**Flow network.**
- Digraph \( G = (V, E) \), nonnegative edge capacities \( c(e) \).
- Two distinguished nodes: \( s = \text{source}, t = \text{sink} \).
- Assumptions: no parallel edges, no edges entering \( s \) or leaving \( t \).

**Def.** An \( s-t \) cut is a partition \((A, B)\) of \( V \) with \( s \in A \) and \( t \in B \).

**Def.** The capacity of a cut \((A, B)\) is: 
\[
\text{cap}(A, B) = \sum_{e \text{ out of } A} c(e)
\]
Min s-t cut problem. Find an s-t cut of minimum capacity.

Minimum Cut Problem

Max flow problem. Find s-t flow of maximum value.

Maximum Flow Problem

Flow value lemma. Let $f$ be any flow, and let $(A, B)$ be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving $s$.

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = \text{val}(f)$$
Flows and Cuts

**Flow value lemma.** Let \( f \) be any flow, and let \((A, B)\) be any s-t cut. Then
\[
\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in } A} f(e) = \text{val}(f).
\]

Proof (Pf).
\[
\text{val}(f) = \sum_{e \text{ out of } A} f(e)
\]

by flow conservation, all terms except \( v = s \) are 0
\[
= \sum_{v \in A} \left( \sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to } v} f(e) \right)
\]
\[
= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e).
\]

**Weak duality.** Let \( f \) be any flow, and let \((A, B)\) be any s-t cut. Then the value of the flow is at most the capacity of the cut.

\[
\text{val}(f) \leq \text{cap}(A, B).
\]

Proof (Pf).
\[
\text{val}(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \leq \sum_{e \text{ out of } A} c(e) = \text{cap}(A, B).
\]

**Certificate of Optimality**

**Corollary.** Let \( f \) be any flow, and let \((A, B)\) be any cut. If \( \text{val}(f) = \text{cap}(A, B) \), then \( f \) is a max flow and \((A, B)\) is a min cut.
Towards a Max Flow Algorithm

**Greedy algorithm.**
- Start with \( f(e) = 0 \) for all edge \( e \in E \).
- Find an s-t path \( P \) where each edge has \( f(e) < c(e) \).
- Augment flow along path \( P \).
- Repeat until you get stuck.

**Residual Graph**

- Original edge: \( e = (u, v) \in E \).
  - Flow \( f(e) \), capacity \( c(e) \).
- Residual edge.
  - "Undo" flow sent.
  - \( e = (u, v) \) and \( e^R = (v, u) \).
  - Residual capacity:
    \[
    c_f(e) = \begin{cases} 
    c(e) - f(e) & \text{if } e \in E \\
    f(e) & \text{if } e^R \in E 
    \end{cases}
    \]
- Residual graph: \( G_f = (V, E_f) \).
  - Residual edges with positive residual capacity.
  - \( E_f = \{ e : f(e) < c(e) \} \cup \{ e^R : c(e) > 0 \} \).
Ford-Fulkerson Algorithm

Max-Flow Min-Cut Theorem

Augmenting path theorem. Flow $f$ is a max flow iff there are no augmenting paths.

Max-flow min-cut theorem. [Elias-Feinberg-Shannon 1956, Ford-Fulkerson 1956]
The value of the max flow is equal to the value of the min cut.

Pf. Let $f$ be a flow. Then TFAE:

(i) There exists a cut $(A, B)$ such that $\text{val}(f) = \text{cap}(A, B)$.
(ii) Flow $f$ is a max flow.
(iii) There is no augmenting path relative to $f$.

(i) $\Rightarrow$ (ii) This was the corollary to weak duality lemma.

(ii) $\Rightarrow$ (iii) We show contrapositive.
- Let $f$ be a flow. If there exists an augmenting path, then we can improve $f$ by sending flow along path.

(iii) $\Rightarrow$ (i)
- Let $f$ be a flow with no augmenting paths.
- Let $A$ be set of vertices reachable from $s$ in residual graph.
- By definition of $A$, $s \notin A$.
- By definition of $f$, $t \notin A$.

\[
\text{val}(f) = \sum_{e \in (A)} f(e) - \sum_{e \in (A)} f(e) = \sum_{e \notin (A)} c(e) - \sum_{e \notin (A)} c(e) = \text{cap}(A, B) \]

Proof of Max-Flow Min-Cut Theorem

Assumption. All capacities are integers between 1 and $C$.

Invariant. Every flow value $f(e)$ and every residual capacities $c_f(e)$ remains an integer throughout the algorithm.

Theorem. The algorithm terminates in at most $\text{val}(f^*) \leq nc$ iterations. It can be implemented in $O(mnC)$ time.

Pf. Each augmentation increase value by at least 1.

Integrality theorem. If all capacities are integers, then there exists a max flow $f$ for which every flow value $f(e)$ is an integer.

Pf. Since algorithm terminates, theorem follows from invariant.
**Ford-Fulkerson: An Exponential Input**

**Q.** Is generic Ford-Fulkerson algorithm polynomial in input size?

\[
\text{Let } r = \frac{1 + \sqrt{5}}{2} = 0.618... \\
\text{Max flow } = 1 + r + r^2.
\]

Augmentations: first augment 1 unit, then repeatedly choose path with lowest capacity.

---

**Choosing Good Augmenting Paths**

**Goal:** choose augmenting paths so that:
- Can find augmenting paths efficiently.
- Few iterations.

**Choose augmenting paths with:** [Edmonds-Karp 1972, Dinitz 1970]
- Max bottleneck capacity.
- Sufficiently large bottleneck capacity.
- Fewest number of edges.

---

**Shortest Augmenting Path: Overview of Analysis**

**L1.** The length of the shortest augmenting path never decreases.

**L2.** After at most \( m \) augmentations, the length of the shortest augmenting path strictly increases.

**Theorem.** The shortest augmenting path algorithm performs at most \( O(mn) \) augmentations. It can be implemented in \( O(mn^2) \) time.
- \( O(m) \) time to find shortest augmenting path via BFS.
- \( O(m) \) augmentations for paths of exactly \( k \) edges.
Shortest Augmenting Path: Analysis

Level graph.
- Define \( l(v) \) = length of shortest \( s-v \) path in \( G \).
- \( L_G = (V, E) \) is subgraph of \( G \) that contains only those edges \((u, v) \in E \) with \( l(v) = l(u) + 1 \).
- Compute \( L_G \) in \( O(mn) \) time using BFS, deleting back and side edges.
- \( P \) is a shortest \( s-u \) path in \( G \) iff it is an \( s-u \) path \( L_G \).

\[
\begin{array}{c}
\text{Diagram of Level Graph} \\
\end{array}
\]

Shortest Augmenting Path: Analysis

\[ L_G \]

number of edges

\[ \ell = 0 \quad \ell = 1 \quad \ell = 2 \quad \ell = 3 \]

L2. After at most \( m \) augmentations, the length of the shortest augmenting path strictly increases.
- At least one edge (the bottleneck edge) is deleted from \( L \) after each augmentation.
- No new edges added to \( L \) until length of shortest path strictly increases.

\[ \text{Diagram of Level Graph} \]

\[ L \]

\[ \ell = 0 \quad \ell = 1 \quad \ell = 2 \quad \ell = 3 \]

L1. The length of the shortest augmenting path never decreases.
- Let \( f \) and \( f' \) be flow before and after a shortest path augmentation.
- Let \( L \) and \( L' \) be level graphs of \( G_f \) and \( G_{f'} \).
- Only back edges added to \( G_{f'} \).
- Path with back edge has length greater than previous length.

\[ \text{Diagram of Level Graph} \]

\[ L \]

\[ \ell = 0 \quad \ell = 1 \quad \ell = 2 \quad \ell = 3 \]

\[ L' \]

Shortest Augmenting Path: Analysis

\[ L' \]

Number of edges

\[ \ell = 0 \quad \ell = 1 \quad \ell = 2 \quad \ell = 3 \]

L2. After at most \( m \) augmentations, the length of the shortest augmenting path strictly increases.
- At least one edge (the bottleneck edge) is deleted from \( L \) after each augmentation.
- No new edges added to \( L \) until length of shortest path strictly increases.

Theorem. The shortest augmenting path algorithm performs at most \( O(mn) \) augmentations. It can be implemented in \( O(mn \log n) \) time.

Note: \( \Theta(mn) \) augmentations necessary on some networks.
- Try to decrease time per augmentation instead.
- Dynamic trees \( \Rightarrow O(mn \log n) \) [Sleator-Tarjan, 1983]
- Simple idea \( \Rightarrow O(mn^2) \)
Shortest Augmenting Path: Improved Version

Two types of augmentations.
- Normal augmentation: length of shortest path doesn’t change.
- Special augmentation: length of shortest path strictly increases.

L3. Group of normal augmentations takes $O(mn)$ time.
- Explicitly maintain level graph - it changes by at most $2n$ edges after each normal augmentation.
- Start at $s$, advance along an edge in $L$ until reach $t$ or get stuck.
  - if reach $t$, augment and delete at least one edge
  - if get stuck, delete node

Stop: length of shortest path must have strictly increased.

Shortest Augmenting Path: Improved Version

Two types of augmentations.
- Normal augmentation: length of shortest path doesn’t change.
- Special augmentation: length of shortest path strictly increases.

L3. Group of normal augmentations takes $O(mn)$ time.
- At most $n$ advance steps before you either
  - get stuck: delete a node from level graph
  - reach $t$: augment and delete an edge from level graph

Theorem. Algorithm runs in $O(mn^2)$ time.
- $O(mn)$ time between special augmentations.
- At most $n$ special augmentations.
# History of Worst-Case Running Times

<table>
<thead>
<tr>
<th>Year</th>
<th>Discoverer</th>
<th>Method</th>
<th>Asymptotic Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1951</td>
<td>Dantzig</td>
<td>Simplex</td>
<td>$m n^2 C$</td>
</tr>
<tr>
<td>1955</td>
<td>Ford, Fulkerson</td>
<td>Augmenting path</td>
<td>$m n C$</td>
</tr>
<tr>
<td>1970</td>
<td>Edmonds-Karp</td>
<td>Shortest path</td>
<td>$m^2 n$</td>
</tr>
<tr>
<td>1970</td>
<td>Edmonds-Karp</td>
<td>Fattest path</td>
<td>$m \log C (m \log n)$</td>
</tr>
<tr>
<td>1970</td>
<td>Dinitz</td>
<td>Improved shortest path</td>
<td>$m n^2$</td>
</tr>
<tr>
<td>1972</td>
<td>Edmonds-Karp, Dinitz</td>
<td>Capacity scaling</td>
<td>$m^2 \log C$</td>
</tr>
<tr>
<td>1973</td>
<td>Dinitz-Gabow</td>
<td>Improved capacity scaling</td>
<td>$m n \log C$</td>
</tr>
<tr>
<td>1974</td>
<td>Karzanov</td>
<td>Preflow-push</td>
<td>$n^t$</td>
</tr>
<tr>
<td>1983</td>
<td>Sleator-Tarjan</td>
<td>Dynamic trees</td>
<td>$m n \log n$</td>
</tr>
<tr>
<td>1986</td>
<td>Goldberg-Tarjan</td>
<td>FIFO preflow-push</td>
<td>$m n \log (n^t / m)$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1997</td>
<td>Goldberg-Rao</td>
<td>Length function</td>
<td>$m^{n/2} \log (n^t / m) C$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$m^{n/3} \log (n^t / m) C$</td>
</tr>
</tbody>
</table>

† Edge capacities are between 1 and $C$.  

---

(*) next time