Section 1: Solutions

Review of Graph Concepts

- **Degree:** The number of edges connected to a vertex.
- **Acyclic Graph:** A graph without cycles.
- **Tree:** An undirected, acyclic graph.
- **Path:** A list of vertices $v_0, v_1, ..., v_k$ in a graph such that (v_i, v_{i+1}) is an edge in the graph for all $0 \leq i < k$.

1. Gale-Shapley

Consider the following stable matching instance:

$p_1 : r_3, r_1, r_2, r_4$
$p_2 : r_2, r_1, r_4, r_3$
$p_3 : r_2, r_3, r_1, r_4$
$p_4 : r_3, r_4, r_1, r_2$

$r_1 : p_4, p_1, p_3, p_2$
$r_2 : p_1, p_3, p_2, p_4$
$r_3 : p_1, p_3, p_4, p_2$
$r_4 : p_3, p_1, p_2, p_4$

(a) Run the Gale-Shapley Algorithm with p_i proposing on the instance above. When choosing which free p_i to propose next, always choose the one with the smallest index (e.g., if p_1 and p_2 are both free, always choose p_1).

Solution:

The steps of the Gale-Shapley Algorithm with the p_i with lowest index proposing first:

p_1 chooses r_3	(p_1, r_3)
p_2 chooses r_2	$(p_1, r_3), (p_2, r_2)$
p_3 chooses r_2	$(p_1, r_3), (p_3, r_2)$
p_2 chooses r_1	$(p_1, r_3), (p_2, r_1), (p_3, r_2)$
p_4 chooses r_3	$(p_1, r_3), (p_2, r_1), (p_3, r_2)$
p_4 chooses r_4	$(p_1, r_3), (p_2, r_1), (p_3, r_2), (p_4, r_4)$

(b) Run the Gale-Shapley Algorithm with p_i proposing on the same instance. But now, when choosing which free p_i to propose next, always choose the one with the largest index. Do you get the same result?

Solution:

The steps of the Gale-Shapley Algorithm with the p_i with highest index proposing first:

p_4 chooses r_3	(p_4, r_3)
p_3 chooses r_2	$(p_3, r_2), (p_4, r_3)$
p_2 chooses r_2	$(p_3, r_2), (p_4, r_3)$
p_2 chooses r_1	$(p_2, r_1), (p_3, r_2), (p_4, r_3)$
p_1 chooses r_3	$(p_1, r_3), (p_2, r_1), (p_3, r_2)$
p_4 chooses r_4	$(p_1, r_3), (p_2, r_1), (p_3, r_2), (p_4, r_4)$

We ended up with the same result!

(c) Now run the algorithm with r_i proposing, breaking ties by taking the free r_i with the smallest index. Do you get the same result?

Solution:

The steps of the Gale-Shapley Algorithm with r_i proposing:

r_1 chooses p_4	(p_4, r_1)
r_2 chooses p_1	$(p_1, r_2), (p_4, r_1)$
r_3 chooses p_1	$(p_1, r_3), (p_4, r_1)$
r_2 chooses p_3	$(p_1, r_3), (p_3, r_2), (p_4, r_1)$
r_4 chooses p_3	$(p_1, r_3), (p_3, r_2), (p_4, r_1)$
r_4 chooses p_1	$(p_1, r_3), (p_3, r_2), (p_4, r_1)$
r_4 chooses p_2	$(p_1, r_3), (p_2, r_4), (p_3, r_2), (p_4, r_1)$

No, the result is different when we have the r_i propose as opposed to the p_i.

2. A Quick Proof

Is it possible to have a stable matching instance with more than 2 stable matchings? If so, give an instance and at least 3 stable matchings. If not, prove that every instance has at most 2 stable matchings.

Solution:

Consider the following instance:

$p_1 : r_1, r_2, r_3, r_4$
$p_2 : r_2, r_1, r_4, r_3$
$p_3 : r_3, r_4, r_1, r_2$
$p_4 : r_4, r_3, r_2, r_1$

$r_1 : p_2, p_1, p_4, p_3$
$r_2 : p_1, p_2, p_3, p_4$
$r_3 : p_4, p_3, p_2, p_1$
$r_4 : p_3, p_4, p_1, p_2$

This instance has four stable matchings:
$(p_1, r_1), (p_2, r_2), (p_3, r_3), (p_4, r_4)$
$(p_1, r_1), (p_2, r_2), (p_3, r_4), (p_4, r_3)$

$(p_1, r_2), (p_2, r_1), (p_3, r_3), (p_4, r_4)$
$(p_1, r_2), (p_2, r_1), (p_3, r_4), (p_4, r_3)$

3. Induction Review

Consider the following claim: Every tree with at least 2 nodes has at least 2 nodes of degree 1.

You may use the following fact without proof: Every tree has at least one node of degree 1.

(a) What is the inductive hypothesis and the correct "skeleton" of the inductive step (i.e., the right things to assume and the right target)? **Solution:**

For the inductive hypothesis, let $P(n)$ be "Every tree with exactly n nodes has at least 2 nodes of degree 1."

The inductive step should start with "Let T' be an arbitrary tree with $k+1$ nodes."

Our conclusion will be that T' has at least two nodes of degree 1, so $P(k+1)$ holds.

(b) Prove the claim by induction.
Solution:

(**Note:** By mistake, this problem has a more difficult induction proof than we anticipated. There are easier non-induction proofs for the claim, check Ed for some hints.)

Let $P(n)$ be "Every tree with exactly n nodes has at least 2 nodes of degree 1." We prove the claim by induction on n.

Base Case: $n = 2$. There is only one undirected tree with two nodes. It has 2 nodes of degree 1.

Inductive Hypothesis: Suppose $P(n)$ holds for $n = 2, ..., k$ for an arbitrary $k \geq 2$.

Inductive Step: Let T' be an arbitrary tree with $k+1$ nodes.

Now, let u be a vertex of T' of degree 1 (this vertex exists by the above fact). Let T'' be the *graph* created by deleting u from T'. To show that T'' is a tree, we must show that it is connected and acyclic. It is acyclic because deleting edges cannot create cycles. To show that it is connected, recall that a graph is connected if there is a path between any two vertices. The internal nodes of paths all have degree 2, so u cannot be an internal node in any path. Therefore, between any two vertices in T'', the original path in T' connecting them remains intact, so T'' is connected.

Because T'' is a tree, we may apply the inductive hypothesis to get 2 nodes of degree 1 in T''. Say they are a and b. When we reattach u to get back T', this changes the degree of at most one of a and b. Hence, u and at least 1 of a and b are the desired 2 nodes of degree 1, proving $P(k+1)$.

4. Find the Bug: Failed Induction

In this problem you will fix an incorrect induction proof.

Let's do a little bit of problem setup. Suppose you have a stable matching instance with n people in P and n people in R. Of the n members of R, 5 are **popular**. That is, every person in P has those 5 members of R as their first 5 choices (in some order, not necessarily the same for each person in P). Similarly, you have 5 **popular** members of P, such that every person in R has those 5 as their top choices.

Let $P(n)$ be "In every stable matching instance with $2n$ in two groups of n people of which 5 members of each group are popular: in every stable matching, popular people in each group are matched to each other."

Spoof. We will show $P(n)$ holds for all $n \geq 5$ by induction on n.

Base Case $(n = 5)$
With both sets having size 5, every person is popular. Since every stable matching pairs every agent, every agent is matched to a popular agent.

Inductive Hypothesis: Suppose $P(n)$ holds for $n = 5, ..., k$ for an arbitrary integer $k \geq 5$.

Inductive Step: Let $r_1, ..., r_k$, $p_1, ..., p_k$ be k people in each group, with $r_1, ..., r_5, p_1, ..., p_5$ being the popular agents. We add agents r_{k+1} and p_{k+1}. By popularity, r_{k+1} has $p_1, ..., p_5$ (in some order) as its 5 favorite agents and p_{k+1} has $r_1, ..., r_5$ (in some order) as their 5 favorite agents. Further, let p_{k+1} and r_{k+1} be each other's 6th choices (i.e. top choice outside the popular riders.

Now, consider any stable matching in the old (size-k) instance, and create a stable matching for the new instance by pairing r_{k+1} with p_{k+1}.

We now show that this matching is stable for the new instance. Since it was stable for the small instance, the only possible unstable pairs must involve r_{k+1} or p_{k+1}. By IH, every popular agent is matched to another popular agent. Regardless of where r_{k+1} and p_{k+1} was added to the popular agent's list, they fall after the popular agents, so r_{k+1} and p_{k+1} cannot form an unstable pair with the popular agents. And since they have each other as their next choices, they cannot form an unstable pair with anyone else. Thus we have that there are no unstable pairs, and the matching is stable. The popular agents remain matched to each other, as required. $\square$

(a) There are at least two errors in this proof. Describe them! **Solution:**

The first mistake is in the setup of the inductive step. We need to show a claim for every instance of size $k+1$. Instead we build a particular instance of size $k+1$. In this example, the mistake is quite fundamental – there is no reason in the problem that p_{k+1} and r_{k+1} should have each other as their 6th choices. That is, if we introduced a recursive definition of stable matching instances and changed this to structural induction, we would have more steps to do beyond the one here (In contrast to the tree problem where all the cases are actually handled).

The second bug is again a mistake with handling a for-all. This time, the quantifier on the "every stable matching" part of the statement. We don't check every stable matching! We check every matching we built by starting with a stable matching on the small instance – how do we know there aren't stable matchings where p_{k+1} is matched to r_4 (for example)? We need to start with an arbitrary stable matching and argue whether the popular agents are matched or not.

(b) Write a correct proof of this claim. Do NOT use induction. Use a proof by contradiction instead. **Solution:**

Suppose, for the sake of contradiction, that there is a stable matching instance with 5 popular members of each group, and there is a stable matching M for this instance so that some popular person p from the first group is not matched to any popular member of the second group. Since there are the same number of popular people in the two groups, there is a popular member r of the second group that is also not matched to a popular agent. We claim that p and r form an unstable pair. Indeed, since each is popular, they are each in the top 5 of each other's lists, but each is matched to a non-popular agent, which must be 6th or lower on both lists. Thus p and r would rather be with each other than with their matches, so they form an unstable pair. But M was supposed to be a stable matching. A contradiction! So every popular person in the first group must be matched to a popular person in the second group.

5. Proof Practice: Proving Code Correct

Recall Breadth-First Search from 332.

```
1. BFS(Graph G, Vertex start)
2. initialize queue q to hold start
3. mark start
4. while(q is not empty) {
5.    u = q.dequeue()
6.    for each node v adjacent to u
7.        if(v is not marked)
8.            mark v and q.queue(v)
9.    }
```

We often prove properties of algorithms, e.g only good things happen, by induction on the number of steps. The inductive statement $P(n)$ is basically that only good things happen in the first n steps. The inductive step then requires showing that good things will happen in the next step.

Often this step can be done by contradiction. Namely, one assumes that something bad happens in the next step and derives a contradiction. When we write this proof out, we can save space by writing this directly as a proof by contradiction, where we suppose that the statement is false and we consider the *first* step where something bad happened. We have exactly the same properties we used for the inductive step and the base case: Only good things happened in prior steps!

(The fact that this new way of writing things is equivalent to induction is based on the "well-ordering principle" of the natural numbers – Every non-empty set of natural numbers has a smallest element – which is equivalent to induction.)

Consider the following claim:

For all $n \geq 1$, for the first n times that a vertex v is inserted into `q`, there is a path from `start` to v. (This isn't full correctness for Breadth-First Search which also would require that *all* vertices with a path from `start` are eventually found.)

(a) Prove the claim by induction. **Solution:**

We prove the claim by induction on n.

Base Case: $n = 1$: The first vertex added to `q` is `start` .

IH: Suppose for the first k times that a vertex is added to `q`, at the top of the while loop, the vertex removed from `MPQ` had the correct distance.

IS: Now consider the $(k+1)^{st}$ vertex added to `q`. This only can happen in line 8. Let `v` be the name of that vertex. By the condition on line 6, `v` is adjacent to some vertex `u` that was added to `q` in one of the first k steps. By the inductive hypothesis, there is some path P from `start` to `u`. Since `v` is adjacent to `u` we can add the edge `(u,v)` to P to get a path from `start` to `u`.

Therefore by the principle of induction, every vertex inserted into `q` is reachable from `start`.

(b) Prove the claim by contradiction. **Hint:** Consider the first insertion of a vertex where the statement is false. **Solution:**

We prove this by contradiction:
Suppose that not all vertices added to `q` are reachable from `start`. Consider the first time that a vertex that is not reachable from `start` is inserted into `q` . It cannot be the vertex `start`, since that is reachable from itself. Therefore it must be a vertex `v` inserted in line 8. By the condition on line 6, `v` is adjacent to some vertex `u` that was previously added to `q`. Since `v` was the first vertex that is not reachable from `start`, there is a path from `start` to `u`. Combining this with the edge from `u` to `v` to the path shows that `v` is reachable,

too, contradicting our assumption.

6. Practice A Reduction

Suppose that is a set of r riders and h horses with many more riders than horses; in particular, $2h < r < 3h$. You wish to set up a set of 3 rounds of rides which will give each rider exactly one chance to ride a horse. To keep things fair among the horses, you wish for each to have exactly 2 or 3 rides.

Because it's winter, by the time the third ride starts it will be very dark, so every rider would prefer *any* horse on the first two rides over being on the third ride. Between the first two rides, each rider doesn't have a preference over time of day, and have the same preference over horses. If a rider must be on the third ride, it has the same preference list for that ride as well.

Each horse has a single list over riders, which doesn't change by ride. Since horses love their jobs, they prefer to being one of the horses on the third ride to one of the ones left home.

Design an algorithm which calls the following library **exactly once** and ensures there are no pairs r, h which would both prefer to change the matching and get a better result for themselves.

`BasicStableMatching`
Input: A set of $2k$ agents in two groups of k agents each. Each agent has an ordered preference list of all k members of the other group,.
Output: A stable matching among the $2k$ agents.

(a) Give a 1-2 sentence summary of your idea.

(b) Give the algorithm you're going to run.

(c) Give a 1-2 sentence summary of the idea of your proof.

(d) Write a proof of correctness.

(e) Give the running time of your algorithm; briefly justify (1-3 sentences)

Solution:

We will create a `Basic` instance with $3h$ agents representing horses and $3h$ agents representing riders.

For each horse h_i in the original instance, create three agents $h_i^{(1)}, h_i^{(2)}, h_i^{(3)}$ with copies of h_i's preference list among the riders. Each starts with h_i's original list (we will need to add some more preferences to fill that list out to length $3h$ as we see below).

For each rider r_j, create a list as follows: from r_j's original list, put $h_i^{(1)}$ followed by $h_i^{(2)}$ in place of h_i in the original list. Then at the end, add another copy of the original list with each h_i replaced by $h_i^{(3)}$.

To make the total number of riders equal to $3h$, add "dummy" riders[a] $d_1, ..., d_\ell$ until the number of riders and horses is equal. Each dummy will have a list of the $h_i^{(3)}$, followed by the $h_i^{(2)}$ and $h_i^{(1)}$ (the h_i can be in any order relative to each other, as long as the time-of-day ordering is followed). Finally add the dummies to the end of the preference lists of all horses (in any order).

We now have an instance with $6h$ in two groups of $3h$ agents each, representing riders and horses respectively, and every list contains all the agents in the other group. Run the `BasicStableMatching` algorithm, then delete the dummy riders, and leave any horse whose partner was deleted unmatched.

The `Basic` algorithm doesn't produce unstable pairs, so we won't either (once we delete the dummies)

(b) We claim the result is a correct assignment. First, observe that each (real) rider is matched, and no horse is free on the first two rides. Since each horse prefers the real riders to the dummies and each rider prefers any of the first two rides to the third, a dummy rider matched with a horse on the first two rides would have created an unstable pair (the horse on the first two rides with any rider assigned to the third ride). Thus no horse is free on the first two rides.

It remains to show there is no unstable pair among matched agents. Suppose, for contradiction, there is a pair r, h_i where r and h_i would both prefer to be paired on ride j (over their current state). Then, by construction of the lists, r prefers $h_i^{(j)}$ on its preference list and $h_i^{(j)}$ prefers r on its preference list. This would have been an unstable pair for the `Basic` instance. But the algorithm produces a stable matching, which by definition has no such unstable pairs, a contradiction!

(c) $\Theta(h^2)$. We have $3h$ agents on each side, so the guarantee on `BasicStableMatching` gives a $\Theta(h^2)$ guarantee for that call. All the other operations (copying lists, creating agents, etc.) can be done in time linear in the size of the final instance (since it's just copy-pasting) which is also $\Theta(h^2)$ ($\Theta(h)$ agents, each with lists of length $\Theta(h)$).

[a] a "dummy" is like a dummy for clothing (a mannequin) it *looks like* a real rider, but doesn't actually represent a real rider. Just like a mannequin looks like a real person but isn't one. Dummies are a very common tool in reductions.