CSE3081 Design and Analysis of Algorithms

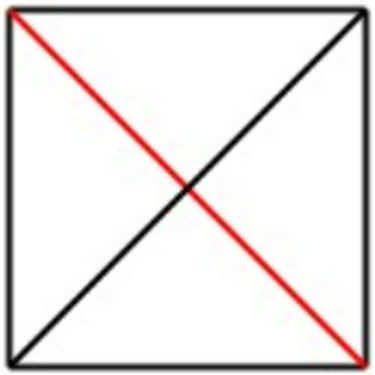
Dept. of Computer Engineering,
Sogang University

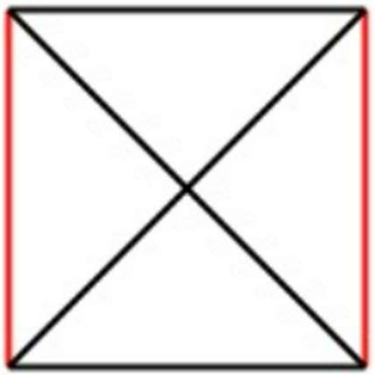
Chapter 25. Matchings in Bipartite Graphs

Overview

- Many real-world problems can be modeled as finding matchings in an undirected graph.

- For an undirected graph $G = (V, E)$, a matching is a subset of edges $M \subseteq E$ such that every vertex in V has at most one incident edge in M.





Real-World Example of Matching

- You have one or more positions to fill and several candidates to interview.
- You are able to interview candidates at certain time slots.
- Candidates indicate the subsets of time slots at which they are available.
- How can you schedule the interviews so that each time slot has at most one candidate scheduled, while maximizing the number of candidates that you can interview?

- You can model this scenario as a matching problem on a bipartite graph.
- Each vertex represents either a candidate or a time slot
- An edge exists between a candidate and a time slot if the candidate is available then.

- An edge in the matching means you are interviewing the candidate at the time slot.
- Your goal is to find a maximum matching.

Real-World Example of Matching

- U.S. National Resident Matching Program assigns medical students to hospitals where they will be stationed as medical students.
- Each student ranks the hospitals by preference, and each hospital ranks the students.
- The goal is to assign students to hospitals so that there is never a student and a hospital that both have regrets because the student was not assigned to the hospital, yet each ranked the other higher than who or where they were assigned.

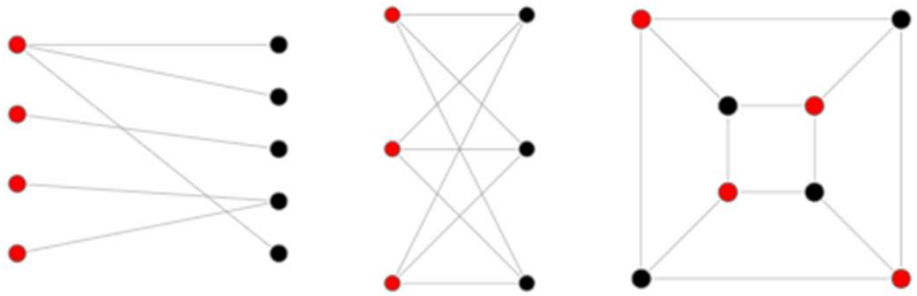
Real-World Example of Matching

- Workers must be assigned to tasks in order to maximize the overall effectiveness of the assignment.
- For each worker and each task, the worker has some quantified effectiveness for that task.
- Assuming there are equal number of workers and tasks, the goal is to find a matching with the maximum total effectiveness.

Bipartite Graphs

- The algorithms in this chapter find matchings in bipartite graphs.
- The input is an undirected graph $G = (V, E)$, where $V = L \cup R$, the vertex sets L and R are disjoint, and every edge in E is incident on one vertex in L and one vertex in R.

- A matching, therefore, matches vertices in L with vertices in R.
- In some applications, the sets L and R have equal cardinality, and in other applications they need not be the same size.



Matching in non-bipartite graphs

- Some problems can be modeled as matching in "general" undirected graphs.
  - The input may not be a bipartite graph.
  - e.g.) scheduling, computational chemistry

- Matching in general graphs models problems in which you want to pair up entities represented by vertices.

- Two vertices are adjacent if they represent compatible entities, and you need to find a large set of compatible pairs.

- There are polynomial-time algorithms that can solve maximum matching in general graphs, but they are more complicated than algorithms for solving maximum matching in bipartite graphs.

25.1 Maximum Bipartite Matching (Revisited)

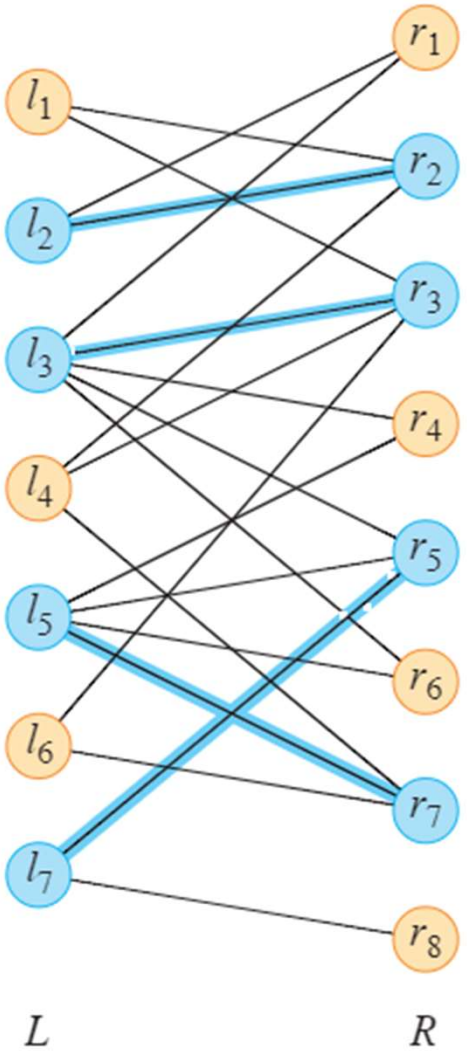
Introduction

- In Section 24.3, we learned that maximum bipartite matching can be solved by finding a maximum flow. The time complexity of using Ford-Fulkerson method to find maximum bipartite matching was $O(VE)$.

- In this section, we learn Hopcroft-Karp algorithm, which runs in $O(\sqrt{V}E)$ time.

Matching in an Undirected Bipartite Graph

- A vertex that has an incident edge in matching M is matched under M, and otherwise it is unmatched.
- A maximal matching is a matching M to which no other edges can be added.
  - For every edge $e \in E - M$, the edge set $M \cup \{e\}$ fails to be a matching.
- A maximum matching is always maximal, but the reverse may not always hold.



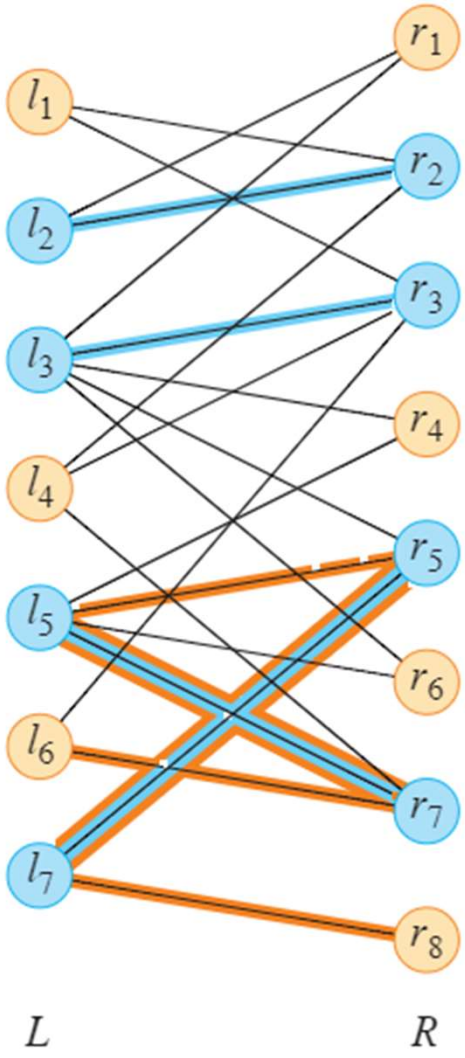
Approach to Finding Maximum Matching

- Many algorithms, including the Hopcroft-Karp algorithm, work by incrementally increasing the size of a matching.

- Given a matching M in an undirected graph $G = (V, E)$, an M-alternating path is a simple path whose edges alternate between being M and being in $E - M$.

- An M-augmenting path is an M-alternating path whose first and last edges belong to $E - M$.

- Since an M-augmenting path contains one more edge in $E - M$ than in M, it must consist of an odd number of edges.

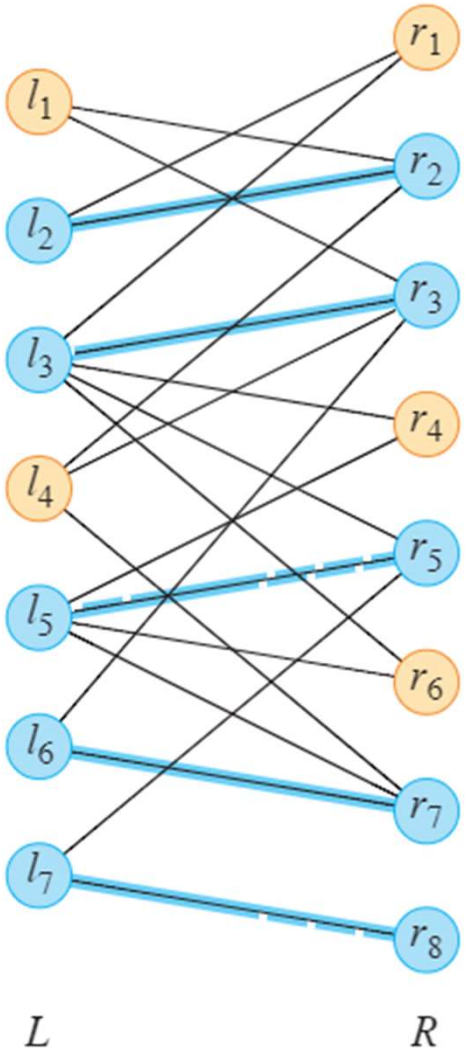


Approach to Finding Maximum Matching

- By removing from matching M the edges in an M-augmenting path that are not in M, the result is a new matching with one more edge than M.

- Lemma 25.1 proves this argument.

- Symmetric Difference
  - $X \oplus Y = (X - Y) \cup (Y - X) = (\mathrm{X} \cup Y) - (X \cap Y)$
  - elements that belong to X or Y, but not both.
  - The operator $\oplus$ is commutative and associative.
  - $X \oplus X = \emptyset$
  - $X \oplus \emptyset = \emptyset \oplus X = X$



Lemma 25.1

- Let M be a matching in any undirected graph $G = (V, E)$, and let P be an M-augmenting path. Then the set of edges $M' = M \oplus P$ is also a matching in G with $|M'| = |M| + 1$.

  - Let P contain q edges, so that $\lceil q/2 \rceil$ edges belong to $E - M$ and $\lceil q/2 \rceil$ edges belong to M, and let these q edges be $(v_1, v_2), (v_2, v_3), \ldots, (v_q, v_{q+1})$.
  - Because P is an M-augmenting path, vertices v_1 and v_{q+1} are unmatched under M and all other vertices in P are matched.
  - Edges $(v_1, v_2), (v_3, v_4), \ldots, (v_q, v_{q+1})$ belong to $E - M$, and edges $(v_2, v_3), (v_4, v_5), \ldots, (v_{q-1}, v_q)$ belong to M.
  - The symmetric difference $M' = M \oplus P$ reverses these roles, so that edges $(v_1, v_2), (v_3, v_4), \ldots, (v_q, v_{q+1})$ belongs to M' and $(v_2, v_3), (v_4, v_5), \ldots, (v_{q-1}, v_q)$ belong to $E - M'$.
  - Each vertex $v_1, v_2, \ldots, v_q, v_{q+1}$ is matched under M', which gains one additional edge relative to M, and no other vertices or edges in G are affected by the change from M to M'.
  - Hence, M' is matching in G, and $|M'| = |M| + 1$.

Corollary 25.2: Intro

- Since taking the symmetric difference of a matching M with an M-augmenting path increases the size of the matching by 1, Corollary 25.2 shows that taking the symmetric difference of M with k vertex-disjoint M-augmenting paths increases the size of the matching by k.

Corollary 25.2

- Let M be a matching in any undirected graph $G = (V, E)$ and $P_1, P_2, \ldots, P_k$ be vertex-disjoint M-augmenting paths. Then the set of edges $M' = M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k)$ is a matching in G with $|M'| = |M| + k$.

  - Since the M-augmenting paths $P_1, P_2, \ldots, P_k$ are vertex disjoint, we have that $P_1 \cup P_2 \cup \cdots \cup P_k = P_1 \oplus P_2 \oplus \cdots \oplus P_k$.
  - Because the operator $\oplus$ is associative, we have

$$\begin{aligned} M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k) &= M \oplus (P_1 \oplus P_2 \oplus \cdots \oplus P_k) \\ &= (\cdots((M \oplus P_1) \oplus P_2) \oplus \cdots \oplus P_{k-1}) \oplus P_k \ . \end{aligned}$$

  - A simple induction on i using Lemma 25.1 shows that $M \oplus (P_1 \cup P_2 \cup \cdots \cup P_{i-1})$ is a matching in G containing $|M| + i - 1$ edges and that path P_i is an augmenting path with respect to $M \oplus (P_1 \cup P_2 \cup \cdots \cup P_{i-1})$.
  - Each of these augmenting paths increases the size of the matching by 1, and so $|M'| = |M| + k$.

Lemma 25.3: Intro

- As the Hopcroft-Karp algorithm goes from matching to matching, it is useful to consider the symmetric difference between two matchings.

Lemma 25.3

- Let M and M^* be matchings in graph $G = (V, E)$, and consider the graph $G' = (V, E')$, where $E' = M \oplus M^*$. Then, G' is a disjoint union of simple paths, simple cycles, and/or isolated vertices. The edges in each such simple path or simple cycle alternate between M and M^*. If $|M^*| > |M|$, then G' contains at least $|M^*| - |M|$ vertex-disjoint M-augmenting paths.

  - Each vertex in G' has degree 0, 1, or 2, since at most two edges of E' can be incident on a vertex: at most one edge from M and at most one edge from M^*.
  - Therefore, each connected component of G' is either a singleton vertex, an even length simple cycle with edges alternately in M and M^*, or a simple path with edges alternately in M and M^*.
  - Since $E' = M \oplus M^* = (M \cup M^*) - (M \cap M^*)$ and $|M^*| > |M|$, the edge set E' must contain $|M^*| - |M|$ more edges from M^* than from M.
  - Because each cycle in G' has an even number of edges drawn alternately from M and M^*, each cycle has an equal number of edges from M and M^*.
  - Therefore, the simple paths in G' account for there being $|M^*| - |M|$ more edges from M^* than M.

Lemma 25.3

- Each path containing a different number of edges from M and M^* either starts and ends with edges from M, containing one more edge from M than from M^*, or starts and ends with edges from M^*, containing one more edge from M^* than from M.
- Because E' contains $|M^*| - |M|$ more edges from M^* than from M, there are at least $|M^*| - |M|$ paths of the latter type, and each one is an M-augmenting path.
- Because each vertex has at most two incident edges from E', these paths must be vertex-disjoint.

Corollary 25.4: Intro

- If an algorithm finds a maximum matching by incrementally increasing the size of the matching, how does it determine when to stop?

- Corollary 25.4 gives the answer: when there are no augmenting paths.

Corollary 25.4

- Matching M in graph $G = (V, E)$ is a maximum matching if and only if G contains no M-augmenting path.

  – (contrapositive of the forward direction) If there is an M-augmenting path P in G, then by lemma 25.1, the matching $M \oplus P$ contains one more edge than M, meaning that M cannot be a maximum matching.

  – (contrapositive of the backward direction) Let M^* be a maximum matching in Lemma 25.3, so that $|M^*| > |M|$. Then, G contains at least $|M^*| - |M| > 0$ vertex-disjoint M-augmenting paths.

The Hopcroft-Karp Algorithm

- The algorithm starts with the matching M empty. Then, it repeatedly runs a variant of either breadth-first search or depth-first search from an unmatched vertex that takes alternating paths until you find another matched vertex. It uses the resulting M-augmenting path to increase the size of M by 1.

```
HOPCROFT-KARP(G)
1  M = ∅
2  repeat
3      let P = {P_1, P_2, ..., P_k} be a maximal set of vertex-disjoint
           shortest M-augmenting paths
4      M = M ⊕ (P_1 ∪ P_2 ∪ ··· ∪ P_k)
5  until P == ∅
6  return M
```

The Hopcroft-Karp Algorithm

- The algorithm runs in $O(\sqrt{V}E)$ time.
- The repeat loop of lines 2-5 iterates $O(\sqrt{V})$ times.
- Line 3 runs in $O(E)$ time in each iteration.

HOPCROFT-KARP(G)

1 $M = \emptyset$
2 **repeat**
3 let $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ be a maximal set of vertex-disjoint shortest M-augmenting paths
4 $M = M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k)$
5 **until** $\mathcal{P} == \emptyset$
6 **return** M

Running Time of Hopcroft-Karp Algorithm

- We first look at how to find a maximal set of vertex-disjoint shortest M-augmenting paths. There are three phases.

- Phase 1: Form a directed version G_M of the undirected bipartite graph G.

- Phase 2: Create a directed acyclic graph H from G_M via a variant of breadth-first search.

- Phase 3: Find a maximal set of vertex-disjoint shortest M-augmenting paths by running a variant of depth-first search on the transpose H^T of H.
  - Transpose of a directed graph reverses the direction of each edge.
  - Since H is acyclic, so is H^T.

Phase 1

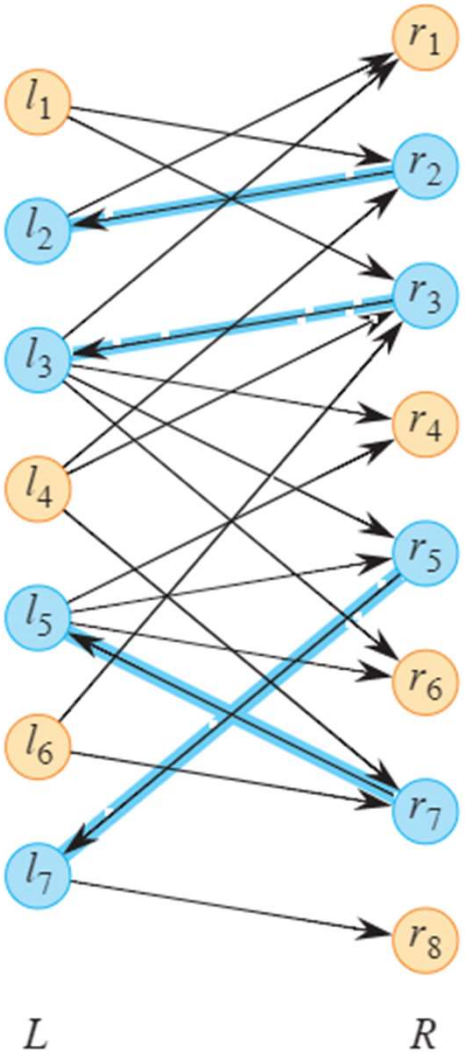
- Given a matching M, we can think of an M-augmenting path P as starting at an unmatched vertex in L, traversing an odd number of edges, and ending at an unmatched vertex in R.
- The edges in P traversed from L to R must belong to $E - M$, and the edges in P traversed from R to L must belong to M.
- Therefore, we create the directed graph G_M as follows:

- $G_M = (V, E_M)$, where

$$E_M = \{(l, r) : l \in L, r \in R, \text{ and } (l, r) \in E - M\} \quad \text{(edges from } L \text{ to } R\text{)}$$
$$\cup \{(r, l) : r \in R, l \in L, \text{ and } (l, r) \in M\} \quad \text{(edges from } R \text{ to } L\text{)}\ .$$

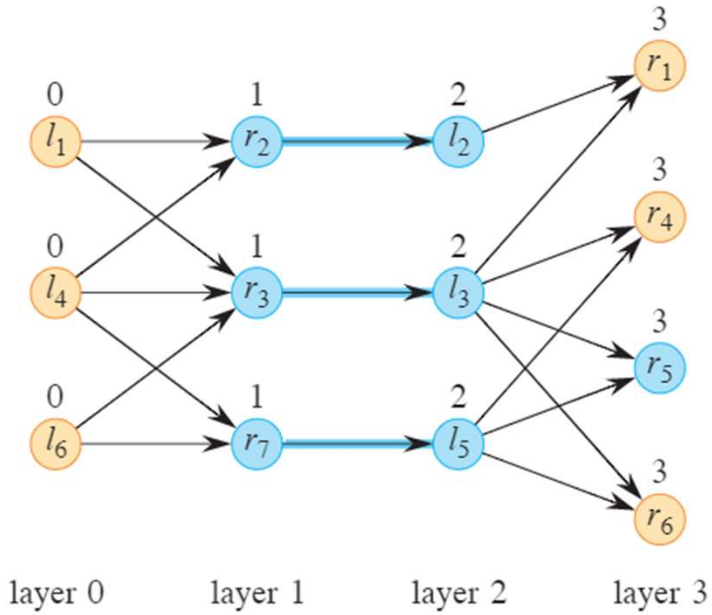
Phase 1

- Directed graph G_M created from phase 1.



Phase 2

- The DAG $H = (V_H, E_H)$ created by the second phase has layers of vertices.
- Each layer contains only vertices from L or only vertices from R, alternating from layer to layer.
- The layer that a vertex resides in is given by that vertex's minimum breadth-first search distance in G_M from any unmatched vertex in L.
- Vertices in L appear in even-numbered layers, and vertices in R appear in odd-numbered layers.

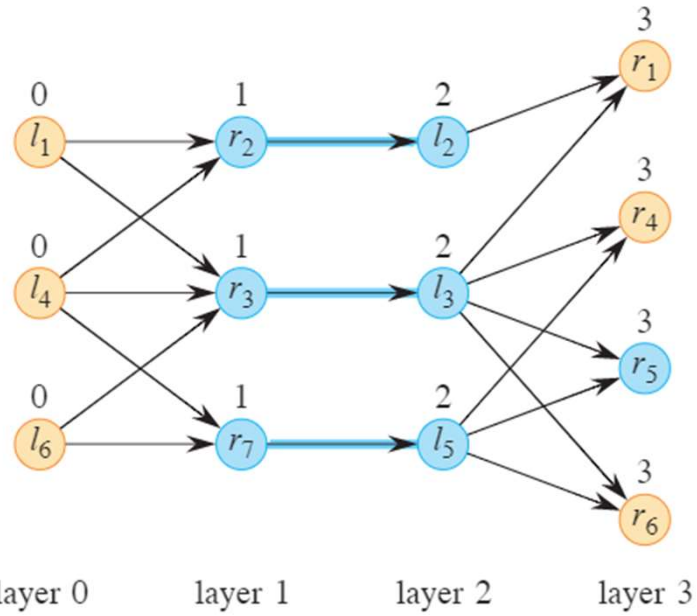


Phase 2

- Let q denote the smallest distance in G_M of any unmatched vertex in R.
- Then, the last layer in H contains the vertices in R with distance q.
- Vertices whose distance exceeds q do not appear in V_H.
  - Vertices l_7 and r_8 are omitted.
- The edges in E_H form a subset of E_M:

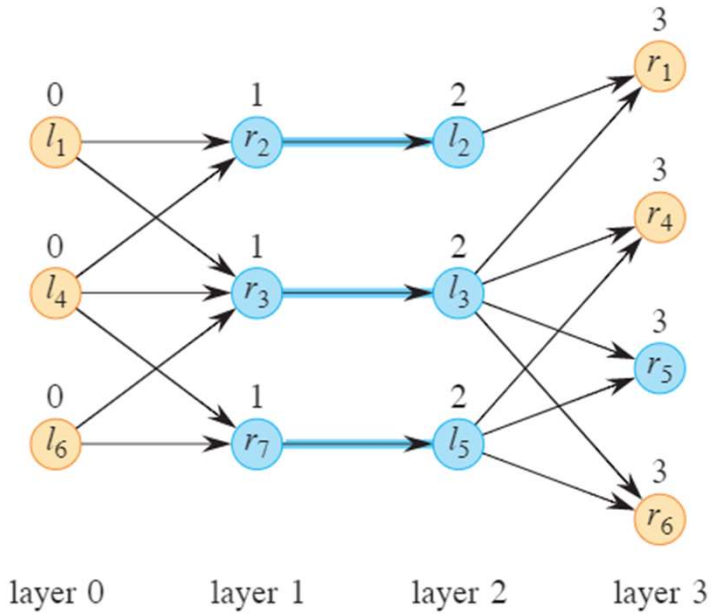
$$E_H = \{(l, r) \in E_M : r.d \leq q \text{ and } r.d = l.d + 1\} \cup \{(r, l) \in E_M : l.d \leq q\}$$

  - where d is the vertex's breadth-first distance in G_M from any unmatched vertex in L.
- Edges that do not go between two consecutive layers are omitted from E_H.



Phase 2

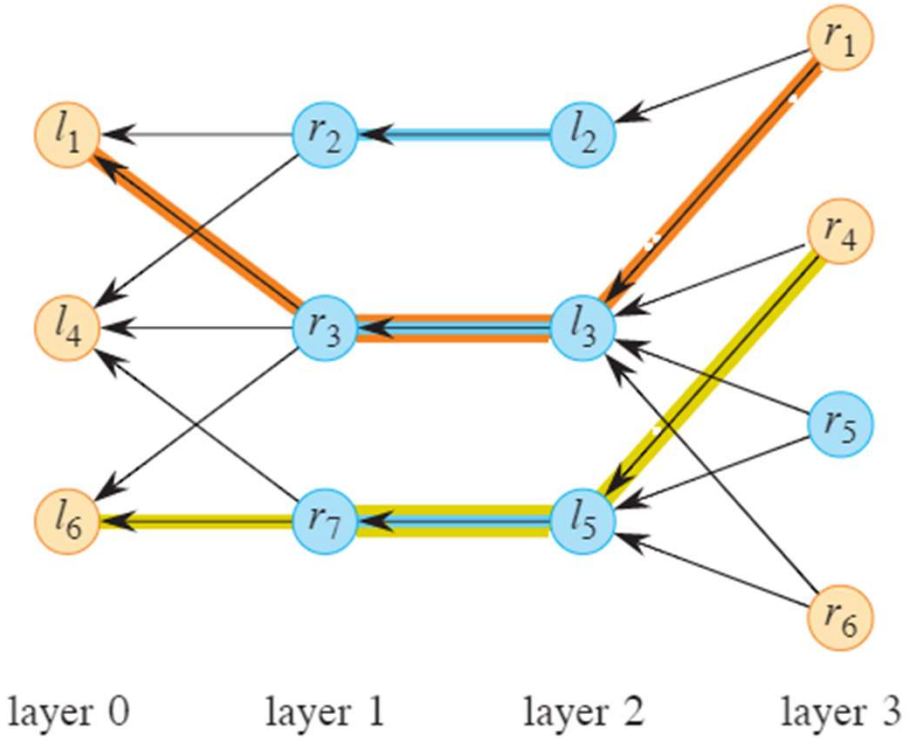
- To determine the breadth-first distance of vertices, run breadth-first search on the graph G_M, but starting from all the unmatched vertices in L.
- Every path in H from a vertex in layer 0 to an unmatched vertex in layer q corresponds to a shortest M-augmenting path in the original bipartite graph G. (Just use the undirected versions of the directed edges in H.)
- Moreover, every shortest M-augmenting path in G is present in H.



Phase 3

- This phase identifies a maximal set of vertex-disjoint shortest M-augmenting paths.

- We start by creating a transpose H^T of H.

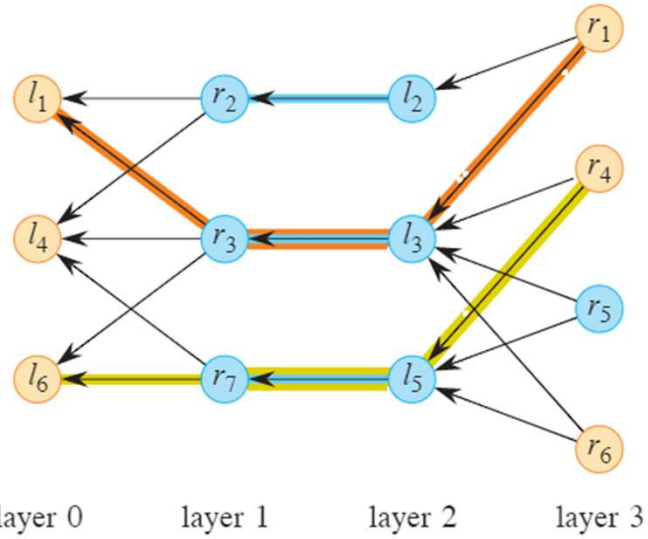


Phase 3

- Then, for each unmatched vertex r in layer q, it performs a depth-first search starting from r until it either reaches a vertex in layer 0 or has exhausted all possible paths without reaching a vertex in layer 0.
- The depth-first search needs to keep track of the predecessor attributes π in the depth-first tree of each search.
- Upon reaching a vertex in layer 0, tracing back along the predecessors identifies an M-augmenting path.
- Each vertex is searched from only when it is first discovered in any search.
- If the search from a vertex r in layer q cannot find a path of undiscovered vertices to an undiscovered vertex in layer 0, then no M-augmenting path including r goes into the maximal set.

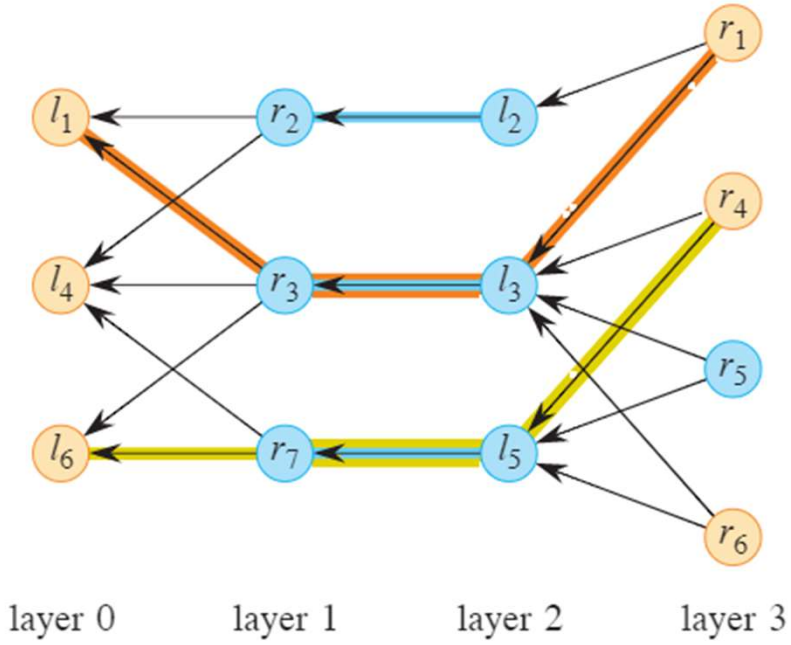
Phase 3

- The first depth-first search starts from vertex r_1. It identifies the M-augmenting path $\langle (r_1, l_3), (l_3, r_3), (r_3, l_1) \rangle$.
- The next depth-first search starts from vertex r_4. This search first examines the edge (r_4, l_3), but because l_3 was already discovered, backtracks and examines edge (r_4, l_5). It identifies $\langle (r_4, l_5), (l_5, r_7), (r_7, l_6) \rangle$.
- The next depth-first search starting from r_6 fails to find a path of undiscovered vertices to a vertex in layer 0.
- Therefore, the maximal set of vertex-disjoint shortest M-augmenting paths contains two paths, $\langle (r_1, l_3), (l_3, r_3), (r_3, l_1) \rangle$ and $\langle (r_4, l_5), (l_5, r_7), (r_7, l_6) \rangle$.



Phase 3

- Note that this maximal set is not a maximum set.
- The maximum set is $\langle (r_1, l_2), (l_2, r_2), (r_2, l_1) \rangle$, $\langle (r_4, l_3), (l_3, r_3), (r_3, l_4) \rangle$, $\langle (r_6, l_5), (l_5, r_7), (r_7, l_6) \rangle$.
- Still, the algorithm requires to find a maximal set, not a maximum set.



Running Time of the Three Phases

- We assume that in the original bipartite graph G, each vertex has at least one incident edge so that $|V| = O(E)$, which implies $|V| + |E| = O(E)$.
- The first phase creates the directed graph G_M by simply directing each edge of G, so that $|V_M| = |V|$ and $|E_M| = |E|$.
- The second phase performs a breadth-first search on G_M, taking $O(V_M + E_M) = O(E_M) = O(E)$ time.
- The third phase performs depth-first searches from the unmatched vertices in layer q. Once a vertex is discovered, it is not searched from again, and so the running time is $O(V_H + E_H) = O(E)$.
- Hence, all three phases take just $O(E)$ time.

- Once the maximal set of vertex-disjoint shortest M-augmenting paths have been found in line 3, updating the matching in line 4 takes $O(E)$ time, as it is just a matter of going through the edges of the M-augmenting paths and adding edges to and removing edges from the matching M.

The Number of Iterations of the Repeat Loop

- It remains to show that the repeat loop iterates $O(\sqrt{V})$ times.

- Lemma 25.5 shows that after each iteration of the repeat loop, the length of an augmenting path increases.

Lemma 25.5

- Let $G = (V, E)$ be an undirected bipartite graph with matching M, and let q be the length of a shortest M-augmenting path. Let $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ be a maximal set of vertex-disjoint M-augmenting paths of length q. Let $M' = M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k)$, and suppose that P is a shortest M'-augmenting path. Then, P has more than q edges.

- We consider separately the cases in which P is vertex-disjoint from the augmenting paths in $\mathcal{P}$ and in which it is not vertex-disjoint.
  - First, assume that P is vertex-disjoint from the augmenting paths in $\mathcal{P}$. Then, P contains edges that are in M but are not in any of $P_1, P_2, \ldots, P_k$, so that P is also an M-augmenting path.
  - Since P is disjoint from $P_1, P_2, \ldots, P_k$ but is also an M-augmenting path, and since $\mathcal{P}$ is a maximal set of shortest M-augmenting paths, P must be longer than any of the augmenting paths in $\mathcal{P}$, each of which has length q.
  - Therefore, P has more than q edges.

Lemma 25.5

- Now, assume that P visits at least one vertex from the M-augmenting paths in $\mathcal{P}$.
- By Corollary 25.2, M' is a matching in G with $|M'| = |M| + k$.
- Since P is an M'-augmenting path, by Lemma 25.1, $M' \oplus P$ is a matching with $|M' \oplus P| = |M'| + 1 = |M| + k + 1$.
- Now let $A = M \oplus M' \oplus P$. We claim that $A = (P_1 \cup P_2 \cup \cdots \cup P_k) \oplus P$:

$$
\begin{aligned}
A &= M \oplus M' \oplus P \\
&= M \oplus (M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k)) \oplus P \\
&= (M \oplus M) \oplus (P_1 \cup P_2 \cup \cdots \cup P_k) \oplus P && \text{(associativity of } \oplus\text{)} \\
&= \emptyset \oplus (P_1 \cup P_2 \cup \cdots \cup P_k) \oplus P && (X \oplus X = \emptyset \text{ for all } X) \\
&= (P_1 \cup P_2 \cup \cdots \cup P_k) \oplus P && (\emptyset \oplus X = X \text{ for all } X)\,.
\end{aligned}
$$

- Lemma 25.3 with $M^* = M' \oplus P$ gives that A contains at least $|M' \oplus P| - |M| = k + 1$ vertex-disjoint M-augmenting paths.
- Since each such M-augmenting path has at least q edges, we have $|A| \geq (k+1)q = kq + q$.

Lemma 25.5

- Now we claim that P shares at least one edge with some M-augmenting path in $\mathcal{P}$.
- Under the matching M', every vertex in each M-augmenting path in $\mathcal{P}$ is matched.
  - Only the first and last vertex in each M-augmenting path P_i is unmatched under M, and under $M \oplus P_i$, all vertices in P_i are matched.
  - Because the M-augmenting paths in $\mathcal{P}$ are vertex-disjoint, no other path in $\mathcal{P}$ can affect whether the vertices in P_i are matched.
  - That is, the vertices in P_i are matched under $(M \oplus P_i) \oplus P_j$ if and only if they are matched under $M \oplus P_i$, for any other path $P_j \in \mathcal{P}$.
- Suppose that P shares a vertex v with some path $P_i \in \mathcal{P}$. Vertex v cannot be an endpoint of P, because the endpoints of P are unmatched under M'.
- Therefore, v has an incident edge in P that belongs to M'.
- Since any vertex has at most one incident edge in a matching, this edge must also belong to P_i, thus proving the claim.

Lemma 25.5

- Because $A = (P_1 \cup P_2 \cup \cdots \cup P_k) \oplus P$ and P shares at least one edge with some $P_i \in \mathcal{P}$, we have that $|A| < |P_1 \cup P_2 \cup \cdots \cup P_k| + |P|$.
- Thus, we have

$$\begin{aligned} kq + q &\leq |A| \\ &< |P_1 \cup P_2 \cup \cdots \cup P_k| + |P| \\ &= kq + |P| \ , \end{aligned}$$

- so that $q < |P|$.
- We conclude that P contains more than q edges.

Lemma 25.6: Intro

- This lemma bounds the size of a maximum matching, based on the length of a shortest augmenting path.

Lemma 25.6

- Let M be a matching in graph $G = (V, E)$, and let a shortest M-augmenting path in G contain q edges. Then, the size of maximum matching in G is at most $|M| + |V|/(q + 1)$.

- Let M^* be a maximum matching in G. By Lemma 25.3, G contains at least $|M^*| - |M|$ vertex-disjoint M-augmenting paths.
- Each of these paths contains at least q edges, and hence at least $q + 1$ vertices.
- Because these paths are vertex-disjoint, we have $(|M^*| - |M|)(q + 1) \leq |V|$, so that $|M^*| = |M| + |V|/(q + 1)$.

Lemma 25.7: Intro

- This final lemma bounds the number of iterations of the repeat loop of lines 2-5.

Lemma 25.7

- When the HOPCROFT-KARP procedure runs on an undirected bipartite graph $G = (V, E)$, the repeat loop of lines 2-5 iterates $O(\sqrt{V})$ times.
  - By Lemma 25.5, the length q of the shortest M-augmenting paths found in line 3 increases from iteration to iteration.
  - After $\lceil \sqrt{V} \rceil$ iterations, therefore, we must have $q \geq \lceil \sqrt{V} \rceil$.
  - Consider the situation after the first time line 4 executes with M-augmenting paths whose length is at least $\lceil \sqrt{V} \rceil$.
  - Since the size of a matching increases by at least one edge per iteration, Lemma 25.6 implies that the number of additional iterations before achieving a maximum matching is at most

$$\frac{|V|}{\lceil \sqrt{|V|} \rceil + 1} < \frac{|V|}{\sqrt{|V|}} = \sqrt{|V|} .$$

  - Hence, the total number of loop iterations is less than $2\sqrt{V}$.

25.2 The Stable-Marriage Problem

Introduction

- In Section 25.1, the goal was to find a maximum matching in an undirected bipartite graph.

- If the input graph $G = (V, E)$ with vertex partition $V = L \cup R$ is a complete bipartite graph, then the maximum matching can be found using a simple greedy algorithm.

- A complete bipartite graph is a bipartite graph which contains an edge from every vertex in L to every vertex in R.

Introduction

- We add some information to each vertex in a complete bipartite graph: a ranking of the vertices in the other side.

- That is, each vertex in L has an ordered list of all the vertices in R, and vice-versa.

- To keep things simple, let us assume that L and R each contain n vertices.

- The goal here is to match each vertex in L with a vertex in R in a "stable" way.

The Stable-Marriage Problem

- Suppose there is a group of woman (L) and a group of men (R).
- Each woman ranks all the men in terms of desirability.
- Each man does the same with all the women.

- The goal is to pair up women and men (a matching) so that if a woman and a man are not matched to each other, then at least one of them prefers their assigned partner.
- If a woman and a man are not matched to each other but each prefers the other over their assigned partner, they form a blocking pair.
- A blocking pair has incentive to opt out of the assigned pairing and get together on their own. If that were to occur, then this pair would block the matching from being "stable".

- A stable matching is a matching that has no blocking pair.
  - If there is a blocking pair, then the matching is unstable.

The Stable-Marriage Problem: Example

- Four women and four men have the following preferences:

Wanda: Brent, Hank, Oscar, Davis
Emma: Davis, Hank, Oscar, Brent
Lacey: Brent, Davis, Hank, Oscar
Karen: Brent, Hank, Davis, Oscar

Oscar: Wanda, Karen, Lacey, Emma
Davis: Wanda, Lacey, Karen, Emma
Brent: Lacey, Karen, Wanda, Emma
Hank: Lacey, Wanda, Emma, Karen

- A stable matching comprises the following pairs:

Lacey and Brent
Wanda and Hank
Karen and Davis
Emma and Oscar

The Stable-Marriage Problem: Example

- We can verify that this matching has no blocking pair.
- Even though Karen prefers Brent and Hank to her partner Davis, Brent prefers his partner Lacey to Karen, and Hank prefers his partner Wanda to Karen, so that neither Karen and Brent nor Karen and Hank form a blocking pair.

The Stable-Marriage Problem: Example

- Stable matchings need not be unique.
- Suppose there are three women and three men with these preferences.

Monica:	Chandler, Joey, Ross
Phoebe:	Joey, Ross, Chandler
Rachel:	Ross, Chandler, Joey
Chandler:	Phoebe, Rachel, Monica
Joey:	Rachel, Monica, Phoebe
Ross:	Monica, Phoebe, Rachel

- In this case, there are three stable matchings:

Matching 1	Matching 2	Matching 3
Monica and Chandler	Phoebe and Chandler	Rachel and Chandler
Phoebe and Joey	Rachel and Joey	Monica and Joey
Rachel and Ross	Monica and Ross	Phoebe and Ross

Can we always find a stable matching?

- It is always possible to come up with a stable matching no matter what rankings each participant provides.

- A simple algorithm known as Gale-Shapley algorithm always finds a stable matching.

Gale-Shapley Algorithm

- The algorithm has two variants: "woman-oriented" and "man-oriented".
  - We examine the woman-oriented version.
- Each participant is either "free" or "engaged".
  - Everyone is free at the initial state.
- Engagements occur when a free woman proposes to a man.
- When a man is first proposed to, he goes from free to engaged.
  - He always stays engaged, though not necessarily to the same woman.
- If an engaged man receives a proposal from a woman whom he prefers to the woman he's currently engaged to, that engagement is broken.
  - The woman to whom he had been engaged becomes free.
  - The man and the woman he prefers become engaged.
- Each woman proposes to the men in her preference list, in order, until the last time she becomes engaged.
- When a woman is engaged, she temporarily stops proposing, but if she becomes free again, she continues down her list.
- Once everyone is engaged, the algorithm terminates.

Gale-Shapley Algorithm

- The procedure GALE-SHAPLEY
  - Any free woman may be selected line 2.

```
GALE-SHAPLEY(men, women, rankings)
 1  assign each woman and man as free
 2  while some woman w is free
 3      let m be the first man on w's ranked list to whom she has not proposed
 4      if m is free
 5          w and m become engaged to each other (and not free)
 6      elseif m ranks w higher than the woman w' he is currently engaged to
 7          m breaks the engagement to w', who becomes free
 8          w and m become engaged to each other (and not free)
 9      else m rejects w, with w remaining free
10  return the stable matching consisting of the engaged pairs
```

- For the man-oriented version, we can just reverse the roles of men and women in the procedure.

Gale-Shapley Algorithm: Example

- Suppose we have the following input to the stable-marriage problem.

Wanda: Brent, Hank, Oscar, Davis
Emma: Davis, Hank, Oscar, Brent
Lacey: Brent, Davis, Hank, Oscar
Karen: Brent, Hank, Davis, Oscar

Oscar: Wanda, Karen, Lacey, Emma
Davis: Wanda, Lacey, Karen, Emma
Brent: Lacey, Karen, Wanda, Emma
Hank: Lacey, Wanda, Emma, Karen

Gale-Shapley Algorithm: Example

- The algorithm proceeds as follows:
  1. Wanda proposes to Brent. Brent is free, so Wanda and Brent become engaged.
  2. Emma proposes to Davis. Davis is free, so Emma and Davis become engaged.
  3. Lacey proposes to Brent. Since Brent prefers Lacey, they become engaged, and Wanda becomes free.
  4. Karen proposes to Brent. Since Brent prefers Lacey, Brent rejects Karen.
  5. Karen proposes to Hank. Since Hank is free, so Karen and Hank become engaged.
  6. Wanda proposes to Hank. Since Hank prefers Wanda, they become engaged, and Karen becomes free.
  7. Karen proposes to Davis. Since Davis prefers Karen, they become engaged, and Emma becomes free.
  8. Emma proposes to Hank. Since Hank prefers Wanda, Hank rejects Emma.
  9. Emma proposes to Oscar. Oscar is free, so Emma and Oscar become engaged.
- At this point, nobody is free, so the algorithm terminates.

- Result: Wanda-Hank, Emma-Oscar, Lacey-Brent, Karen-Davis

Theorem 25.9: Proof of Bounded Time Termination

- The procedure GALE-SHAPLEY always terminates and returns a stable matching.

- To prove that the while loop of GALE-SHAPLEY procedure terminates, we use contradiction.
- If the loop does not terminate, it is because some woman remains free.
- In order for a woman to remain free, she must have proposed to all the men and been rejected by each one.
- In order for a man to reject a woman, he must already be engaged.
- Therefore, all men are engaged.
- Once a man is engaged, he stays engaged.
- There are equal number n of women and men, however, which means that every woman is engaged, leading to a contradiction that no women are free.

Theorem 25.9: Proof of Bounded Time Termination

- We must also show that the while loop makes a bounded number of iterations.
- Since each of the n women goes through her ranking of the n men in order, possibly not reaching the end of her list, the total number of iterations is at most n^2.
- Therefore, the while loop always terminates, and the procedure returns a matching.

Theorem 25.9: Proof of Correctness

- Now we show that the procedure returns a stable matching. In other words, there are no blocking pairs.
- We first observe that once a man m is engaged to a woman w, all subsequent actions for m occur in lines 6-8.

```
6       elseif m ranks w higher than the woman w' he is currently engaged to
7           m breaks the engagement to w', who becomes free
8           w and m become engaged to each other (and not free)
```

- Once a man is engaged, he stays engaged, and any time he breaks an engagement to a woman w, it's for a woman whom he prefers to w.

Theorem 25.9: Proof of Correctness

- Suppose a woman w is matched with a man m, but she prefers man m'.
- We'll show that w and m' is not a blocking pair, because m' does not prefer w to his partner.
- Because w ranks m' higher than m, she must have proposed to m' before proposing to m.
- m' either rejected her proposal or accepted it and later broke the engagement.
- If m' rejected the proposal from w, it is because he was already engaged to some woman he prefers to w.
- If m' accepted and later broke the engagement, he was at some point engaged to w but later accepted a proposal from a woman he prefers to w.
- In either case, he ultimately ends up with a partner whom he prefers to w.
- We conclude that even though w might prefer m' to her partner m, it is not also the case that m' prefers w to his partner.
- Therefore, the procedure returns a matching that contains no blocking pairs.

Corollary 25.10

- Given preference rankings for n women and n men, the Gale-Shapley algorithm can be implemented to run in $O(n^2)$ time.

Can the algorithm produce different results in each run?

- In line 2 of the GALE-SHAPLEY procedure, we can choose any order of women proposing to men.

- Because of this, we may wonder whether difference choices can produce different stable matchings.
  - Remember that stable matchings are not unique.

- The answer is no: the algorithm always returns exactly the same result, regardless of the order at line 2.

- Moreover, the stable matching returned is optimal for the women.

Theorem 25.11

- Regardless of how women are chosen in line 2 of GALE-SHAPLEY, the procedure always returns the same stable matching, and in this stable matching, each woman has the best partner possible in any stable matching.

- The proof that each woman has the best partner possible in any stable matching is by contradiction.
- Suppose that the GALE-SHAPLEY procedure returns a stable matching M, but that there is a different stable matching M' in which some woman w prefers her partner m' to the partner m she has in M.
- Because w ranks m' higher than m, she must have proposed to m' before proposing to m.
- Then there is a woman w' whom m' prefers to w, and m' was already engaged to w' when w proposed, or m' accepted the proposal from w and later broke the engagement in favor of w'.

Theorem 25.11

- Either way, there is a moment when m' decided against w in favor of w'.
- Now suppose, without loss of generality, that this moment was the first time that any man rejected a partner who belongs to some stable matching.

- We claim that w' cannot have a partner m'' in a stable matching whom she prefers to m'.
- If there were such a man m'', then in order for w' to propose to m', she would have proposed to m'' and been rejected at some point before proposing to m'.
- If m' accepted the proposal from w and later broke it to accept w', then since this was the first rejection in a stable matching, we get the contradiction that m'' could not have rejected w' beforehand.
- If m' was already engaged to w' when w proposed, then again, m'' could not have rejected w' beforehand, thus proving the claim.

Theorem 25.11

- Since w' does not prefer anyone to m' in a stable matching and w' is not matched with m' in M' (because m' is matched with w in M'), w' prefers m' to her partner in M'.
- Since w' prefers m' over her partner in M' and m' prefers w' over his partner w in M', the pair w' and m' is a blocking pair in M'.
- Because M' has a blocking pair, it cannot be a stable matching, thereby contradicting the assumption that there exists some stable matching in which each woman has the best partner possible other than the matching M returned by GALE-SHAPLEY.

- We put no condition on the execution of the procedure, which means that all possible orders in which line 2 selects women result in the same stable matching being returned.

Corollary 25.12

- There can be stable matchings that the GALE-SHAPLEY procedure does not return.

- Theorem 25.11 says that for a given set of rankings, GALE-SHAPLEY returns just one matching, no matter how it chooses women in line 2.
- The earlier example of three women and three men with three different stable matchings shows that there can be multiple stable matchings for a given set of rankings.
- A call of GALE-SHAPLEY is capable of returning only one of these stable matchings.

Corollary 25.13

- In the stable matching returned by the procedure GALE-SHAPLEY, each man has the worst partner possible in any stable matching.

- Let M be a matching returned by a call to GALE-SHAPLEY.
- Suppose that there is another stable matching M' and a man m who prefers his partner w in M to his partner w' in M'.
- Let the partner of w in M' be m'.
- By Theorem 25.11, m is the best partner that w can have in any stable matching, which means that w prefers m to m'.
- Since m prefers w to w', the pair w and m is a blocking pair in M', contradicting the assumption that M' is a stable matching.

End of Class

Questions?

Instructor office: AS-1013

Email: jso1@sogang.ac.kr