Stable Matching

Till now, we have discussed the maximum matching problem, first for bipartite graphs, and then for general graphs. Now, we come to a different problem related to matching in bipartite graphs, the **stable matching problem**.

Stable Matching Problem

Suppose, in a university, there are n students and n professors. Each professor offers a distinct research project, and we need to allocate the projects among students. Every student must be assigned exactly one project (and vice versa).

Now, the problem is, that each student has some individual preferences for research projects in their mind; let's presume that each student has a complete ranking of the projects, from their most preferred to their least preferred. Similarly, every professor has a preference ranking of each student, to be chosen for their project. We wish to pair up professors and students, such that the matching is **stable**.

To understand what stability means, let's take a concrete example: suppose that there are three students a, b and c, and three professors p, q and r.

Let their individual preference orders be the following:

Preference order of a: $p > q > r$	Preference order of p: $b > a > c$
Preference order of b: $q > p > r$	Preference order of q: $a > c > b$
Preference order of c: $p > r > q$	Preference order of r: $c > a > b$

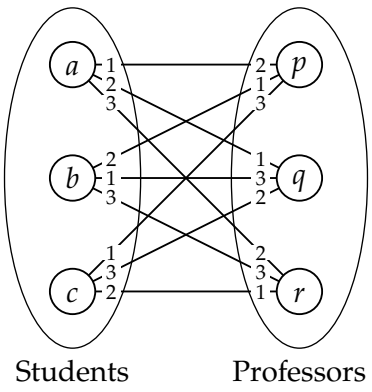

Figure 1: Preference order of students and professors shown on a bipartite graph. The number on the edge denotes the rank of preference.

Suppose, we pair up the professors and the students as follows: Student a with professor r; student b with professor q; and student c with professor p. This is denoted by the matching $M_1 = \{(a, r), (b, q), (c, p)\}$ (Figure 2).

But, student a has gotten his least preferred project (under professor r). Unhappy with his allocation, he goes to visit professor p in his room (which is his most preferred allocation). He finds out that professor p is also not happy with the allocation, and in fact professor p prefers to work with student a rather than his currently allocated student (c). Then, a and p are together incentivised (by their preference orders) to ignore the allocation, and together work on professor p's research project. Thus, we say that **matching M_1 is unstable**, since it leads to

such unhappy pairs of people who prefer to be matched to each other, rather than their current partners.

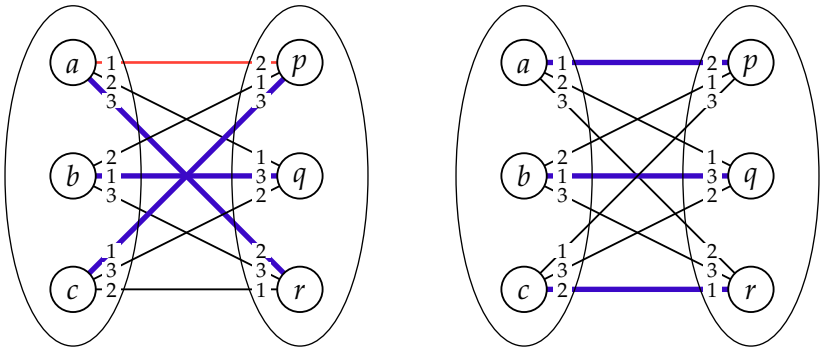

Figure 2: Two possible matchings M_1 and M_2.

Instead, let's say we started with a different allocation: $a - p, b - q, c - r$. Call this matching M_2. M_2 is actually a **stable matching**, so there do not exist any such pair that is incentivised to collude and break the matching.

For example, we see that professor q got his least preferred student (b) to work on his project. But, even if he approaches the other students $(a$ and $c)$, they wouldn't be willing to leave their currently allocated project to work with him, as neither of them prefer professor q's project over what they have been allocated by this matching. So, M_2 is stable. (Can you find some other stable matchings?)

So our problem is, on such a general instance (where the cardinality of the sets is n), we need to find a stable matching (if it exists). The framework of stable matching has a lot of applications, for example, matching men with women in a marriage portal, doctors with hospitals, applicants with jobs. We now formally define the stable matching problem.

Formal Definition

Let A and B be two sets of n vertices each, $|A| = |B| = n$. Each vertex $a \in A$ has a total preference order $>_a$ over B. Similarly, every vertex $b \in B$ has a total preference order $>_b$ over A.

A matching M is a set of pairs (a, b) where $a \in A$ and $b \in B$, such that no $a \in A$ is matched to more than one element in B, and vice versa (each element has at most one paired element from the other set).

For a matching M, a pair (a, b) is said to be a **blocking pair**, if
1. a is unmatched, and b is unmatched; or
2. a is unmatched, b is matched to some $M(b) \in A$ such that $a >_b M(b)$.
3. b is unmatched, a is matched to some $M(a) \in B$ such that $b >_a M(a)$.
4. Both a and b are matched to $M(a)$ and $M(b)$, and $a >_b M(b)$ and $b >_a M(a)$.

A matching M is said to be a **stable matching** if there is no blocking pair with respect to M.

The **stable matching problem**: Given two sets A and B of size n, and a total preference order $>_a$ for each $a \in A$ on B, and a total preference order $>_b$ for each $b \in B$ on A, find a stable matching M.

Gale-Shapley Algorithm

For describing the algorithm, let us switch to the marriage proposal analogy (as it is classically described). Let us call the sets A and B the sets of men and women respectively. The algorithm described below is called **"men-proposing deferred acceptance algorithm"**.

Initially, everyone is unmatched. We pick any arbitrary unmatched man and let him propose to his most preferred woman whom he has not proposed yet (regardless of whether she is matched or not). Whenever an unmatched woman receives a proposal, she accepts her proposal and gets matched. Whenever a matched woman receives a proposal, if she prefers the proposer more than her current matched partner, then she accepts the proposal (and her previous matched partner becomes unmatched). Otherwise, she rejects the proposal.

We continue until every man (and hence every woman) is matched.

GALE-SHAPLEY(G)

```
1  M := φ                          ▷ initially, all vertices are unmatched
2  while there is an unmatched a ∈ A
3     choose any such a
4     a tries to match with his most preferred b ∈ B who he hasn't yet proposed
5     if b is unmatched
6        add (a, b) to M
7     else if a >_b M(b)           ▷ b prefers a over her current partner
8        remove (M(b), b), add (a, b) to M
```

We will show that the above algorithm terminates in polynomial time, and returns a stable matching. As a corollary, this also shows that **every instance of the stable matching problem has a stable matching**.

> **Theorem**
>
> The deferred acceptance algorithm computes a stable matching in polynomial time.

Observe, that once a $b \in B$ gets matched, she never gets unmatched again, though she may change her partner. Also, every woman who has received at least one proposal is matched. Since there are n men and each man proposes to the women on his preference list at most once, the maximum number of proposals that can take place is n^2. Because exactly one new proposal is made in each iteration of the loop, the algorithm must terminate after at most n^2 iterations.

To show that M is a stable matching, suppose (a, b) is a blocking pair in M. Then, there are two cases:

1. a never got to propose b during the algorithm. Then, that means the current partner of a is more preferable to him, than b (since all men propose women in their order of preference). So, (a, b) cannot form a blocking pair.
2. a proposed b during the algorithm. At termination, they are unmatched, which means either b rejected a's proposal, or they separated later as b was proposed by a better suitor. In either case, b prefers her current match more than a, hence (a, b) don't form a blocking pair. ■

Example

Let us run the Gale-Shapley algorithm on the initial example on the first page. Assume that the set A is the set of students, so the students 'propose' to the professors for getting their preferred research projects.

Preference order of a: $p > q > r$	Preference order of p: $b > a > c$
Preference order of b: $q > p > r$	Preference order of q: $a > c > b$
Preference order of c: $p > r > q$	Preference order of r: $c > a > b$

Initially everyone is unmatched.
1. a proposes to p. Since p is unmatched, (a, p) form a pair. $M = \{(a, p)\}$.
2. b proposes to q. Since q is unmatched, (b, q) form a pair. $M = \{(a, p), (b, q)\}$.
3. c proposes to p. Since p is already matched, and p does not prefer c over his current student (a), p rejects c's proposal. $M = \{(a, p), (b, q)\}$
4. c proposes to r. Since r is unmatched, (c, r) form a pair. $M = \{(a, p), (b, q), (c, r)\}$.

At this point, all students have been matched, so we get the stable matching $M = \{(a, p), (b, q), (c, r)\}$.

Let's try to again run the algorithm, but this time the professors 'propose' to their preferred students, to work on their projects.

Initially, everyone is unmatched.
1. Professor p proposes to b. Since b is unmatched, (b, p) form a pair. $M = \{(b, p)\}$.
2. Professor q proposes to a. Since a is unmatched, (a, q) form a pair. $M = \{(a, q), (b, p)\}$.
3. Professor r proposes to c. Since c is unmatched, (c, r) form a pair. $M = \{(a, q), (b, p), (c, r)\}$.

At this point, all professors have been matched to some student (in fact, their most preferred students), we have a stable matching $M' = \{(a, q), (b, p), (c, r)\}$. (Note that this is different from M.)

Optimality

We know that any instance of the stable matching problem may have multiple stable matchings. Which one does the deferred acceptance algorithm compute? Observe that on line 3 in the algorithm, we do not specify how we choose the unmatched vertex $a \in A$ for that iteration. It might be a reasonable guess that choosing men in different orders (in line 3) will lead to different stable matchings in the algorithm. But, it turns out this is not the case! We will show this by proving a stronger statement.

For a vertex $a \in A$, let $h(a)$ be the most preferred woman (according to $>_a$) that a can be matched with in some stable matching M. Then, we claim that the matching computed by Gale-Shapley is nothing but $\{(a, h(a)) \mid a \in A\}$. We call this matching **men-optimal**.

> **A-optimal stable matching**
>
> In the A-proposing deferred acceptance algorithm, the stable matching M is A-optimal, i.e. for all $a \in A$, there does not exist any stable matching M' such that $M'(a) >_a M(a)$.

Consider a run of the men-proposing Gale-Shapley algorithm. Define $R_i = \{(a, b) \in A \times B \mid b$ has rejected a in the first i iterations$\}$, and $R = \cup_i R_i$. To prove the statement, it suffices to

show that for any (a, b) in R (a rejection pair), there does not exist any stable matching which matches a with b.

We show this by induction on the number of iterations i. Clearly the statement holds for R_0. Assume the statement to be true for R_i. In the $(i+1)$th iteration, suppose some woman b received a proposal from a man a, and b rejects their current partner a'. So, $a >_b a'$, and $R_{i+1} = R_i \cup \{(a', b)\}$. Since a proposes b in this iteration, he must have been rejected by all women b' that he strictly prefers to b (for all $b' >_a b$). Thus, all such (a, b') are rejected pairs, $(a, b') \in R_i$. By the induction hypothesis, a cannot be matched to any such b' in some stable matching.

Suppose there is some stable matching M' which matches (a', b). Consider the woman a gets matched to in this matching, call her $M'(a)$. By the induction hypothesis, we know that $M'(a)$ cannot be some b', i.e. $M'(a)$ cannot be someone who a strictly prefers to b. $M'(a)$ also cannot be b since b is matched with a different man (a'). Thus, it must be the case that $b >_a M'(a)$ (a strictly prefers b over his current match). It is also the case that $a >_b a'$ (from the previous paragraph), i.e. b strictly prefers a over her current match a'. Therefore, (a, b) forms a blocking pair, hence M' is not a stable matching.

So, every rejected pair (in a run of the GS algorithm) cannot be matched in any stable matching. For some $a \in A$, let his preference order over women be $b_1 >_a b_2 >_a \ldots >_a b_n$. Suppose he is finally matched with b_j at the termination of the algorithm. Then all of $b_1, b_2, \ldots, b_{j-1}$ must have rejected a at some point in the algorithm. By what we have proved, there does not exist any stable matching where a gets matched with any of $b_1, b_2, \ldots, b_{j-1}$. Thus, $h(a) = b_j$, i.e. b_j is the most optimal choice that a can get in a stable matching, which he gets in the GS algorithm. ■

Problems

1. Describe how to implement the Gale-Shapley algorithm so that it runs in $O(n^2)$ time.

Here is the informal pseudocode for the algorithm:

GALE-SHAPLEY(G)

```
1  M := φ                          ▷ initially, all vertices are unmatched
2  while there is an unmatched a ∈ A
3      choose any such a
4      a tries to match with his most preferred b ∈ B who he hasn't yet proposed
5      if b is unmatched
6          add (a, b) to M
7      else if a >_b M(b)          ▷ b prefers a over her current partner
8          remove (M(b), b), add (a, b) to M
```

Let us assume that the preference orders $>_a$ are given as lists of women, for each man a, where the first element in the list is the most preferred woman b according to $>_a$ (and similarly for $>_b$ for each woman b).

We need the following operations to work in $O(1)$:
1. Select some unmatched man (line 2-3)
2. For some man a, find his most preferred woman b whom he hasn't proposed yet (line 4)
3. For some woman b and men x, y, check if $x >_b y$ (line 7)

So, we maintain the following data structures to facilitate these operations:
- **Rank matrix** (R_B): a precomputed $n \times n$ matrix, where $R_B[b, a]$ denotes the rank of a in the preference order of b ($>_b$). The most preferred man has rank 1, and the least preferred man has rank n.
- **Next proposal pointer** `next_proposal[a]` for each $a \in A$ points to the next woman that a should propose to, according to the preference order $>_a$. Initially it points to the head of the list $>_a$ (the most preferred woman according to a).
- **Unmatched men stack/queue** `unmatched` is some collection supporting $O(1)$ insertions and arbitrary extractions, to store the list of currently unmatched men.
- The **matching** M will be stored as an array `match` of size n where `match[b]` denotes the man that b is currently matched with, for all $b \in B$.

So, the algorithm proceeds as follows:
1. Initialise R_B as an empty $n \times n$ matrix.
   - For all women $b \in B$, iterate on their preference list $>_b$. On the ith iteration, let the element be a, then set $R_B[b, a] = i$.
2. Initialise `next_proposal[a]` to point to the head of the list $>_a$.
3. Push all $a \in A$ into `unmatched`.
4. Initialise `match[a]` to null.
5. While `unmatched` is not empty, pop an element a from it.
   - Let $b =$ `next_proposal[a]`.
   - If `match[b]` is null, set `match[b]` to a.
   - Else, let $a' =$ `match[b]`.
     - If $R_B[b, a] < R_B[b, a']$, then set `match[b]` to a, and push a' to `unmatched`
     - Else, push a to `unmatched`.
   - Increment `next_proposal[a]`.

Here, step 1 runs in $O(n^2)$, steps 2-4 run in $O(n)$, and step 5 runs for $O(n^2)$ iterations, and each iteration runs in $O(1)$. Thus, the implementation runs in $O(n^2)$. The correctness trivially follows from the base algorithm.

2. The National Resident Matching Program differs from the scenario for the stable marriage problem in two ways. First, a hospital may be matched with more than one student, so that hospital h takes $r_h \geq 1$ students. Second, the number of students might not equal the number of hospitals. Describe how to modify the Gale-Shapley algorithm to fit the requirements of the National Resident Matching Program.

Suppose there are m hospitals, each with r_h available positions. There are n students.

We can describe a student-proposing deferred acceptance algorithm:

- Initialise all students to be unassigned. For each hospital h, set its available capacity r_h.
- While there is some unassigned student who has some unproposed hospital,
  - Choose any such student s. Let the highest ranked unproposed hospital (according to $>_s$) be h. s proposes to h.
  - When a hospital h receives a proposal from a student s,
    - If its available capacity is non-zero, it accepts student s. (s gets assigned to h). The hospital's available capacity decreases.
    - If its available capacity is zero: Let s' be the least preferred student according to $>_h$ who is currently assigned to h. Then,
      - If $s >_h s'$, then h lets go of s' (who becomes unassigned), and accepts s (who becomes assigned to h).
      - Otherwise, h rejects s.

The algorithm terminates when every student has been assigned or all open availabilities have been fulfilled.

3. Prove the following property, which is known as weak Pareto optimality: Let M be a stable matching produced by the Gale-Shapley algorithm, with women proposing to men. Then, for a given instance of the stable-marriage problem, there is no matching (stable or unstable), such that **every** woman has a partner whom she prefers to her partner in the stable matching M.

Note that this claim is different from the (wo)men-optimal matching claim which we have already proved.

Take A to be the set of women, and B to be the set of men. Let M be a matching produced by the women-proposing deferred acceptance algorithm.

For contradiction, suppose there is some matching M' (stable or unstable), such that for all women $a \in A$, $M'(a) >_a M(a)$, i.e. every woman prefers their M'-partner to her M-partner. Since a prefers $M'(a)$ over her final partner $M(a)$, it implies for all $a \in A$, at some point in the algorithm, $M'(a)$ must have rejected a.

Consider the proposal that was made in the last iteration of the Gale-Shapley algorithm. Say, woman $w \in A$ proposes to man $m \in B$ in the last iteration. (Call this iteration i_1.) Since this is the last iteration, this must lead to a matching (otherwise w will remain unmatched after this). So, $(w, m) \in M$ is present in the GS matching.

Now, think about the M'-partner of m, $M'(m) = y \in A$. At some point during the algorithm, y must have been rejected by $m = M'(y)$. This may have happened in iteration i_1 itself, or some iteration before that. Regardless, this means that at iteration i_1, before m accepted w's proposal, he must have necessarily been matched with some woman. Let's call her w'.

But then, this means that after m accepts w on iteration i_1, then w' becomes unmatched. This contradicts i_1 being the last iteration of the algorithm (since we cannot terminate yet, as there is w' who is unmatched). Thus, no such matching M' exists.

4. The stable roommates problem is similar to the stable marriage problem, except that the graph is a complete graph, not bipartite, with an even number of vertices. Each vertex represents a person, and each person ranks all the other people. The definitions of a blocking pair and stable matching extend in a natural way: a blocking pair comprises two people who both prefer each other to their current partner, and a matching is stable if there are no blocking pairs.

   Unlike the stable marriage problem, the stable roommates problem can have inputs for which no stable matching exists. Find such an input and explain why no stable matching exists.

Consider an instance of the stable roommate problem with $n = 4$, let the 4 vertices be called a, b, c, d.

Preference order of a: $b >_a c >_a d$

Preference order of b: $c >_b a >_b d$

Preference order of c: $a >_c b >_c d$

Preference order of d: $a >_d b >_d c$

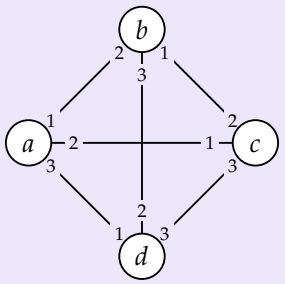

Figure 3: An instance of the stable roommates problem, with $n = 4$ vertices.

We claim, that no matching M on this instance can be stable. First observe that d is everyone's last preference. Suppose there does exist some stable matching M on this graph. Then, let $x = M(d)$ be the vertex matched with d. Then, the remaining two vertices (other than d and x) get matched together.

Notice that a, b, c have a cyclic symmetry. WLOG, consider $x = a$. Then, $M = \{(a, d), (b, c)\}$. But then, (a, c) forms a blocking pair, since a prefers c over d (obviously), and a is c's first choice.

Regardless of who x is, we will find a blocking pair between x and whoever's first preference is x. Thus, in this instance, no stable matching exists.

5. Consider the men-proposing deferred acceptance algorithm for a stable marriage instance with n men and n women. Suppose that during the execution of the algorithm, a man m is rejected by a woman w.

   (a) Prove that m can never be matched with w in any stable matching.

(b) Use the above result to prove that the matching produced by the men-proposing deferred acceptance algorithm is *men-optimal*; that is, every man weakly prefers his partner in the output matching to his partner in any other stable matching.

See proof of men-optimal matching, (page 4).

6. Consider a stable marriage instance with n men and n women.

(a) Give an example of a preference profile having exactly one stable matching.
(b) Give an example of a preference profile having more than one stable matching.
(c) Investigate whether a stable marriage instance can have exponentially many stable matchings. State an appropriate bound and justify your answer.
(d) Explain why enumerating all $n!$ perfect matchings is not an efficient algorithm for finding a stable matching.

(a) If we can satisfy everyone's first preference, then that must be the only stable matching, because if some such pair is not in the matching, then that will be a blocking pair, as they both prefer each other than anyone else. For example, for $n = 3$ we have:

Preference order of a: $p > q > r$	Preference order of p: $a > b > c$
Preference order of b: $q > p > r$	Preference order of q: $b > a > c$
Preference order of c: $r > p > q$	Preference order of r: $c > a > b$

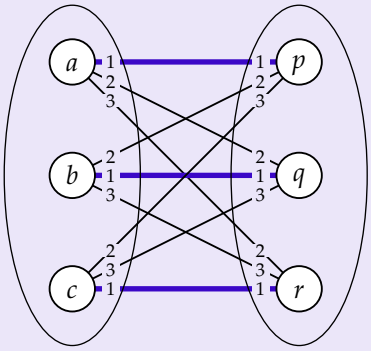

Figure 4: An instance of the stable-matching problem with only one stable matching.

$M = \{(a,p),(b,q),(c,r)\}$. Suppose there exists some stable matching $M' \neq M$. Then consider the pair in M which is not in M', call it (x,y). Then, (x,y) forms a blocking pair since x's first preference is y, and y's first preference is x. Thus, M' is not a stable matching, so M is the only stable matching.

(b) Consider the example on the first page:

Preference order of a: $p > q > r$	Preference order of p: $b > a > c$
Preference order of b: $q > p > r$	Preference order of q: $a > c > b$
Preference order of c: $p > r > q$	Preference order of r: $c > a > b$

We have shown two stable matchings for this instance:
- $M = \{(a,p),(b,q),(c,r)\}$
- $M' = \{(a,q),(b,p),(c,r)\}$

(c) Yes, we can construct a stable marriage instance to have exponentially many stable matchings.

Consider the 2-person instance of the problem, with two men a_1 and a_2, and two women b_1 and b_2. Let their individual preference orders be:

Preference order of a_1: $b_1 > b_2$	Preference order of b_1: $a_2 > a_1$
Preference order of a_2: $b_2 > b_1$	Preference order of b_2: $a_1 > a_2$

This subinstance has two stable matchings: $\{(a_1, b_1), (a_2, b_2)\}$ (men-optimal) and $\{(a_1, b_2), (a_2, b_1)\}$ (female-optimal).

We can construct an n-person instance (for even n) by replicating this subinstance $n/2$ times. Thus, we have $A = \bigcup_{i=1}^{n/2}\{a_{i,1}, a_{i,2}\}$, $B = \bigcup_{i=1}^{n/2}\{b_{i,1}, b_{i,2}\}$. We define their preference order such that everyone from the ith subinstance strictly prefers members of their own subinstance (over anyone else), in accordance to the orders defined above. (The rest of the preference order does not matter).

Preference order of $a_{i,1}$: $b_{i,1} > b_{i,2} > \ldots$	Preference order of $b_{i,1}$: $a_{i,2} > a_{i,1} > \ldots$
Preference order of $a_{i,2}$: $b_{i,2} > b_{i,1} > \ldots$	Preference order of $b_{i,2}$: $a_{i,1} > a_{i,2} > \ldots$

Claim: In any stable matching M, no two people from different sub instances can be matched.

Suppose we have a matching where $a_{i,x}$ is matched to $b_{j,y}$. $(i \neq j; x, y \in \{1, 2\})$. Since $a_{i,x}$ is matched out of the subinstance, some woman in instance i also should be matched out of the subinstance. Call her $b_{i,z}$. But then, $(a_{i,x}, b_{i,z})$ form a blocking pair, since they prefer each other (within subinstance) to their current partners (outside subinstance). Thus, such pairings cannot happen.

Since in any stable matching, everyone will pair up with someone in the same subinstance, we can choose the stable matching for each subinstance independently. For each subinstance there are 2 stable matchings. By independent choices, we get that for $n/2$ subinstances we can get a total of $2^{n/2}$ stable matchings, which is exponential in n.

Therefore, a lower bound on the maximum number of stable matchings that an instance can have is $\Omega(2^{n/2})$.

(d) Enumerating all $n!$ perfect matchings takes $n!$ steps, which is worse than an exponential time algorithm. Instead, Gale-Shapley gives a polynomial time algorithm ($O(n^2)$) for finding a stable matching, which is way more efficient than any superexponential, or for that matter, even any exponential time algorithm.

7. Let M be a perfect matching between n men and n women.
   (a) Design a $O(n^2)$ algorithm to determine whether M is stable.
   (b) Write pseudocode for your algorithm.
   (c) Prove the correctness of your algorithm.
   (d) Analyze its time and space complexity.

Actually, the way the instances are being depicted in the figures (e.g. Figure 2, Figure 4) gives a clue on how we can achieve this. (If you are given one of these figures, you can check if the matching is stable or not in $O(n^2)$.)

To check if M is stable or not, we need to check for each $(a, b) \in A \times B$, whether (a, b) forms a blocking pair. Since M is a perfect matching, the only way this can happen is if $(a, b) \notin M$, and $b >_a M(a)$ and

$a >_b M(b)$. So we need to efficiently check who is more preferable according to someone's preference order. If we can do this in $O(1)$, we are done.

To do it, we will precompute $n \times n$ rank matrices for both men and women. Let $R_A[m, w]$ denote the rank of woman w in m's rank list (ranks are defined as 1 to the most preferred person, ..., n to the least preferred person). Similarly, $R_B[w, m]$ is the rank of man m in w's rank list. Note that if $x >_a y$, we have $R[a, x] < R[a, y]$.

For every $(a, b) \in A \times B$, if $R_A[a, b] < R_A[a, M(a)]$ and $R_B[b, a] < R_B[b, M(b)]$, then (a, b) is a blocking pair, and thus M is not stable.

IS-STABLE(G, M)

```
1   initialise R_A and R_B as n × n matrices
2   for a ∈ A
3       for i = 1 to n
4           b ← a's ith choice according to >_a
5           R_A[a, b] ← i
6   for b ∈ B
7       for i = 1 to n
8           a ← b's ith choice according to >_b
9           R_B[b, a] ← i
10  for a ∈ A
11      for b ∈ B
12          if R_A[a, b] < R_A[a, M(a)] and R_B[b, a] < R_B[b, M(b)]
13              return false
14  return true
```

Correctness: Firstly, lines 2 through 9 correctly compute the ranks (trivially). So, the property holds: for any $a \in A$ and $x, y \in B$, $x >_a y \Leftrightarrow R_A[a, x] < R_A[a, y]$ (similarly for B).

Thus, the check in line 12 checks if the pair (a, b) is a blocking pair.

Suppose M is a stable matching. Then, we know that there does not exist any blocking pair in M (by definition). So, the condition on line 12 will never be true, so we never go into the if block. Thus, we return true.

Suppose M is not a stable matching, then there must exist some pair (a, b) which is a blocking pair. Then, on some iteration of the double for loops, the condition on line 12 will evaluate to true, so we go into the body and return false.

Time Complexity: Lines 2 through 5 and 6 through 9 are nested loops which run in $O(n^2)$ (assuming that the preference orders are given as lists which we can iterate over). Lines 10 through 13 is also a nested loop which runs in $O(n^2)$. Thus the overall algorithm runs in $O(n^2)$.

Space Complexity: To compute R_A and R_B, we use $O(n^2)$ extra space.