Section 1: Stable Matchings and Proof Review

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).

(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?

(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?

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.

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)?

(b) Prove the claim by induction.

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 6^{th} 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. □

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

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

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.

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

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)