COMP 382: Reasoning about Algorithms

Max Flows and Min Cuts in Graphs

Prof. Maryam Aliakbarpour

co-instructors: Prof. Anjum Chida & Prof. Konstantinos Mamouras

November 4, 2025

Today's Lecture

1. Max Flows and Min Cuts in Graphs

Reading:

- Chapter 10 of the *Algorithms* book [Erickson, 2019]

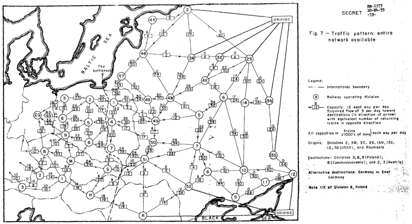
Content adapted from the same chapter in [Erickson, 2019].

1. Max Flows and Min Cuts in Graphs

A Cold War Problem: The Birth of Network Flow

In the mid-1950s, the U.S. Air Force's analysis of the Soviet railway system gave rise to two fundamental questions:

1. What is the maximum amount of material the Soviets can ship from their interior to Eastern Europe?
2. What are the critical bottlenecks? i.e., how can we disrupt the network with minimum effort?



Modeling the Network

Researchers Theodore Harris and Frank Ross modeled the vast rail system as a **flow network graph**:

- **Nodes:** Represented geographic regions or cities.

Modeling the Network

Researchers Theodore Harris and Frank Ross modeled the vast rail system as a **flow network graph**:

- **Nodes:** Represented geographic regions or cities.
- **Edges:** Represented the railway links connecting the regions.

Modeling the Network

Researchers Theodore Harris and Frank Ross modeled the vast rail system as a **flow network graph**:

- **Nodes:** Represented geographic regions or cities.
- **Edges:** Represented the railway links connecting the regions.
- **Edge Capacity (Weight):** Represented the maximum rate of material flow (e.g., tons per day) along a rail link.

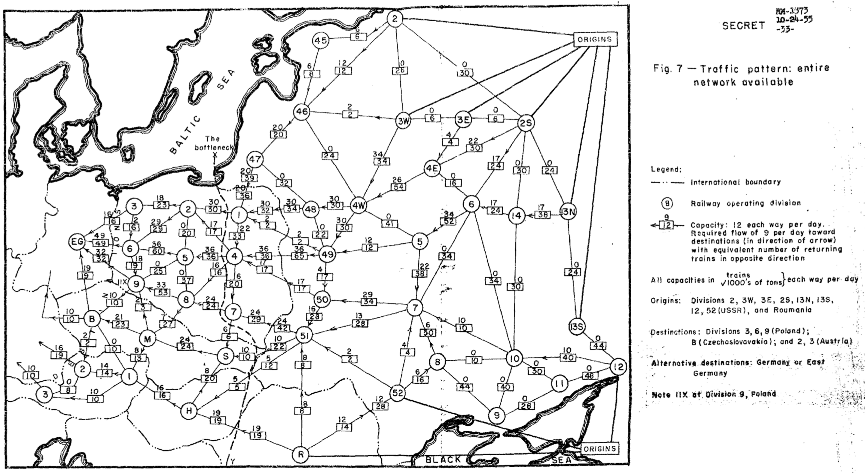


Figure: Harris and Ross's map of the Warsaw Pact rail network, declassified in 1999. Adapted from [Erickson, 2019].

The Two Big Questions

This model led to two fundamental questions:

1. **Maximum Flow:** What is the absolute maximum amount of supplies that can be moved from a **source** (Russia) to a **sink** (Europe) through the entire network at any given time?

The Two Big Questions

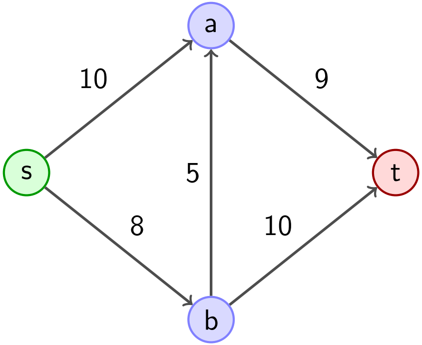
This model led to two fundamental questions:

1. **Maximum Flow:** What is the absolute maximum amount of supplies that can be moved from a **source** (Russia) to a **sink** (Europe) through the entire network at any given time?
2. **The Bottleneck (Minimum Cut):** What is the cheapest way to disrupt the network? This involves finding the set of links with the smallest combined capacity that, if removed, would completely sever the connection from source to sink.

What Is a Flow Network?

A flow network is a directed graph $G = (V, E)$ that models the flow of "stuff" from a source to a destination. Think of it like water pipes.

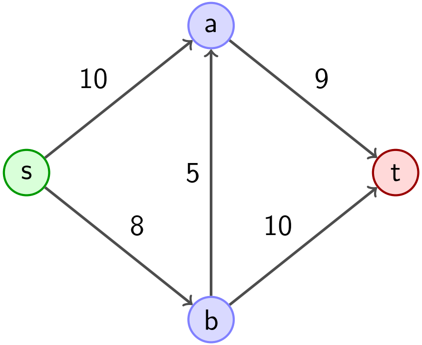
- We have a single **source** where the flow originates.



What Is a Flow Network?

A flow network is a directed graph $G = (V, E)$ that models the flow of "stuff" from a source to a destination. Think of it like water pipes.

- We have a single **source** where the flow originates.
- We have a single **sink** (or destination) where the flow terminates.

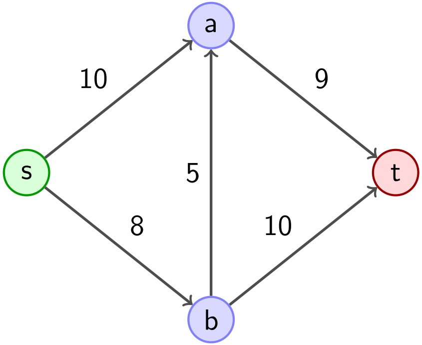


What Is a Flow Network?

A flow network is a directed graph $G = (V, E)$ that models the flow of "stuff" from a source to a destination. Think of it like water pipes.

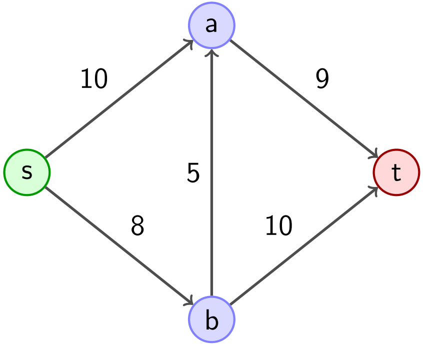
- We have a single **source** where the flow originates.
- We have a single **sink** (or destination) where the flow terminates.
- Each edge has a maximum **capacity** denoted by c:

$$c(b, a) = 5$$



What Is a Flow?

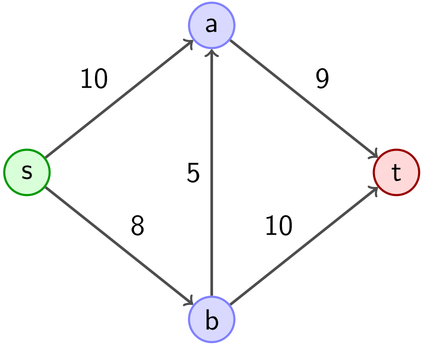
A **flow** f is an assignment of a real number to each edge $(u, v) \in E$ in the network, representing the rate at which material moves from u to v.



What Is a Flow?

A **flow** f is an assignment of a real number to each edge $(u, v) \in E$ in the network, representing the rate at which material moves from u to v.

Sending two units of flow from s to t via:
$s \to a \to t$



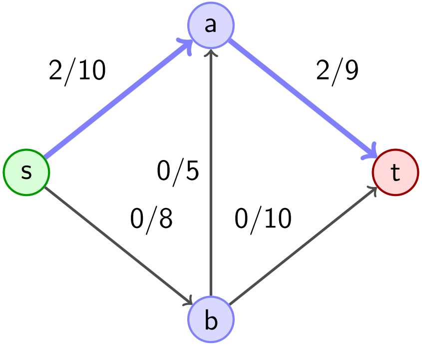
What Is a Flow?

A **flow** f is an assignment of a real number to each edge $(u, v) \in E$ in the network, representing the rate at which material moves from u to v.

Sending two units of flow from s to t via:
$s \rightarrow a \rightarrow t$

f(s,a) $=$ 2, f(a,t) $=$ 2

f(s,b) $=$ 0, f(b, a) $=$ 0, f(b, t) $=$ 0.



What Is a Flow?

A **flow** f is an assignment of a real number to each edge $(u, v) \in E$ in the network, representing the rate at which material moves from u to v.

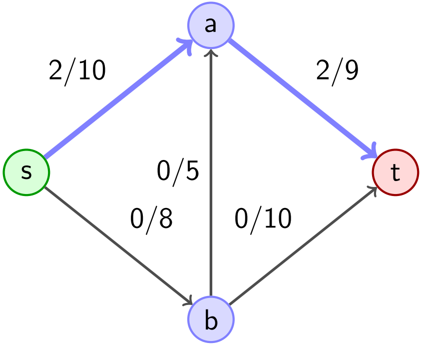
Sending two units of flow from s to t via:
$s \to a \to t$

f(s,a) $=$ 2, f(a,t) $=$ 2

f(s,b) $=$ 0, f(b, a) $=$ 0, f(b, t) $=$ 0.

Size of flow:

$$|f| = 2$$



What Is a Valid Flow?

For f to be a valid flow, it must obey two main rules:

1. **Capacity Constraint:** The flow through any edge cannot exceed its capacity

$$\forall (u, v) \in E : \qquad f(u, v) \leq c(u, v)$$

What Is a Valid Flow?

For f to be a valid flow, it must obey two main rules:

1. **Capacity Constraint:** The flow through any edge cannot exceed its capacity

$$\forall (u,v) \in E : \qquad f(u,v) \leq c(u,v)$$

2. **Flow Conservation:** For every node except the source and sink, the total flow entering the node must equal the total flow leaving it.

$$\forall\ v \in V \setminus \{s,t\} : \qquad \underbrace{\sum_{u \in IN(v)} f(u,v)}_{\text{incoming flow to } v} - \underbrace{\sum_{u \in OUT(v)} f(v,u)}_{\text{outgoing flow from } v} = 0$$

The Maximum Flow Problem

The goal: What is the maximum amount of “stuff” we can send from the source node **s** to the sink node **t**?

The Maximum Flow Problem

The goal: What is the maximum amount of "stuff" we can send from the source node **s** to the sink node **t**?

The maximum flow problem is to find a valid flow f that maximizes this value.

$$\max_{\text{valid } f} |f| \ .$$

Here, we define the **value** or the **size of a flow**, denoted by $|f|$, as the total net flow leaving the source node s, or going to the sink node t.

$$|f| := \underbrace{\sum_{u \in OUT(s)} f(s,u)}_{\text{outgoing flow from } s} - \underbrace{\sum_{u \in IN(s)} f(u,s)}_{\text{incoming flow to } s}$$

The Maximum Flow Problem

The goal: What is the maximum amount of "stuff" we can send from the source node **s** to the sink node **t**?

The maximum flow problem is to find a valid flow f that maximizes this value.

$$\max_{\text{valid } f} |f| \; .$$

Here, we define the **value** or the **size of a flow**, denoted by $|f|$, as the total net flow leaving the source node s, or going to the sink node t.

$$|f| := \underbrace{\sum_{u \in OUT(s)} f(s,u)}_{\text{outgoing flow from } s} - \underbrace{\sum_{u \in IN(s)} f(u,s)}_{\text{incoming flow to } s} = \underbrace{\sum_{u \in IN(t)} f(u,t)}_{\text{incoming flow to } t} - \underbrace{\sum_{u \in OUT(t)} f(t,u)}_{\text{outgoing flow from } t}$$

The Ford-Fulkerson Algorithm

It's an iterative method with a simple, greedy approach:

1. Start with zero flow everywhere.
2. Find a path from **s** to **t** that has available capacity. This is called an *augmenting path*.
3. Push as much flow as possible along this path. The amount is limited by the "bottleneck" edge on the path.
4. Update the network to reflect the new flow.
5. **Repeat** until no more augmenting paths can be found.

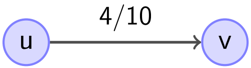
But how do we "update" the network? And what if we make a bad choice for a path?

Residual Graphs: The Key to “Undoing” Bad Choices

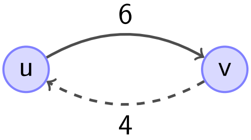
The *residual graph* shows us how much *additional* flow we can push on any edge.

For an edge (u, v) with capacity $c(u, v)$ and current flow $f(u, v)$:

- The *forward edge* (u, v) in the residual graph has capacity $c(u, v) - f(u, v)$. This is the remaining capacity.
- We add a *backward edge* (v, u) with capacity equal to the current flow $f(u, v)$. This represents our ability to “cancel” or “push back” flow.



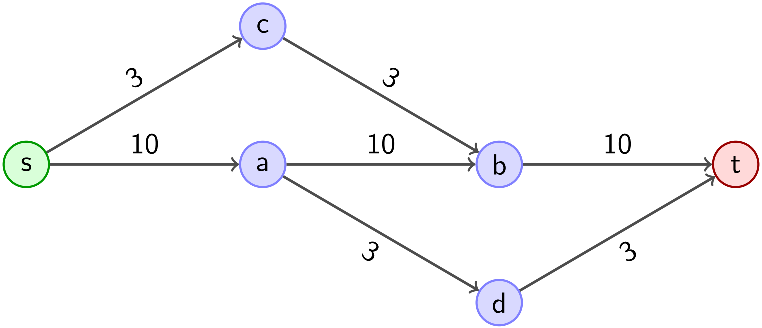
Original Graph with Flow



Residual Graph

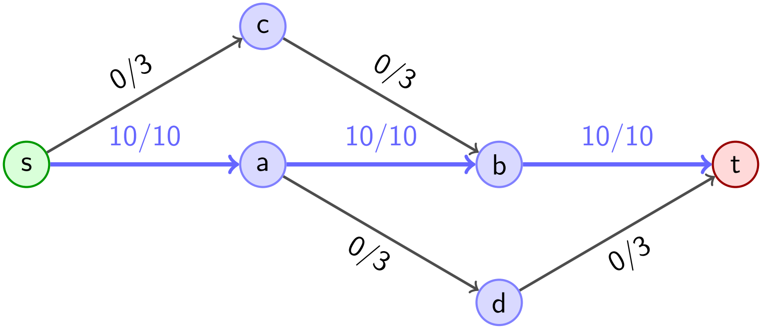
Why Backward Edges Work: Rerouting Flow

A greedy path choice can be a mistake. Committing to an early, seemingly good path might block other paths needed to achieve the true maximum flow.



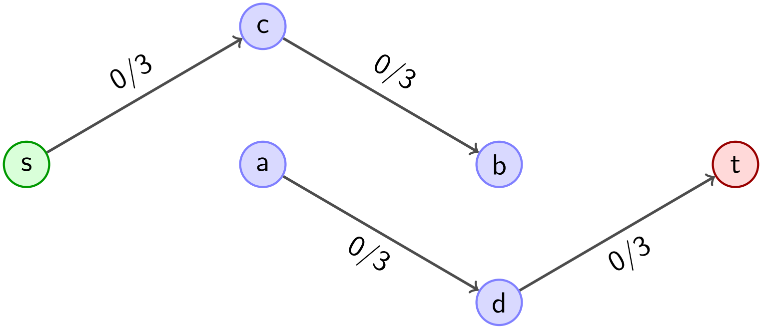
Why Backward Edges Work: Rerouting Flow

A greedy path choice can be a mistake. Committing to an early, seemingly good path might block other paths needed to achieve the true maximum flow.



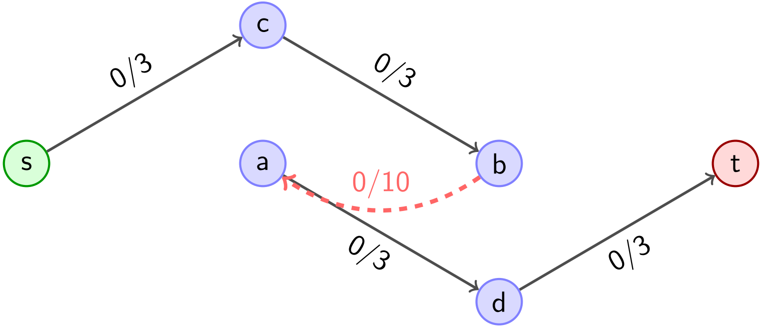
Why Backward Edges Work: Rerouting Flow

A greedy path choice can be a mistake. Committing to an early, seemingly good path might block other paths needed to achieve the true maximum flow.



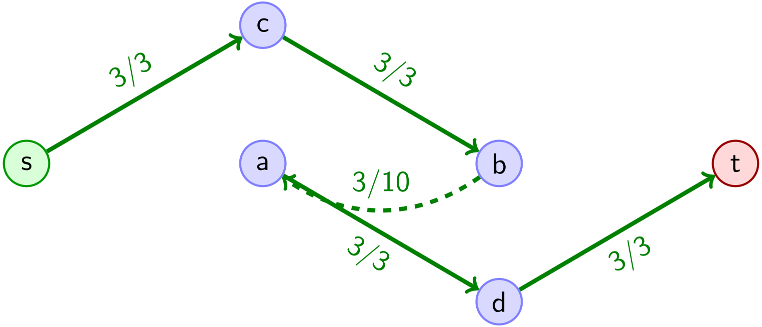
Why Backward Edges Work: Rerouting Flow

Allowing flow to be "pushed back" along an edge corrects our earlier suboptimal path choices. Pushing flow along the backward edge (b, a) decreases the flow on the original edge (a, b).



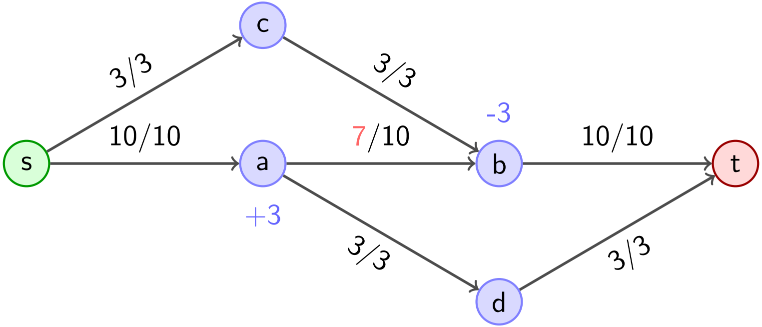
Why Backward Edges Work: Rerouting Flow

Allowing flow to be "pushed back" along an edge corrects our earlier suboptimal path choices. Pushing flow along the backward edge (b, a) decreases the flow on the original edge (a, b).



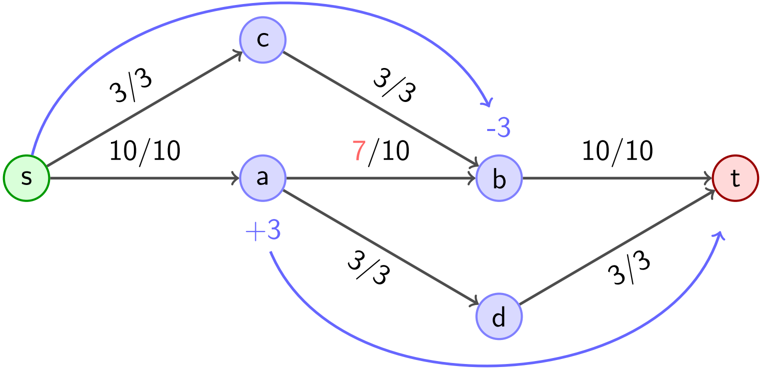
Why Backward Edges Work: Rerouting Flow

This frees up flow at a to continue toward the sink, while the new path takes over the job of supplying flow to node b.



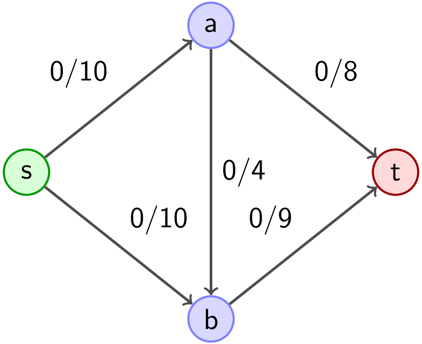
Why Backward Edges Work: Rerouting Flow

This frees up flow at a to continue toward the sink, while the new path takes over the job of supplying flow to node b.



Ford-Fulkerson: An Example

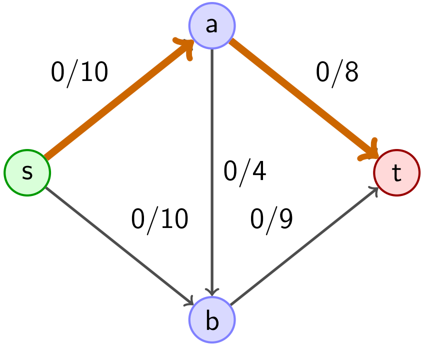
Let's find the max flow for this network. Total flow so far: **0**.



Example: Augmenting Path 1

Path: $s \to a \to t$. **Bottleneck:** $\min(10, 8) = \mathbf{8}$.

Augmenting Path



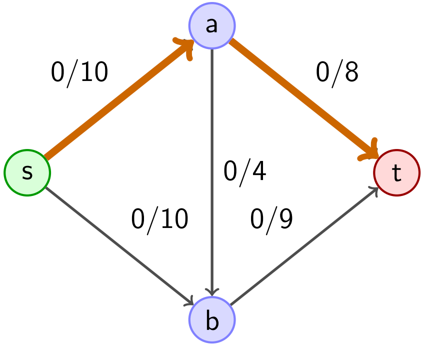
We push 8 units of flow.

Example: Augmenting Path 1

Path: $s \to a \to t$.

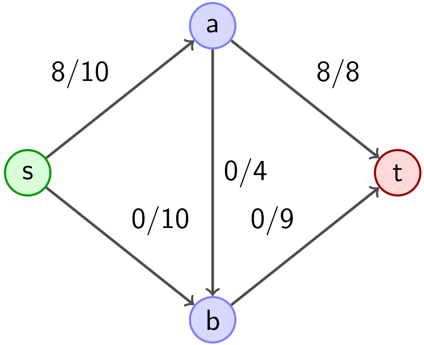
Bottleneck: $\min(10, 8) = \mathbf{8}$.

Augmenting Path



We push 8 units of flow.

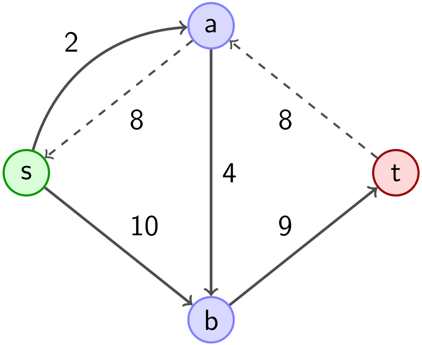
Updated Flow



New total flow: **8**.

Example: Augmenting Path 2

Residual Graph Path

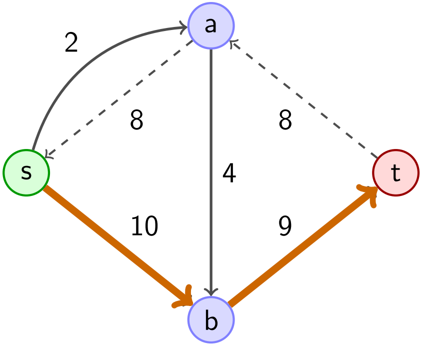


Example: Augmenting Path 2

Path: $s \to b \to t$.

Bottleneck: $\min(10, 9) = \mathbf{9}$.

Residual Graph Path

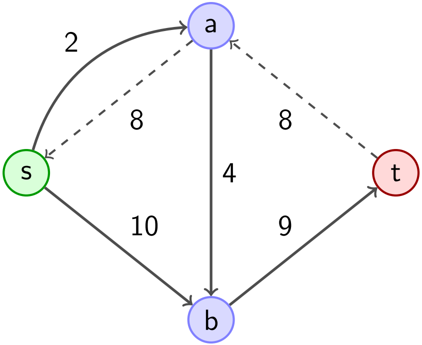


We push 9 units of flow.

Example: Augmenting Path 2

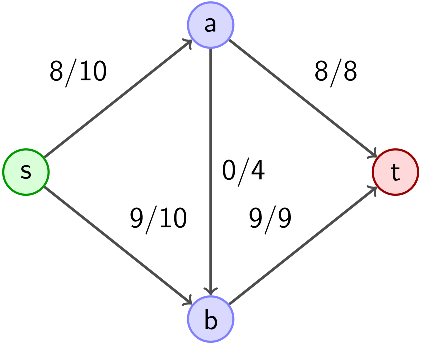
Path: $s \to b \to t$. **Bottleneck:** $\min(10, 9) = \mathbf{9}$.

Residual Graph Path



We push 9 units of flow.

Updated Flow

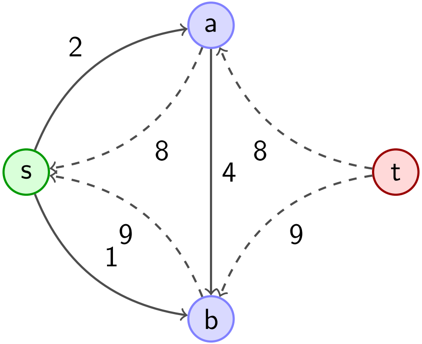


New total flow: $8 + 9 = \mathbf{17}$.

Example: Searching for another Path

After pushing 17 units of flow, let's search for another path in the residual graph.

Final Residual Graph



In the residual graph, there is no incoming edge to t.

No more augmenting paths can be found.

Example: No More Paths Found

Since there is no path from **s** to **t** in the final residual graph, the algorithm terminates.

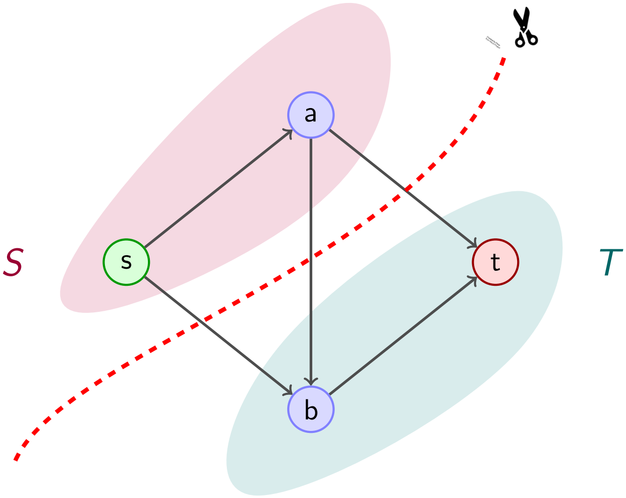
The total flow is the sum of the flows sent along the augmenting paths found:

$$\textbf{Maximum Flow} = \mathbf{17}$$

This result is guaranteed to be the maximum by the max-flow min-cut theorem.

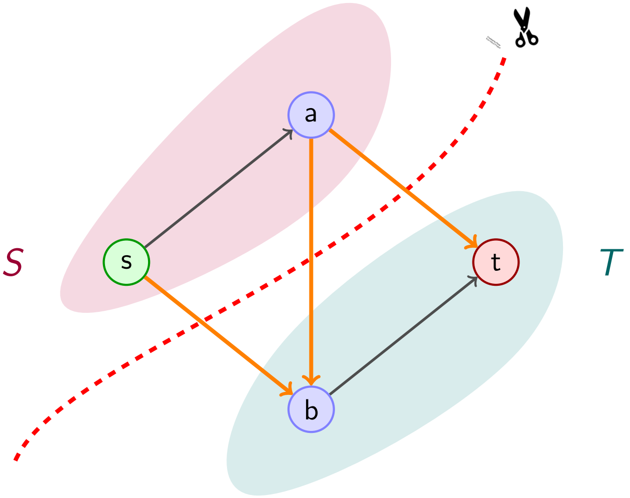
What Is an s-t Cut?

- An *s-t cut* is a **partition** of the vertices into two sets, S and T, such that the source $s \in S$ and the sink $t \in T$.



What Is an s-t Cut?

- An *s-t cut* is a **partition** of the vertices into two sets, S and T, such that the source $s \in S$ and the sink $t \in T$.
- The **capacity** (or the **size**) of the cut is the sum of capacities of all edges going from a node in S to a node in T.

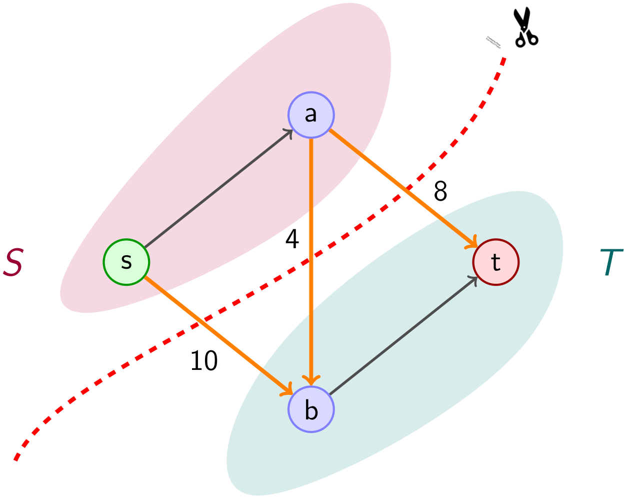


What Is an s-t Cut?

- An *s-t cut* is a **partition** of the vertices into two sets, S and T, such that the source $s \in S$ and the sink $t \in T$.
- The **capacity** (or the **size**) of the cut is the sum of capacities of all edges going from a node in S to a node in T.

The capacity is the sum of edges crossing from S to T (orange):

$$10 + 8 + 4 = 22.$$



The Minimum Cut Problem

The goal: To find an s-t cut (S, T) with the minimum possible capacity.

Mathematically, we want to find:

$$\min_{\text{all s-t cuts } (S,T)} (\text{capacity}(S, T)) = \min_{\text{all s-t cuts } (S,T)} \left(\sum_{u \in S, v \in T} c(u, v) \right)$$

The Max-Flow Min-Cut Theorem

Theorem

In any flow network, the value of the maximum (s,t)-flow is equal to the capacity of the minimum (s,t)-cut.

$$\max_{\text{flows } f} |f| = \min_{\text{cuts } (S,T)} \text{capacity}(S,T)$$

The Max-Flow Min-Cut Theorem

Theorem

In any flow network, the value of the maximum (s, t)-flow is equal to the capacity of the minimum (s, t)-cut.

$$\max_{\text{flows } f} |f| = \min_{\text{cuts } (S,T)} \text{capacity}(S, T)$$

Main Lemma: Weak Duality

Lemma (Weak Duality Lemma)

For any feasible (s,t)-flow f and any (s,t)-cut (S,T), the value of the flow is at most the capacity of the cut.

$$|f| \leq \mathit{capacity}(S,T)$$

Important implication

$$\text{Weak duality} \quad \Rightarrow \quad \text{max flow} \leq \text{min cut}$$

Completing the Max-Flow Min-Cut Proof

To prove the theorem, we will take two final steps:

- **Step 1:** We prove the weak duality lemma. This establishes that

$$\max \text{flow} \leq \min \text{cut}\,.$$

- **Step 2:** For a maximum flow f^*, we will then construct a specific (s, t)-cut (S, T) and show its capacity is equal to the flow's value.

$$|f^*| = \text{capacity}(S, T)$$

Combining these steps proves that this cut must be a minimum cut, and therefore $\max \text{flow} = \min \text{cut}$.

Today's steps

For the rest of this lecture we prove the following which completes the proof of max flow min cut theorem.

- We prove weak duality
- We show if f^* is the maximum flow, we can find a special (s, t)-cut $c^* = (S, T)$ such that
$$\text{max flow } = |f^*| = \text{capacity}(S, T)\,.$$

Together this implies that c^* is indeed a cut with the minimum capacity.

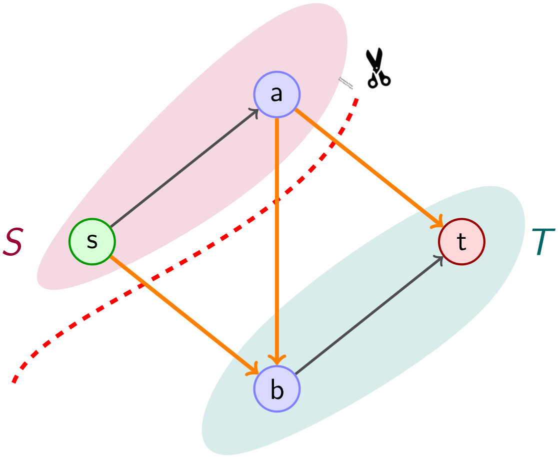
Hence, we conclude the proof of max flow min cut theorem.

4. Proof of Weak Duality

Weak Duality: An Intuitive View

Claim. Every flow from s to t must cross any s–t cut.

- Think of the set S as one large "super-node."
- The entire flow $|f|$ originates inside this super-node at s.
- To reach the sink $t \in T$, all of the flow must eventually **exit** S.
- The only exit is via edges crossing from S to T.
- Those edges can carry at most the sum of their capacities.



Proof of Weak Duality

Proof: Recall from earlier:

$$\forall\ v \in V \setminus \{s,t\} : \qquad \underbrace{\sum_{u \in IN(v)} f(u,v)}_{\text{incoming flow to } v} - \underbrace{\sum_{u \in OUT(v)} f(v,u)}_{\text{outgoing flow from } v} = 0\,,$$

and,

$$|f| = \underbrace{\sum_{u \in OUT(s)} f(s,u)}_{\text{outgoing flow from } s} - \underbrace{\sum_{u \in IN(s)} f(u,s)}_{\text{incoming flow to } s}\ .$$

Proof of Weak Duality

We can express the value of the flow as the net flow out of the set S:

$$
\begin{aligned}
|f| &= \underbrace{\sum_{u \in OUT(s)} f(s,u)}_{\text{outgoing flow from } s} - \underbrace{\sum_{u \in IN(s)} f(u,s)}_{\text{incoming flow to } s} \\
&= \underbrace{\sum_{u \in OUT(s)} f(s,u)}_{\text{outgoing flow from } s} - \underbrace{\sum_{u \in IN(s)} f(u,s)}_{\text{incoming flow to } s} \\
&\quad + \sum_{v \in S \setminus \{s\}} \left(\underbrace{\sum_{u \in OUT(v)} f(v,u)}_{\text{outgoing flow from } v} - \underbrace{\sum_{u \in IN(v)} f(u,v)}_{\text{incoming flow to } v} \right)
\end{aligned}
$$

(via flow conservation for $v \in S \setminus \{s\}$)

Proof of Weak Duality

$$
\begin{aligned}
|f| &= \sum_{v \in S} \left(\sum_{u \in OUT(v)} f(v,u) - \sum_{u \in IN(v)} f(u,v) \right) \\
&= \sum_{v \in S, u \in T} f(v,u) - \sum_{v \in S, u \in T} f(u,v) \qquad \text{(flow within S cancels)}
\end{aligned}
$$

Proof of Weak Duality

$$
\begin{aligned}
|f| &= \sum_{v \in S} \left(\sum_{u \in OUT(v)} f(v,u) - \sum_{u \in IN(v)} f(u,v) \right) \\
&= \sum_{v \in S, u \in T} f(v,u) - \sum_{v \in S, u \in T} f(u,v) && \text{(flow within S cancels)} \\
&\leq \sum_{v \in S, u \in T} f(v,u) && \text{(since } f(u,v) \geq 0\text{)} \\
&\leq \sum_{v \in S, u \in T} c(v,u) && \text{(since } f(v,u) \leq c(v,u)\text{)} \\
&= \text{capacity}(S,T)
\end{aligned}
$$

Proof of Weak Duality

$$
\begin{aligned}
|f| &= \sum_{v \in S} \left(\sum_{u \in OUT(v)} f(v,u) - \sum_{u \in IN(v)} f(u,v) \right) \\
&= \sum_{v \in S, u \in T} f(v,u) - \sum_{v \in S, u \in T} f(u,v) && \text{(flow within S cancels)} \\
&\leq \sum_{v \in S, u \in T} f(v,u) && (\text{since } f(u,v) \geq 0) \\
&\leq \sum_{v \in S, u \in T} c(v,u) && (\text{since } f(v,u) \leq c(v,u)) \\
&= \text{capacity}(S,T)
\end{aligned}
$$

Corollary: The maximum flow value is less than or equal to the minimum cut capacity.

5. When flow leads to a cut

Residual Graph and Maximal Flow

Key Observation:

If a flow f^* is maximum, its residual graph G_{f^*} has **no augmenting path**.

$$\text{Maximum flow} \quad \Rightarrow \quad \text{No augmenting path.}$$

Why?

Residual Graph and Maximal Flow

Key Observation:

If a flow f^* is maximum, its residual graph G_{f^*} has **no augmenting path**.

$$\text{Maximum flow} \quad \Rightarrow \quad \text{No augmenting path.}$$

Why?

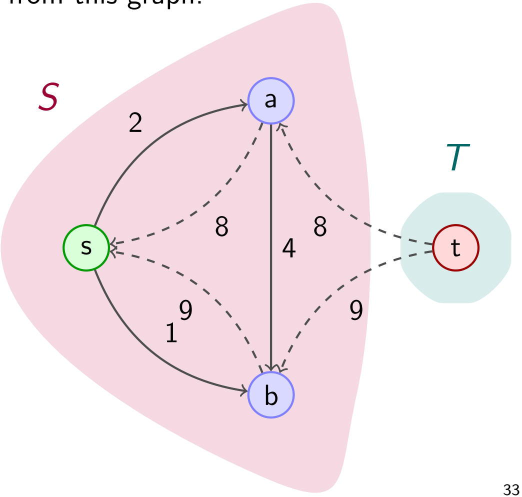
If G_{f^*} had an augmenting path, we could send more flow. That would contradict maximality.

Finding a Special Cut

Suppose there is **no** augmenting paths from s to t in the residual graph G_{f^*}. A cut (which turns out to be a min-cut) can be found from this graph:

- $S \leftarrow$ set of all vertices reachable from s.
- $T \leftarrow$ be all other vertices.

In our example:
from s we can reach $\{a, b\}$.
So $S = \{s, a, b\}$ and $T = \{t\}$.



Finding a Special Cut

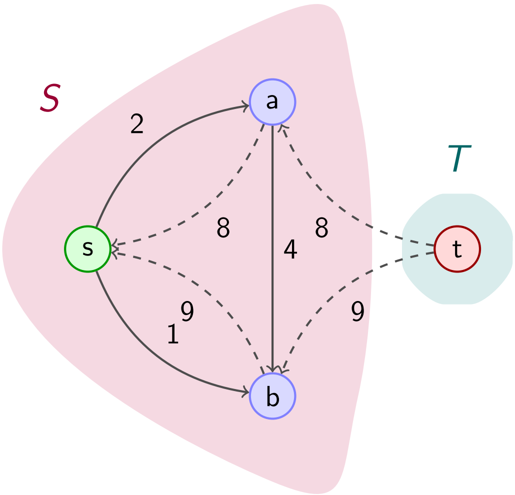
Nodes in T are not reachable. This implies two crucial facts for our cut:

- Every edge from S to T is **saturated**.

$$\forall\ v \in S, u \in T : \quad f(v,u) = c(v,u)$$

- Every edge from T to S is **empty**.

$$\forall\ v \in S, u \in T : \quad f(u,v) = 0$$



Proof: Final Step

Let's revisit the flow value equation from the Weak Duality slide, using our special cut (S, T):

$$|f| = \sum_{v \in S, u \in T} f(v, u) - \sum_{v \in S, u \in T} f(u, v)$$

Proof: Final Step

Let's revisit the flow value equation from the Weak Duality slide, using our special cut (S, T):

$$|f| = \sum_{v \in S, u \in T} f(v, u) - \sum_{v \in S, u \in T} f(u, v)$$

From the previous slide, we know for this cut:

- $f(v, u) = c(v, u)$ for edges from S to T.
- $f(u, v) = 0$ for edges from T to S.

Proof: Final Step

Let's revisit the flow value equation from the Weak Duality slide, using our special cut (S, T):

$$|f| = \sum_{v \in S, u \in T} f(v, u) - \sum_{v \in S, u \in T} f(u, v)$$

From the previous slide, we know for this cut:

- $f(v, u) = c(v, u)$ for edges from S to T.
- $f(u, v) = 0$ for edges from T to S.

Substituting these in:

$$|f| = \sum_{u \in S, v \in T} c(u, v) - \sum_{u \in S, v \in T} 0 \qquad = \sum_{u \in S, v \in T} c(u, v) = \text{capacity}(S, T)$$

Optimality of Ford-Fulkerson algorithm

This is why the Ford–Fulkerson algorithm can safely terminate: once no augmenting path exists, the current flow is going to be equal to the capacity of the cut. And, using the weak duality the flow is indeed maximum.

Running Time of Ford-Fulkerson

Running Time of Ford-Fulkerson

How fast is the augmenting path algorithm?

- Each iteration involves finding a path in the residual graph, which can be done with BFS or DFS in $O(E)$ time.
- If all capacities are **integers**, each augmenting path increases the total flow by at least 1.
- Let f^* be the maximum flow. The algorithm performs at most $|f^*|$ iterations.

Worst-Case Running Time

The total running time is $O(E \cdot |f^*|)$.

Running Time of Ford-Fulkerson

Worst-Case Running Time

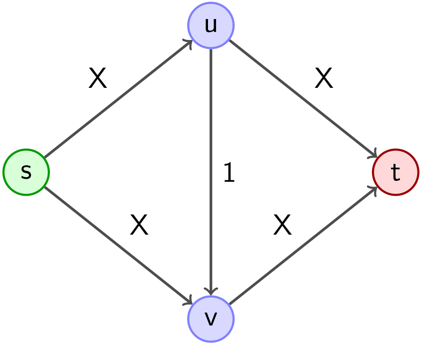
The total running time is $O(E \cdot |f^*|)$.

Is this good?

- This is a **pseudo-polynomial** time bound. It depends on the *value* of the maximum flow, not just the size of the graph.
- If $|f^*|$ is very large, this can be very slow! The value of $|f^*|$ can be exponential in the number of bits needed to represent the capacities.

A Bad Example for Ford-Fulkerson

The choice of augmenting path is critical. Consider this network where X is a large integer:

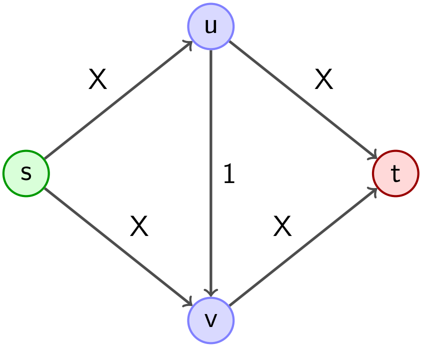


The maximum flow is clearly $2X$.

What if the algorithm makes unlucky choices?

A Bad Example for Ford-Fulkerson

The choice of augmenting path is critical. Consider this network where X is a large integer:



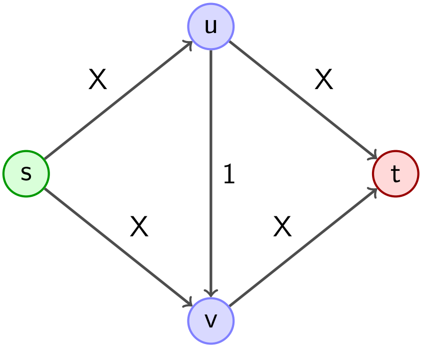
The maximum flow is clearly $2X$.

What if the algorithm makes unlucky choices?

1. Find path $s \to u \to v \to t$. Bottleneck is 1. Push 1 unit of flow.

A Bad Example for Ford-Fulkerson

The choice of augmenting path is critical. Consider this network where X is a large integer:



The maximum flow is clearly $2X$.

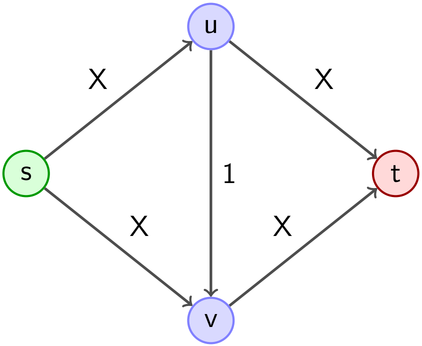
What if the algorithm makes unlucky choices?

1. Find path $s \to u \to v \to t$. Bottleneck is 1. Push 1 unit of flow.

2. The residual graph now has an edge $v \to u$. Find path $s \to v \to u \to t$. Bottleneck is 1. Push 1 unit of flow.

A Bad Example for Ford-Fulkerson

The choice of augmenting path is critical. Consider this network where X is a large integer:



The maximum flow is clearly $2X$.

What if the algorithm makes unlucky choices?

1. Find path $s \to u \to v \to t$. Bottleneck is 1. Push 1 unit of flow.
2. The residual graph now has an edge $v \to u$. Find path $s \to v \to u \to t$. Bottleneck is 1. Push 1 unit of flow.
3. This cycle can repeat... $2X$ times! Each time, we only add 1 to the total flow.

This leads to a running time of $\Theta(X \cdot E)$, which is exponential in the input size (since X can be represented with $O(\log X)$ bits).

Edmonds-Karp: Better Path Choices

To guarantee a polynomial running time, we need a smarter way to choose the augmenting path.

In 1972, Jack Edmonds and Richard Karp proposed two heuristics that lead to efficient algorithms. The key is to choose a path that is "best" in some sense, rather than an arbitrary one.

Two Heuristics for Choosing Augmenting Paths

1. **Fattest Path:** Choose the augmenting path with the **largest bottleneck capacity**.
2. **Shortest Path:** Choose the augmenting path with the **fewest number of edges**.

Both of these strategies avoid the "bad" behavior from the previous example and lead to polynomial time algorithms.

Edmonds-Karp: Fattest Augmenting Path

The Rule: In each step, find an augmenting path P that maximizes its bottleneck capacity, $min_{e \in P} c_f(e)$.

Intuition: This is a greedy approach that tries to make as much progress as possible in each iteration by sending the largest possible amount of flow.

How to find it?

- This can be solved with an algorithm similar to Dijkstra's or Prim's in $O(E \log V)$ time.

Running Time

For integer capacities, the number of augmentations is bounded. The total running time is:

$$O(E^2 \log V \log |f^*|)$$

This is polynomial in the input size, but still depends on the value of the flow.

Edmonds-Karp: Shortest Augmenting Path

The Rule: In each step, find an augmenting path with the minimum number of edges.

Intuition: This seems simple, but it has a powerful effect on the structure of the residual graph over time.

How to find it?

- Just run a **Breadth-First Search (BFS)** on the residual graph, starting from s. This takes $O(E)$ time.

Running Time

The algorithm performs at most $O(VE)$ augmentations. Since each BFS takes $O(E)$ time, the total running time is:

$$O(VE^2)$$

This is a **strongly polynomial** time bound because it does not depend on the capacities at all!

Edmonds-Karp: Shortest Path Intuition

- The bad behavior of the generic Ford-Fulkerson algorithm comes from finding long, low-capacity paths that “zig-zag” back and forth, repeatedly canceling and re-establishing small amounts of flow.

Edmonds-Karp: Shortest Path Intuition

- The bad behavior of the generic Ford-Fulkerson algorithm comes from finding long, low-capacity paths that "zig-zag" back and forth, repeatedly canceling and re-establishing small amounts of flow.

- By choosing the *shortest augmenting path* (fewest number of edges) in the residual graph G_f, the Edmonds-Karp strategy **globally** changes the network's structure.

Edmonds-Karp: Shortest Path Intuition

- The bad behavior of the generic Ford-Fulkerson algorithm comes from finding long, low-capacity paths that "zig-zag" back and forth, repeatedly canceling and re-establishing small amounts of flow.

- By choosing the *shortest augmenting path* (fewest number of edges) in the residual graph G_f, the Edmonds-Karp strategy **globally** changes the network's structure.

- The algorithm terminates quickly because of the cumulative effect of shortest path augmentations.

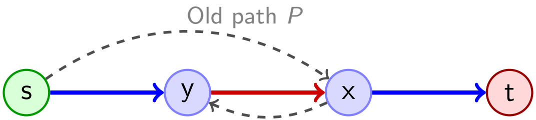
Edmonds-Karp: Why Shortest Path Works

- The length of the shortest path from s to t in the residual graph, $d_f(s,t)$, **never decreases** with an augmentation.
- A powerful result shows that if (u,v) is a saturated edge, the shortest path from s to v increases by 2.
- Since max path length is $O(V)$, this cannot happen more than $O(V)$ in the entire execution.
- Since there are $O(E)$ edges total, the total number of augmentations is at most $O(VE)$.

The Non-Decreasing Distance Property

Core Property: After any augmentation, the shortest path distance from the source s to any vertex x, $d_{f'}(s,x)$, *never decreases*.

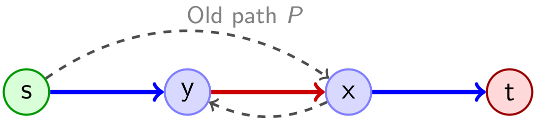
- **Hypothesis:** Suppose, for contradiction, that a shorter path P' in $G_{f'}$ exists: $d_{f'}(s,x) < d_f(s,x)$.
- Let's take x to be the closest vertex to s with this property (smallest $d_{f'}(s,x)$).
- The path P' must use a newly created **backward edge**, (y,x). This edge only exists because the previous shortest path passed through the original edge (x,y).



The Non-Decreasing Distance Property

- P and P' are both shortest path. Thus, we have.

$$d_f(s,y) = d_f(s,x) + 1 \qquad d_{f'}(s,x) = d_{f'}(s,y) + 1$$



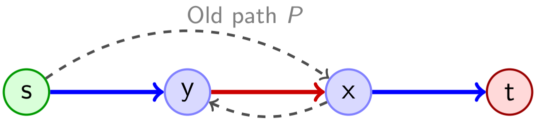
The Non-Decreasing Distance Property

- P and P' are both shortest path. Thus, we have.

$$d_f(s,y) = d_f(s,x) + 1 \qquad d_{f'}(s,x) = d_{f'}(s,y) + 1$$

- The shortest path distance to y must be non-decreasing; Otherwise, we would have picked y instead of x.

$$d_{f'}(s,y) \geq d_f(s,y)\,.$$



The Non-Decreasing Distance Property

- P and P' are both shortest path. Thus, we have.

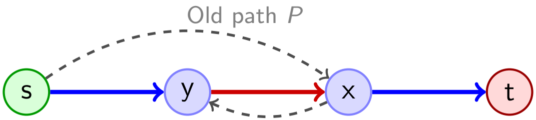
$$d_f(s, y) = d_f(s, x) + 1 \qquad d_{f'}(s, x) = d_{f'}(s, y) + 1$$

- The shortest path distance to y must be non-decreasing; Otherwise, we would have picked y instead of x.

$$d_{f'}(s, y) \geq d_f(s, y) \,.$$

- Putting these together yields a contradiction:

$$d_{f'}(s, x) = d_{f'}(s, y) + 1 \geq d_f(s, y) + 1 = d_f(s, x) + 2$$



Why an Edge Saturates at Most $O(V)$ Times

- **Initial Saturation:** (u, v) is on the shortest path:

$$d_f(s, v) = d_f(s, u) + 1$$

After augmentation, the forward capacity $c_f(u, v) = 0$.

Why an Edge Saturates at Most $O(V)$ Times

- **Initial Saturation:** (u, v) is on the shortest path:

$$d_f(s, v) = d_f(s, u) + 1$$

  After augmentation, the forward capacity $c_f(u, v) = 0$.

- **Re-enabling (Flow Cancellation):** The only way to re-enable (u, v) is to use its backward edge, (v, u), in a later augmentation P' (another shortest path).

$$d_{f'}(s, u) = d_{f'}(s, v) + 1$$

Why an Edge Saturates at Most $O(V)$ Times

- **Initial Saturation:** (u, v) is on the shortest path:

$$d_f(s, v) = d_f(s, u) + 1$$

  After augmentation, the forward capacity $c_f(u, v) = 0$.

- **Re-enabling (Flow Cancellation):** The only way to re-enable (u, v) is to use its backward edge, (v, u), in a later augmentation P' (another shortest path).

$$d_{f'}(s, u) = d_{f'}(s, v) + 1$$

- **Distance Increase:** Since the distance $d_f(s, v)$ *never decreases*, we must have:

$$d_{f'}(s, u) = d_{f'}(s, v) + 1 \geq d_f(s, v) + 1 = d_f(s, u) + 2$$

The shortest distance from s to u must increase by at least 2, every time (u, v) is saturated.

The Edmonds-Karp Runtime Proof

Since the maximum shortest path distance is $V - 1$, an edge can be saturated at most $O(V)$ times.

The total running time of the Edmonds-Karp shortest path algorithm is $\mathbf{O(VE^2)}$.

$$\begin{aligned}\text{Total time} &= \text{number of augmentations} \times \text{time to find the path} \\ &= O(VE) \times O(E) = \mathbf{O(VE^2)}\end{aligned}$$

Key Takeaway

This is a **strongly polynomial** time bound because it does not depend on the magnitude of the capacities (the value $|f^*|$) at all!

Summary of Running Times

The choice of augmenting path makes a huge difference in the efficiency of the Ford-Fulkerson method.

Algorithm	Path Choice	Worst-Case Running Time
Generic Ford-Fulkerson	Arbitrary Path (DFS/BFS)	$O(E \cdot \lvert f^* \rvert)$
Edmonds-Karp	Fattest Path (Dijkstra-like)	$O(E^2 \log V \log \lvert f^* \rvert)$
Edmonds-Karp	**Shortest Path (BFS)**	$\mathbf{O(VE^2)}$

Key Takeaway:

- Simply using BFS to find the augmenting path guarantees that the Ford-Fulkerson algorithm terminates in polynomial time, regardless of the edge capacities.
- While faster algorithms exist (e.g., Push-Relabel, Dinic's), the Edmonds-Karp shortest path variant is a fundamental and powerful result.

Flow Decomposition

Flow Components: Path and Cycle Flows

Every flow is a combination of these two fundamental unit flows.

1. Path Flow (Unit Flow)

For a directed path P from s to t:

- **Value:** $|P| = 1$.
- **Definition:** The unit flow $P : E \to \mathbb{R}$ is defined as:

$$P(u \to v) = \begin{cases} 1 & \text{if } u \to v \in P \\ -1 & \text{if } v \to u \in P \\ 0 & \text{otherwise} \end{cases}$$

Flow Components: Path and Cycle Flows

Every flow is a combination of these two fundamental unit flows.

2. Cycle Flow (Circulation)

For a directed cycle C:

- **Value:** $|C| = 0$.
- **Definition:** The unit flow $C : E \to \mathbb{R}$ is defined as:

$$C(u \to v) = \begin{cases} 1 & \text{if } u \to v \in C \\ -1 & \text{if } v \to u \in C \\ 0 & \text{otherwise} \end{cases}$$

Flow Linearity:

For now, ignore the capacities...

- A flow is essentially a function mapping an edge to a number.
- It also consistently maps the edge in the opposite direction to the negative of that number.

$$f(u, v) = -f(v, u)$$

- Any linear combination of (s, t)-flows is also an (s, t)-flow.

$$\text{If } h = \alpha f + \beta g$$

  - shorthand for: $h(u, v) = \alpha f(u, v) + \beta g(u, v)$
- The size of the flow is also preserved:

$$|h| = \alpha|f| + \beta|g|$$

The Flow Decomposition Theorem

Every flow is a combination of these two fundamental unit flows: Paths and Cycles.

Theorem

Every **non-negative** (s, t)-flow f can be written as a **positive linear combination** of directed (s, t)-paths and directed cycles.

Applications: Many practical problems (e.g., transportation, communication, logistics) need a list of specific routes used by the flow, not just edge capacities. This theorem, and its associated algorithm, allow us to convert a flow solution to a path-based representation.

Proof Idea: Flow Decomposition (Induction I)

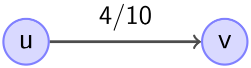
The proof uses induction on $\#\mathbf{f}$, the number of edges carrying non-zero flow.

1. **Base Case:** If $\# f = 0$, the flow is trivially decomposed.

Proof Idea: Flow Decomposition (Induction I)

The proof uses induction on $\#\mathbf{f}$, the number of edges carrying non-zero flow.

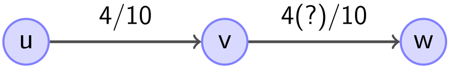
1. **Base Case:** If $\#f = 0$, the flow is trivially decomposed.
2. **Inductive Step (Finding a Flow Structure):**
   - Since $\#f > 1$ an edge (u, v) exists with positive flow.



Proof Idea: Flow Decomposition (Induction I)

The proof uses induction on $\#\mathbf{f}$, the number of edges carrying non-zero flow.

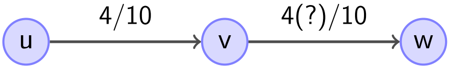
1. **Base Case:** If $\#f = 0$, the flow is trivially decomposed.
2. **Inductive Step (Finding a Flow Structure):**
   - Since $\#f > 1$ an edge (u, v) exists with positive flow.
   - By the *Flow Conservation Property* (for any node $v \in V \setminus \{s, t\}$), if flow enters v, it must also leave v.



Proof Idea: Flow Decomposition (Induction I)

The proof uses induction on $\#\mathbf{f}$, the number of edges carrying non-zero flow.

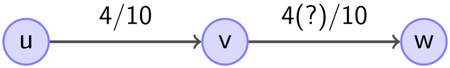
1. **Base Case:** If $\#f = 0$, the flow is trivially decomposed.
2. **Inductive Step (Finding a Flow Structure):**
   - Since $\#f > 1$ an edge (u, v) exists with positive flow.
   - By the *Flow Conservation Property* (for any node $v \in V \setminus \{s, t\}$), if flow enters v, it must also leave v.
   - This property guarantees that as long as we are not at s or t, we can *always extend the walk* to an outgoing edge with positive flow.



Proof Idea: Flow Decomposition (Induction I)

The proof uses induction on $\#\mathbf{f}$, the number of edges carrying non-zero flow.

1. **Base Case:** If $\#f = 0$, the flow is trivially decomposed.
2. **Inductive Step (Finding a Flow Structure):**
   - Since $\#f > 1$ an edge (u, v) exists with positive flow.
   - By the *Flow Conservation Property* (for any node $v \in V \setminus \{s, t\}$), if flow enters v, it must also leave v.
   - This property guarantees that as long as we are not at s or t, we can *always extend the walk* to an outgoing edge with positive flow.
   - The walk must eventually either reach s/t (forming an $s \to t$ Path) or visit a vertex twice (forming a Cycle).



Proof Idea: Flow Decomposition

Once a path or cycle structure is found, we apply the recursive step.

3. **Decompose and Recurse:**
   - Let S be the found structure (Path P or Cycle C).
   - Determine the bottleneck flow $F = \min_{e \in S} f(e)$.
   - Construct a new flow $f' = f - F \cdot S$.

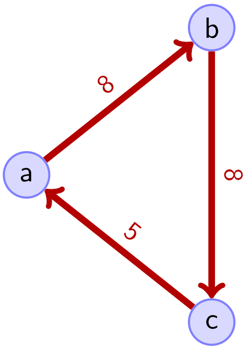
Proof Idea: Flow Decomposition

Once a path or cycle structure is found, we apply the recursive step.

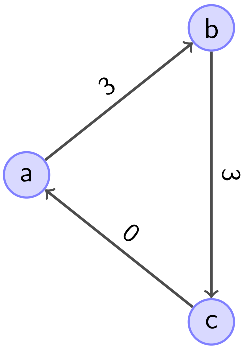
3. **Decompose and Recurse:**
   - Let S be the found structure (Path P or Cycle C).
   - Determine the bottleneck flow $F = \min_{e \in S} f(e)$.
   - Construct a new flow $f' = f - F \cdot S$.
4. **Conclusion:**
   - Subtracting F units empties **at least one edge** in S, so the new flow $\#f' < \#f$.
   - By the inductive hypothesis, f' is decomposed. Adding back $F \cdot S$ completes the decomposition of f.

$$f = f' + F \cdot S$$

Removing Flow Component



Cycle C: $a \to b \to c \to a$.
Bottleneck $F = \min(8, 8, 5) = \mathbf{5}$.



Flow after removing $5 \cdot C$.

Implications of Decomposition Theorem

- The proof also immediately translates directly into an algorithm.
  - The total number of paths and cycles in the decomposition is at most $|E|$, the number of edges in the network.
  - Finding a cycle or a path takes $O(|V|)$ (why not $O(|E|)$?)
  - The total time for decomposition is $O(|V| \cdot |E|)$.

- Any circulation ($|f| = 0$) can be decomposed into a weighted sum of cycles; no paths are necessary.

- Any acyclic (s, t)-flow can be decomposed into a weighted sum of (s, t)-paths; no cycles are necessary.

Chapter Summary: Max Flow and Min Cut

- **Core Duality:** The value of the Maximum (s, t)-Flow is always equal to the capacity of the Minimum (s, t)-Cut (Maxflow-Mincut Theorem).

Chapter Summary: Max Flow and Min Cut

- **Core Duality:** The value of the Maximum (s, t)-Flow is always equal to the capacity of the Minimum (s, t)-Cut (Maxflow-Mincut Theorem).
- **Ford-Fulkerson Method:** This iterative algorithm finds the max flow by repeatedly locating and augmenting flow along a path in the residual graph.

Chapter Summary: Max Flow and Min Cut

- **Core Duality:** The value of the Maximum (s, t)-Flow is always equal to the capacity of the Minimum (s, t)-Cut (Maxflow-Mincut Theorem).
- **Ford-Fulkerson Method:** This iterative algorithm finds the max flow by repeatedly locating and augmenting flow along a path in the residual graph.
- **Residual Graph:** This crucial construct allows the algorithm to "undo" or reroute previously committed flow by using *backward edge*.
- **Efficiency:** The choice of augmenting path is critical for performance:
  - Generic Ford-Fulkerson is $\mathbf{O(E \cdot |f^*|)}$ (pseudo-polynomial).
  - Edmonds-Karp uses the shortest path (via BFS) to guarantee a $\mathbf{O(VE^2)}$ time bound (strongly polynomial).
- **Flow Decomposition:** Any flow can be written as a positive linear combination of unit flows along $\mathbf{s \to t}$ **paths** and **cycles**. Flow cycles can be removed to create an acyclic max flow.

References

Erickson, J. (2019).
Algorithms.
Self-published.