Bipartite Matching

- Given: A bipartite graph \( G=(V,E) \)
  - \( M \subseteq E \) is a matching in \( G \) iff no two edges in \( M \) share a vertex

- Goal: Find a matching \( M \) in \( G \) of maximum possible size

Bipartite matching as a special case of flow

The Network Flow Problem

- How much stuff can flow from \( s \) to \( t \)?
Net Flow: Formal Definition

Given:
A digraph \( G = (V, E) \)
Two vertices \( s, t \) in \( V \) (source & sink)
A capacity \( c(u, v) \geq 0 \) for each \( (u, v) \in E \)
and \( c(u, v) = 0 \) for all non-edges \((u, v)\)

Find:
A flow function \( f: E \rightarrow \mathbb{R} \)
st. for all \( u, v \):
\[ n_0 \leq f(u, v) \leq c(u, v) \]
[Capacity Constraint]
\[ f(u, v) = f(v, u) \]
[Flow Conservation]
Maximizing total flow \( \nu(f) = f_{\text{out}}(s) \)

Notation:
\[ f_{\text{in}}(v) = \sum_{u: (u, v) \in E} f(u, v) \]
\[ f_{\text{out}}(v) = \sum_{w: (v, w) \in E} f(v, w) \]

Example: A Flow Function

While there is an \( s \rightarrow t \) path in \( G \)
Pick such a path, \( p \)
Find \( c \), the min capacity of any edge in \( p \)
Subtract \( c \) from all capacities on \( p \)
Delete edges of capacity \( 0 \)
This does NOT always find a max flow:

If pick \( s \rightarrow b \rightarrow a \rightarrow t \) first, flow stuck at 2.
But flow 3 possible.

A Brief History of Flow

<table>
<thead>
<tr>
<th>Year</th>
<th>Algorithm</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1935</td>
<td>Ford-Fulkerson</td>
<td>Gomory</td>
</tr>
<tr>
<td>1955</td>
<td>Edmonds &amp; Karp</td>
<td>Gomory</td>
</tr>
<tr>
<td>1957</td>
<td>Dinitz</td>
<td>Gomory</td>
</tr>
<tr>
<td>1970</td>
<td>Dinic</td>
<td>Gomory</td>
</tr>
<tr>
<td>1982</td>
<td>Goldberg &amp; Tarjan</td>
<td>Gomory</td>
</tr>
<tr>
<td>1985</td>
<td>Goldberg &amp; Tarjan</td>
<td>Gomory</td>
</tr>
<tr>
<td>1987</td>
<td>Goldberg &amp; Tarjan</td>
<td>Gomory</td>
</tr>
<tr>
<td>1991</td>
<td>Goldberg &amp; Tarjan</td>
<td>Gomory</td>
</tr>
<tr>
<td>1998</td>
<td>Goldberg &amp; Rao</td>
<td>Gomory</td>
</tr>
</tbody>
</table>

Max Flow via a Greedy Alg?

Greed Revisited:
Residual Graph & Augmenting Path
Greed Revisited: An Augmenting Path

Residual Capacity

- The residual capacity (w.r.t. $f$) of $(u,v)$ is $c_P(u,v) = c(u,v) - f(u,v)$ if $f(u,v) \leq c(u,v)$ and $c_P(u,v) = f(v,u)$ if $f(v,u) > 0$

- e.g. $c_P(s,b) = 7$; $c_P(a,x) = 1$; $c_P(x,a) = 3$

Residual Graph & Augmenting Paths

- The residual graph (w.r.t. $f$) is the graph $G_f = (V, E_f)$, where $E_f = \{ (u,v) \mid c_P(u,v) > 0 \}$

- Two kinds of edges
  - Forward edges
    - $f(u,v) < c(u,v)$ so $c_P(u,v) = c(u,v) - f(u,v) > 0$
  - Backward edges
    - $f(u,v) > 0$ so $c_P(v,u) > f(v,u) - f(u,v) > 0$

- An augmenting path (w.r.t. $f$) is a simple $s \rightarrow t$ path in $G_f$.

A Residual Network

An Augmenting Path

Augmenting A Flow

$$
\text{augment}(f,P) = \min_{(u,v) \in P} c_P(u,v) \quad \text{"bottleneck}(P)"
$$

for each $e \in P$

- if $e$ is a forward edge then
  - increase $f(e)$ by $c_P$
- else ($e$ is a backward edge)
  - decrease $f(e)$ by $c_P$

endif

endfor

return$(f)$
Augmenting A Flow

Claim 6.1
If \( G_f \) has an augmenting path \( P \), then the function \( f' = \text{augment}(f, P) \) is a legal flow.

Proof:
- \( f' \) and \( f \) differ only on the edges of \( P \) so only need to consider such edges \((u,v)\)

Proof of Claim 6.1
- If \((u,v)\) is a forward edge then
  \[
  f'(u,v) = f(u,v) + c_P \leq f(u,v) + c(u,v) = f(u,v) + c(u,v) - f(u,v) = f(u,v) + c(u,v) - f(u,v) = c(u,v)
  \]
- If \((u,v)\) is a backward edge then \( f \) and \( f' \) differ on flow along \((v,u)\) instead of \((u,v)\)
  \[
  f'(v,u) = f(v,u) - c_P \geq f(v,u) - c(u,v) = f(v,u) - f(v,u) = 0
  \]
- Other conditions like flow conservation still met

Ford-Fulkerson Method
Start with \( f = 0 \) for every edge
While \( G_f \) has an augmenting path, augment

Questions:
- Does it halt?
- Does it find a maximum flow?
- How fast?

Observations about Ford-Fulkerson Algorithm
- At every stage the capacities and flow values are always integers (if they start that way)
- The flow value \( v'(f') = v(f) + c_P \geq v(f) \) for \( f' = \text{augment}(f, P) \)
  - Since edges of residual capacity 0 do not appear in the residual graph
- Let \( C = \sum_{(s,u) \in E} c(s,u) \)
  - \( v(f) \leq C \)
  - F-F does at most \( C \) rounds of augmentation since flows are integers and increase by at least 1 per step

Running Time of Ford-Fulkerson
- For \( f = 0 \), \( G_f = G \)
- Finding an augmenting path in \( G_f \) is graph search \( O(n+m) = O(m) \) time
- Augmenting and updating \( G_f \) is \( O(n) \) time
- Total \( O(mC) \) time
- Does is find a maximum flow?
  - Need to show that for every flow \( f \) that isn't maximum \( G_f \) contains an \( s-t \)-path
Corollary (1)

We know by Claims 6.6 & 6.8 that any flow $f'$ satisfies $\nu(f') \leq c(S,T)$ and we know that F-F runs for finite time until it finds a flow $f$ satisfying conditions of Claim 6.10.

Therefore by 6.10 for any flow $f$, $\nu(f) \geq c(S,T)$.

Corollary (2)

For any graph $G$, the value $\nu(f)$ of a maximum flow $f$ is $\nu(f) = c(S,T)$ for any $s$-t cut $S \subseteq V$.

Max Flow / Min Cut Theorem

Claim 6.10 For any flow $f$, if $G$ has no augmenting path then there is some $s$-t cut $(S,T)$ such that $\nu(f) = c(S,T)$ (proof next slide).

We know by Claims 6.6 & 6.8 that any flow $f'$ satisfies $\nu(f') \leq c(S,T)$ and we know that F-F runs for finite time until it finds a flow $f$ satisfying conditions of Claim 6.10.

Therefore by 6.10 for any flow $f$, $\nu(f) \geq c(S,T)$.

Corollary (1) F-F computes a maximum flow in $G$.

(2) For any graph $G$, the value $\nu(f)$ of a maximum flow $f$ is $\nu(f) = c(S,T)$ for any $s$-t cut $S \subseteq V$.
Flow Integrality Theorem

If all capacities are integers
- The max flow has an integer value
- Ford-Fulkerson method finds a max flow in which \( f(u,v) \) is an integer for all edges \((u,v)\)

\[
\begin{array}{c}
\text{0.5/1} \\
\text{0.5/1} \\
\text{0.5/1} \\
\end{array}
\]

Claim 6.10

Let \( S = \{ u \mid \exists \text{ a path in } G_f \text{ from } s \text{ to } u \} \)

\[
T = V \setminus S; \ s \in S, \ t \in T
\]

For any \((u,v)\) in \( S \times T \), \( \exists \) an path in \( G_f \) from \( s \) to \( u \), but not to \( v \).

\[
\therefore (u,v) \text{ has 0 residual capacity:} \\
(u,v) \in E \Rightarrow \text{saturated} \ \ f(u,v) = c(u,v) \\
(v,u) \in E \Rightarrow \text{no flow} \ \ f(v,u) = 0
\]

This is true for every edge crossing the cut, i.e.

\[
f^f(S) = \sum_{v \in T} f(u,v) = \sum_{u \in T} c(u,v) = c(S,T) \text{ and } f^f(S) = 0 \text{ so } \\
\forall (f) = f^f(S) - f^f(T) = c(S,T)
\]

Corollaries & Facts

- If Ford-Fulkerson terminates, then it’s found a max flow.
- It will terminate if \( c(e) \) integer or rational (but may not if they’re irrational).
- However, may take exponential time, even with integer capacities:

\[
\begin{array}{c}
\text{c = 10^9, say}
\end{array}
\]

Bipartite matching as a special case of flow

Integer flows implies each flow is just a subset of the edges
Therefore flow corresponds to a matching
\(O(mC)=O(nm)\) running time

Capacity-scaling algorithm

- General idea:
  - Choose augmenting paths \( P \) with ‘large’ capacity \( c_P \)
  - Can augment flows along a path \( P \) by any amount \( b \leq c_P \)
    - Ford-Fulkerson still works
  - Get a flow that is maximum for the high-order bits first and then add more bits later
Capacity Scaling

Capacity on each edge is at most 1

Capacity Scaling Bit 1

Residual capacity across min cut is at most 1

O(nm) time

Capacity Scaling Bit 2

Residual capacity across min cut is at most m

⇒ O(m) augmentations

Capacity Scaling Bit 3

Residual capacity across min cut is at most m

⇒ O(m) augmentations
**Capacity Scaling Bit 3**

After $O(m)$ augmentations

**Capacity Scaling Final**

**Capacity Scaling Min Cut**

**Total time for capacity scaling**

- $\log_2 U$ rounds where $U$ is largest capacity
- At most $m$ augmentations per round
  - Let $c_i$ be the capacities used in the $i$th round and $f_i$ the maxflow found in the $i$th round
  - For any edge $(u,v)$, $c_{i+1}(u,v) \leq 2c_i(u,v) + 1$
  - 1st round starts with flow $f = 2f_1$
  - Let $(S, T)$ be a min cut from the $i$th round
    - $\nu(f_i) = c_i(S, T)$ so $\nu(f_i) = 2f_i$
  - Let $(S, T)$ be a min cut from the $i$th round
    - $\nu(f_{i+1}) \leq c_{i+1}(S, T) \leq 2c_i(S, T) + m = \nu(f) + m$
  - $O(m)$ time per augmentation
  - Total time $O(m^2 \log U)$

**Edmonds-Karp Algorithm**

- Use a shortest augmenting path (via Breadth First Search in residual graph)
- Time: $O(n m^2)$

**BFS/Shortest Path Lemmas**

Distance from $s$ in $G_i$ is never reduced by:

- Deleting an edge
  - Proof: no new (hence no shorter) path created
- Adding an edge $(u,v)$, provided $v$ is nearer than $u$
  - Proof: BFS is unchanged, since $v$ visited before $(u,v)$ examined
**Key Lemma**

Let $f$ be a flow, $G_f$ the residual graph, and $P$ a shortest augmenting path. Then no vertex is closer to $s$ after augmentation along $P$.

**Proof:** Augmentation along $P$ only deletes forward edges, or adds back edges that go to previous vertices along $P$.

**Theorem**

The Edmonds-Karp Algorithm performs $O(mn)$ flow augmentations.

**Proof:**
- Call $(u,v)$ critical for augmenting path $P$ if it’s closest to $s$ having min residual capacity.
- It will disappear from $G_f$ after augmenting along $P$.
- In order for $(u,v)$ to be critical again the $(u,v)$ edge must re-appear in $G_f$, but that will only happen when the distance to $u$ has increased by 1.
- It won’t be critical again until farther from $s$ so each edge critical at most $n$ times.

**Corollary**

- Edmonds-Karp runs in $O(nm^2)$ time.

**Project Selection**

a.k.a. The Strip Mining Problem

- **Given**
  - a directed acyclic graph $G=(V,E)$ representing precedence constraints on tasks (a task points to its predecessors)
  - a profit value $p(v)$ associated with each task $v \in V$ (may be positive or negative)
- **Find**
  - a set $A \subseteq V$ of tasks that is closed under predecessors, i.e. if $(u,v) \in E$ and $u \in A$ then $v \in A$, that maximizes $\text{Profit}(A) = \sum_{v \in A} p(v)$

**Extended Graph**
For each vertex \( v \)
- If \( p(v) \geq 0 \), add \( (s, v) \) edge with capacity \( p(v) \)
- If \( p(v) < 0 \), add \( (v, t) \) edge with capacity \( -p(v) \)

Extended Graph \( G' \):

Want to arrange capacities on edges of \( G \) so that for minimum \( s-t \)-cut \((S, T)\) in \( G' \), the set \( A=S\{s\} \)
- satisfies precedence constraints
- has maximum possible profit in \( G \)

Cut capacity with \( S\{s\} \) is just \( C - \sum_{v \in A} p(v) \)
- \( \text{Profit}(A) \geq C \) for any set \( A \)
- To satisfy precedence constraints don’t want any original edges of \( G \) going forward across the minimum cut
- That would correspond to a task in \( A=S\{s\} \) that had a predecessor not in \( A-S\{s\} \)
- Set capacity of each of these edges to \( C+1 \)

The minimum cut has size at most \( C \)

Project Selection:

Claim
- Any \( s-t \)-cut \((S, T)\) in \( G' \) such that \( A=S\{s\} \) satisfies precedence constraints has capacity
  \[ c(S, T) = C - \sum_{v \in A} p(v) = C - \text{Profit}(A) \]

Corollary
- A minimum cut \((S, T)\) in \( G' \) yields an optimal solution \( A=S\{s\} \) to the profit selection problem

Algorithm
- Compute maximum flow \( f \) in \( G' \), find the set \( S \) of nodes reachable from \( s \) in \( G' \), and return \( S\{s\} \)

Proof of Claim:

- \( A=S\{s\} \) satisfies precedence constraints
- No edge of \( G \) crosses forward out of \( A \) by our choice of capacities
- Only forward edges cut are of the form \((v, t)\) for \( v \in A \) or \((s, v)\) for \( v \in A \)
- The \((v, t)\) edges for \( v \in A \) contribute
  \[ \sum_{v \in A} p(v) = \sum_{v \in S} p(v) = \text{Profit}(A) \]
- The \((s, v)\) edges for \( v \in A \) contribute
  \[ \sum_{v \in A} p(v) = C - \sum_{v \in S} p(v) \]
- Therefore the total capacity of the cut is
  \[ c(S, T) = C - \sum_{v \in A} p(v) = C - \text{Profit}(A) \]