What are the 2 restrictions on a flow $f$:

1. **Flow constraint**: $iN(x) = oUT(x)$
2. **Capacity constraint**: $f(e) < c(e)$

What is the value of a flow $|f|$:

$$|f| = oUT(s) - iN(s) - \sum_{u \in V} f(s, u) - \sum_{u \in V} f(u, s)$$

How does the Ford-Fulkerson algorithm work?
root of the problem
Edmonds-Karp 2

choose path with fewest edges first.

$$\delta_f(s, v) : \text{In } G_f, \text{ min # of edges in a path from } s \text{ to } v.$$
Observation

\[ \delta_f(s, v) \text{ increases monotonically thru exec} \]

\[ \delta_{i+1}(v) \geq \delta_i(v) \]
for every augmenting path, some edge is critical.
critical edges are removed in next residual graph.
key idea: how many times can an edge be critical?
First time edge $e = (u, v)$ becomes critical.

$\delta_i(s, v) = \delta_i(s, u) + 1$

$\delta_{i+1}(s, u) \geq \delta_i(s, v) = \delta_i(s, u) + 1$
first time \((u,v)\) is critical:
time $i+1$: (u,v) is critical: $\delta_{i+1}(s, v) \geq \delta_i(s, u) + 1$

time $j$: Edge (u,v) STRIKES BACK

some path from $s \rightarrow v \rightarrow u \rightarrow t$

must become the shortest augmenting path

$$\delta_j(s,u) = \delta_j(s,v) + 1 \geq \delta_{i+1}(s,v) + 1 \geq \delta_i(s,u) + 1 + 1$$
time $i+1$: $(u,v)$ is critical: $\delta_{i+1}(s,v) \geq \delta_i(s,u) + 1$

$\delta_j(s,u) = \delta_j(s,v) + 1$

$\delta_{i+1}(s,v) \geq \delta_i(s,u) + 1$
time $j$: Edge $(u,v)$ STRIKES BACK

\[
\begin{align*}
\delta_{i+1}(s, v) & \geq \delta_i(s, u) + 1 \\
\delta_j(s, u) & = \delta_j(s, v) + 1
\end{align*}
\]
QUESTION: How many times can \((u,v)\) be critical?

\[
\frac{V}{2} \quad \text{because} \quad \delta_n(s,u) \text{ is always at most } V-1.
\]
edge critical only \( \frac{v}{2} \) times.
there are only \( E \) edges.
ergo, total # of augmenting paths: \( O(EV) \)
time to find an augmenting path: \( O(E + u) \)
total running time of E-K algorithm: \( O(E^2u) \)
FF \quad O(E|f^*|)

EK2 \quad O(E^{2U}) \quad DNIC \quad O(EU^2)

PUSH-RELABEL

FASTER PUSH-RELABEL \quad O(U^3)

Golberg-Rao: \quad O(E \min \{3 U^{2/3}, E^{1/2} \log \left( \frac{U}{e} \right) \})
Bipartite
maximum bipartite matching
maximum bipartite matching

Here, add 4 unit of flow.
bipartite matching

problem: given a bipartite graph, \( G = (L, R, E) \), all \( E \) are between \( L \) and \( R \).

"largest" find a subset \( M \subseteq E \) such that each node occurs at most once in \( M \).

Further find the largest such \( M \).
Chapter 7 Network Flow & The Problem

One of our original goals is to develop the Maximum-Flow Problem. One of our original goals is to develop the Maximum-Flow Problem was to be able to solve the Bipartite Matching Problem, and we now show how to do this. Recall that a bipartite graph $G = (V, E)$ is an undirected graph whose node set can be partitioned as $V = X \cup Y$, with the property that every edge $e \in E$ has one end in $X$ and the other end in $Y$.

A matching $M$ in $G$ is a subset of the edges $M \subseteq E$ such that each node appears in at most one edge in $M$.

The Bipartite Matching Problem is that of finding a matching in $G$ of largest possible size.

**Designing the Algorithm**

The graph defining a matching problem is undirected, while flow networks are directed; but it is actually not difficult to use an algorithm for the Maximum-Flow Problem to find a maximum matching. Beginning with the graph $G$ in an instance of the Bipartite Matching Problem, we construct a flow network $G'$ as shown in Figure 7.9. First we direct all edges in $G$ from $X$ to $Y$. We then add a node $s$, and an edge $(s, x)$ from $s$ to each node in $X$. We add a node $t$, and an edge $(y, t)$ from each node in $Y$ to $t$. Finally, we give each edge in $G'$ a capacity of 1. We now compute a maximum s-t flow in this network $G'$. We will discover that the value of this maximum is equal to the size of the maximum matching in $G$. Moreover, our analysis will show how one can use the flow itself to recover the matching.

$G' = (V = Is, t, x \cup Ly, t) \leftarrow$ as above

---

(a) (b)

Figure 7.9 (a) A bipartite graph. (b) The corresponding flow network, with all capacities equal to 1.
One of our original goals is to develop the Maximum-Flow Problem. We now show how to do this. Recall that a bipartite graph $G = (V, E)$ is an undirected graph whose node set can be partitioned as $V = X \cup Y$, with the property that every edge $e \in E$ has one end in $X$ and the other end in $Y$. A matching $M$ in $G$ is a subset of the edges $M \subseteq E$ such that each node appears in at most one edge in $M$. The Bipartite Matching Problem is that of finding a matching in $G$ of largest possible size.

### Designing the Algorithm

The graph defining a matching problem is undirected, while flow networks are directed; but it is actually not difficult to use an algorithm for the Maximum-Flow Problem to find a maximum matching. Beginning with the graph $G$ in an instance of the Bipartite Matching Problem, we construct a flow network $G'$ as shown in Figure 7.9. First we direct all edges in $G$ from $X$ to $Y$. We then add a node $s$, and an edge $(s, x)$ from $s$ to each node in $X$. We add a node $t$, and an edge $(y, t)$ from each node in $Y$ to $t$. Finally, we give each edge in $G'$ a capacity of 1.

We now compute a maximum $s$-$t$ flow in this network $G'$. We will discover that the value of this maximum is equal to the size of the maximum matching in $G$. Moreover, our analysis will show how one can use the flow itself to recover the matching.

### Algorithm

1. MAKE NEW $G'$ FROM INPUT $G$.
2. RUN FF ON $G'$
3. OUTPUT ALL MIDDLE EDGES WITH FLOW $F(E) = 1$. 

![Diagram of a network with labeled nodes and edges, illustrating the algorithm steps.]
correctness

IF $G$ HAS A MATCHING OF SIZE $K$, THEN $G'$ has a max flow of $K$.

**Proof**: Given a matching $M$ of size $K$, construct a flow $f$ which assigns 1 unit of flow to each edge $e$ in $M$, and 1 unit of flow from $(s,v)$ and $(v,s)$ for every $e = (u,v) \in M$.

- Show that $f$ is a valid flow.
  - Capacity constraint
  - Flow constraint (conservation)
**correctness**

**IF** $G'$ **HAS A FLOW OF** $K$, **THEN**

$G'$ **has a matching of size** $K$.

Consider all edges in $G'$ between $L$ and $R$ with $f(e) > 1$.

Add $e$ to $M$. => $|M| = K$, so $G$ has a $K$-matching.

$G'$ **has a flow of** 2.
integrality theorem

IF CAPACITIES ARE ALL INTEGRAL, THEN $F^t$ returns an integral flow.

Proof: by induction.

Base case: at start, $F^0$ has an integral flow (0)

Specify true after $i$ iterations.

On iteration $i$, flow is integral so residual capacities on all edges are integral. $F^i$ finds an augmenting path $p$, and the min capacity edge will therefore be integral. $\Rightarrow$ flow remains integral on iteration $i+1$.  


' HAS A FLOW OF \( k \), THEN \( G \) HAS \( k \)-MATCHING.

By (6), \( G' \) has integral capacities, the \( \text{FF} \) returns an integral flow.
Every edge has either \( f(e) = 0 \) or \( f(e) = 1 \).
Set \( M \) to be all edges \( b/w \) \( L \) and \( k \) \( w \) \( f(e) = 1 \). Can be at most \( K \) by MIN-CUT theorem.

Each node appears at most once in \( M \) by the conservation constraint.
running time

\( O(E \cdot f(I)) \sim O(E \cdot U) \)
edge-disjoint paths
Algorithm

Argue that this is correct

1. If $G$ has $K$ edge disjoint paths
   \[
   \Rightarrow G\text{ has a $K$ max flow}
   \]

2. By integrality, $f(e) \leq \{0, 1\}$
   \[
   \Rightarrow G\text{ has $K$ edge disjoint paths among all the edges with } f(e) = 1.
   \]

3. If $G$ has a $K$-max flow
   \[
   \Rightarrow G\text{ has $K$ edge disjoint paths}.
   \]
Argue that this is correct:

1. If $G$ has $k$ edge-disjoint paths
   $\Rightarrow$ $G$ has a $k$-max flow

2. If $G$ has a $k$-max flow
   $\Rightarrow$ $G$ has $k$ edge-disjoint paths

- By integrality, $f(e) \in \{0, 1\}$
- $\exists$ $k$ edge-disjoint paths among all the edges with $f(e) = 1$
1. Compute max flow 
2. Remove all edges with \( f(e) = 0 \). 
3. Walk from s. 
   1. If you reach a node you have visited before, erase flow along path 
   2. If you reach t, add this path to your set, erase flow along path. 

*Work by induction on the # of edges with \( f(e) = 1 \)*
IF $G$ has $k$ disjoint paths, then
IF $G'$ HAS A FLOW OF $K$, THEN
vertex-disjoint paths

1. Make $G'$ by substituting gadget for every node in $G$.
2. Compute edge-disjoint paths in $G'$.
3. Turn $f$ into the node-disjoint paths of $G'$. 
## baseball elimination

<table>
<thead>
<tr>
<th></th>
<th>W</th>
<th>L</th>
<th>Left</th>
<th>A</th>
<th>P</th>
<th>N</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATL</td>
<td>83</td>
<td>71</td>
<td>8</td>
<td>-</td>
<td>1</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>PHL</td>
<td>80</td>
<td>79</td>
<td>3</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>NY</td>
<td>78</td>
<td>78</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>MONT</td>
<td>77</td>
<td>82</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>
### Baseball Elimination

<table>
<thead>
<tr>
<th>Team</th>
<th>W</th>
<th>L</th>
<th>Left</th>
<th>N</th>
<th>B</th>
<th>Bo</th>
<th>T</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>NY</td>
<td>75</td>
<td>59</td>
<td>28</td>
<td>3</td>
<td>8</td>
<td>X</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>BAL</td>
<td>71</td>
<td>63</td>
<td>28</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>BOS</td>
<td>69</td>
<td>66</td>
<td>27</td>
<td>8</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOR</td>
<td>63</td>
<td>72</td>
<td>27</td>
<td>7</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DET</td>
<td>49</td>
<td>86</td>
<td>27</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Against

\[ \sqrt{30.5} = \frac{76.25}{25} \]

\[ \frac{75}{71} = \frac{71}{69} = \frac{63}{27 + 8} = \frac{27}{30.5} \]
<table>
<thead>
<tr>
<th></th>
<th>W</th>
<th>L</th>
<th>Left</th>
<th>N</th>
<th>B</th>
<th>Bo</th>
<th>T</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>NY</td>
<td>75</td>
<td>59</td>
<td>28</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>BAL</td>
<td>71</td>
<td>63</td>
<td>28</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>BOS</td>
<td>69</td>
<td>66</td>
<td>27</td>
<td>8</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOR</td>
<td>63</td>
<td>72</td>
<td>27</td>
<td>7</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DET</td>
<td>49</td>
<td>86</td>
<td>27</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Gabriel García Marquez

Love in the Time of Tinder
We have a group of suitors and reviewers
Each has preferences over the other group.
We seek a stable matching between the two.
Unstable Matching

G1 prefers 2 to 3
B2 prefers 1 to 2
Stable Matching
Stable matching has many practical applications
The National Resident Matching Program (NRMP) is a private, not-for-profit corporation established in 1952 to optimize the rank-ordered choices of applicants and program directors. The NRMP is not an application processing service; rather, it provides an impartial venue for matching applicants' and programs' preferences for each other consistently.

The first Main Residency Match® ("the Match") was conducted in 1952 when 10,400 internship positions were available for 6,000 U.S. graduating seniors. By 1973, there were 19,000 positions for just over 10,000 U.S. graduating seniors. Following the demise of internships in 1975, the number of first-year post-graduate (PGY-1) positions dropped to 15,700. The number of PGY-1 positions offered gradually increased through 1994 and then began to decline slowly until 1998. This year saw a record-high 26,678 PGY-1 positions offered (Figure 1), marking the twelfth consecutive annual increase in such positions.

The trend in the total number of applicants since 1952 is more dramatic, starting with 6,000 in 1952 and rising to a high of 36,056 in 1999. After a decline of 5,052 applicants from 1999 to 2003, the number of applicants has increased each year since the 2004 Match. Applicants registered for the 2014 Match reached an all time high of 40,394, an increase of 59 applicants over 2013.

For more information about the NRMP, please visit: www.nrmp.org. Additional data and reports for the Main Residency Match and the Specialties Matching Service® (SMS®) can be found at: www.nrmp.org/match-data. Instructions on how to request NRMP data also are provided.
<table>
<thead>
<tr>
<th>Applicant Type</th>
<th>2013 Graduates</th>
<th>Prior Year Graduates¹</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMG</td>
<td>2571</td>
<td>74</td>
<td>2645</td>
</tr>
<tr>
<td>IMG</td>
<td>146</td>
<td>353</td>
<td>499</td>
</tr>
<tr>
<td>USMG</td>
<td>23</td>
<td>2</td>
<td>25</td>
</tr>
<tr>
<td>TOTAL</td>
<td>2740</td>
<td>429</td>
<td>3169</td>
</tr>
</tbody>
</table>

Keywords: Summary, match results, 1st and 2nd iteration combined, prior year, current year, unmatched, final participation

Note 1: Includes graduates from all years prior to 2013.
Definition: matchings

\[ M = \]
\[ W = \]
\[ S = \]
Definition: matchings

\[ M = \{ m_1, \ldots, m_n \} \]
\[ W = \{ w_1, \ldots, w_n \} \]
\[ S = \{ (m_{i_1}, w_{j_1}), \ldots, (m_{i_k}, w_{i_k}) \} \]

Each \( m_i \) (or \( w_i \)) appears only one in a pairing.
A matching is perfect if every \( m_i \) appears.
Definition: preferences

\[ M = \{m_1, \ldots, m_n\} \]
Example: preferences

\[ M = \{m_1, \ldots, m_n\} \]

\( m_i \) has a preference relation \( \prec m_i \) on the set \( W \)

\( w_1 \prec m_i \) \( w_4 \prec m_i \) \( w_2 \prec m_i \) \( w_8 \cdots w_n \)
Def: instability

$$S = \left\{ \begin{array}{c}
(\text{🐶} \hspace{1em} \text{Harvard}) \\
(\text{ Bulls } \hspace{1em} \text{ Yale})
\end{array} \right\}$$
Def: instability

\[ S = \left\{ \left( \begin{array}{c} w' \\ w^* \end{array} \right), \left( \begin{array}{c} m^* \\ w^* \end{array} \right) \right\} \]

\[ (m^*, w^*) \notin S \]

\[ w' \prec_{m^*} w^* \]

\[ m' \prec_{w^*} m^* \]
\[ M = \{ (s_1, r_1), (s_2, r_2), \ldots (s_n, r_n) \} \]

is a stable matching if

No unmatched pair \((s^*, r^*)\) prefer each other to their partners in \(M\).
Example 2
Prove: for every input there exists a stable matching.
proposal algorithm
StableMatch($M, W, \prec_m, \prec_w$)

1. Initialize all $m, w$ to be free
2. While $\exists \text{free}(m)$ and hasn’t proposed to all $W$
   - Pick such an $m$
     - Let $w \in W$ be highest-ranked to whom $m$ has not yet proposed
       - If $\text{free}(w)$
         - Then Make a new pair $(m, w)$
       - Elseif $(m', w)$ is paired and $m' \prec_w m$
         - Do Break pair $(m', w)$ and make $m'$ free
           - Make pair $(m, w)$
3. Return Set of pairs
<table>
<thead>
<tr>
<th>S</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Proposal algorithm ends
proposal algorithm ends

$O(n^2)$ steps

each $m$ proposes at most once to each $w$.
each $m$ proposes at most $n$ times.
size of $M$ is $n$. 

output is a matching
StableMatch\((M, W, <_m, <_w)\)

1. Initialize all \(m, w\) to be free
2. while \(\exists \text{FREE}(m)\) and hasn’t proposed to all \(W\)
   
   do Pick such an \(m\)

   Let \(w \in W\) be highest-ranked to whom \(m\) has not yet proposed

   if \(\text{FREE}(w)\)

   then Make a new pair \((m, w)\)

   elseif \((m', w)\) is paired and \(m' <_w m\)

   do Break pair \((m', w)\) and make \(m'\) free

   Make pair \((m, w)\)
3. return Set of pairs
\textbf{StableMatch}(M, W, \prec_m, \prec_w)

1. Initialize all \( m, w \) to be \texttt{free}

2. \textbf{while} \( \exists \texttt{FREE}(m) \) and hasn’t proposed to all \( W \)

   \hspace{1em} do \hspace{1em} Pick such an \( m \)

   \hspace{2em} let \( w \in W \) be highest-ranked to whom \( m \) has not yet proposed

   \hspace{3em} \textbf{if} \texttt{FREE}(w)

   \hspace{4em} \textbf{then} \hspace{1em} Make a new pair \((m, w)\)

   \hspace{3em} \textbf{elseif} \ ((m', w)) \text{ is paired and } m' \prec_w m

   \hspace{5em} \textbf{do} \hspace{1em} Break pair \((m', w)\) and make \( m' \) free

   \hspace{4em} \hspace{1em} \textbf{Make pair} \((m, w)\)

3. \textbf{return} Set of pairs
output is perfect
output is perfect

if \( \exists m \) who is free, then

\( \exists w \) who has not been asked
output is stable
output is stable

\[ \exists (m^*, w), (m, w^*) \in S \quad w \prec_{m^*} w^* \quad m \prec_{w^*} m^* \]
output is stable

spse not. \( \exists (m^*, w), (m, w^*) \in S \quad w \prec_{m^*} w^* \quad m \prec_{w^*} m^* \)

\( m^* \) last proposal was to \( w \)
but \( w \prec_{m^*} w^* \) and so \( m^* \) must have already asked \( w^* \)
and must have been rejected by \( m^* \prec_{w^*} m' \)
then either \( m' \prec_{w^*} m \) or \( m' = m \)
which contradicts assumption \( m \prec_{w^*} m^* \)
Proposer wins
Proposer wins
Remarkable theorem

w is valid for m:

best(m):
GS is Suitor-optimal.
GS matching vs R-opt