, $O(m)$ time per iteration $\blacksquare$
Acknowledgments

- Some of the material in these slides are taken from
  - Jeff Erickson’s Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)
  - Shikha Singh