RECAP: For $G$ bipartite and $M \subseteq E(G)$ matching
- a vertex is free if it doesn't belong to an edge of $M$, else it is matched.
- an $M$-alternating path (or cycle) is one whose edges alternate between belonging to $M$ and not belonging to $M$.
- an $M$-augmenting path is an $M$-alternating path from a free vertex to another free vertex.

Algorithm (partly specified) for Max Matching

\[
\begin{align*}
M & \leftarrow \emptyset \\
\text{while} & \exists M\text{-augmenting path } P \\
M & \leftarrow M + P \\
\text{Endwhile} \\
\text{output} & \ M
\end{align*}
\]

(§2) For general graphs (not only bipartite) we can find $P$ in $O(mn)$ time.
Last time we saw this takes \( \leq \frac{n}{2} \) iterations
\((n = \# \text{ vertices}, \quad m = \# \text{ edges})\)
and we saw it correctly outputs a
max matching.

For bipartite \( G \), we can find \( M \)-augmenting \( P \)
if it exists in linear time.

**Def.** If \( G = (L + R, E) \) is bipartite
and \( M \) is a matching, the
residual graph \( G_M \) has vertices \( L + R \)
and directed edges
\[
\begin{align*}
\{(u,v) &\mid u \in L, v \in R, (u,v) \notin M\} \\
\cup \{(v,u) &\mid u \in L, v \in R, (v,u) \in M\}
\end{align*}
\]
\{ M\text{-augmenting paths in } G \} \quad F \triangleq \{ \text{free vertices} \}

\{ \text{directed paths in } G_m \text{ from } L\cap F \text{ to } R\cap F \}

BFS finds a dir. path in \( G_m \) from \( L\cap F \) to \( R\cap F \) in time \( O(m+n) \) if one exists.

\( O(n) \) loop iterations, \( O(m+n) \) time per iteration

\( \Rightarrow \quad O(mn + n^2) \) time altogether.

Hopcroft & Karp improved this to \( O(mn\sqrt{n}) \).

(§14 if notes to be presented in \( n \) 2 weeks)

Minimum Cost Bipartite Perfect Matching.

Input: \( G = (L \cup R, E) \) bipartite

\[ c: E \to \mathbb{R} \]

Convention: \( c(u,v) = \infty \) if \( (u,v) \notin E \).

If \( G^+ \) denotes the complete bipartite graph on vertex sets \( L, R \) and \( M \) is a matching in \( G^+ \)
Problem. Find a perfect matching in $G^+$ with minimum combined edge cost.

Running time will be expressed in terms of

$n = \#\text{ vertices of } G$

$m = \#\text{ edges of } G$.

Idea: Start with empty matching $M_0$.

Build up a sequence of matchings $M_0, M_1, \ldots, M_k$ using one augmenting path at a time.

$M_{k+1}$ will be equal to $M_k + P_k$

where $P_k$ is some (carefully chosen) $M_k$-augmenting path.

How does the cost of a matching change when we replace $M$ with $M + P$?
\[ \begin{align*}
\text{cost}(M) & = 1 + 1 \\
\text{cost}(M \oplus \rho) & = 3 + 4 + 5 \\
\Delta c(P, M) & = \text{cost}(M \oplus \rho) - \text{cost}(M) \\
& = 3 - 1 + 4 - 1 + 5 \\
& = 10
\end{align*} \]

If we assign edge cost to directed edges in \( G_M \) as follows:

\[ \text{cost}(u,v) = \begin{cases} 
\text{c}(u,v) & \text{if } (u,v) \notin M \\
-\text{c}(u,v) & \text{if } (v,u) \in M 
\end{cases} \]

then the bijection

\[ \{M\text{-augmenting paths}\} \leftrightarrow \{ \text{directed paths in } G_M \text{ from } L(F) \text{ to } R(F) \} \]

is \( \text{cost} \)-preserving.
GREEDY ALGORITHM

\[ M_0 \leftarrow \emptyset \]

for \( k = 0, \ldots, \frac{n}{2} - 1 \)
    compute \( G_M \) with its edge costs defined as above.
    \[ P_k = \text{min cut directed path from } \text{LoF to } \text{RoF} \]
    \[ M_{k+1} = M_k \odot P_k \]
end for

output \( M_{\frac{n}{2}} \).

Correct? Yes, but surprisingly subtle to prove.

Induction hypothesis: \( M_k \) has the least cost of all matchings with \( k \) edges.

(See lecture notes for a self-contained proof of induction step.)

Efficient? Use Bellman-Ford. Finding \( P_k \) takes time \( O(mn) \).... if \( G_M \) has no negative cost cycles!
If \( M' = M_k \) had a negative cost cycle \( C \) in \( \Gamma_M \), then consider \( M' = M + C \).

Note \( C \) is an even length \( M \)-alternating cycle. So \( M' \) is a matching which also has \( k \) edges.

\[
\text{cost}(C) = \text{cost}(M') - \text{cost}(M).
\]

By induction hypothesis above, \( M = M_k \) has the least cost among all \( k \)-edge matchings.

\[
\therefore \quad \text{cost}(M') \geq \text{cost}(M)
\]

\[
\therefore \quad \text{cost}(C) \geq 0
\]

\[
\therefore \quad \text{we're allowed to use Bellman-Ford to find } p_k.
\]

Running time: \( O(mn^2) \).

\[
\frac{n^2}{2} \text{ by iterations, } O(mn) \text{ per iteration.}
\]

Next lecture:
1. Make this faster using Dijkstra in place of Bellman-Ford.
2. Prove the algorithm is correct.

Proving correctness holds the key to speeding up the algorithm.