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:

$r_1 : h_3, h_1, h_2, h_4$
$r_2 : h_2, h_1, h_4, h_3$
$r_3 : h_2, h_3, h_1, h_4$
$r_4 : h_3, h_4, h_1, h_2$

$h_1 : r_4, r_1, r_3, r_2$
$h_2 : r_1, r_3, r_2, r_4$
$h_3 : r_1, r_3, r_4, r_2$
$h_4 : r_3, r_1, r_2, r_4$

(a) Run the Gale-Shapley Algorithm with riders proposing on the instance above. When choosing which free rider to propose next, always choose the one with the smallest index (e.g., if r_1 and r_2 are both free, always choose r_1).

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

(c) Now run the algorithm with horses proposing, breaking ties by taking the free horse 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:

Let $P(n)$ be "Every tree with at least n nodes has at least two nodes of degree-one."

(a) What is 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 horses and n riders. Of the n horses, 5 of the horses are **popular**. That is, every rider's list has those 5 horses as their first 5 choices (in some order, not necessarily the same for each rider). Similarly, you have 5 **popular** riders, such that every horse has those 5 riders as their top choices.

Let $P(n)$ be "In every stable matching instance with n horses, n riders, of which 5 horses and 5 riders are popular: in every stable matching, popular horses are matched only to popular riders."

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

Base Case $(n = 5)$
With 5 horses and riders, every horse and rider 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 $h_1, ..., h_k$, $r_1, ..., r_k$ be k horses and riders, with $h_1, ..., h_5, r_1, ..., r_5$ being the popular agents. We add agents h_{k+1} and r_{k+1}. By popularity, h_{k+1} has $r_1, ..., r_5$ (in some order) as its 5 favorite agents and r_{k+1} has $h_1, ..., h_5$ (in some order) as their 5 favorite agents. Further, let r_{k+1} and h_{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 h_{k+1} with r_{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 blocking pairs must involve h_{k+1} or r_{k+1}. By IH, every popular agent is matched to another popular agent. Regardless of where h_{k+1} and r_{k+1} was added to the popular agent's list, they fall after the popular agents, so h_{k+1} and r_{k+1} cannot form a blocking pair with the popular agents. And since they have each other as their next choices, they cannot form a blocking pair with anyone else. Thus we have that there are no blocking 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 Dijkstra's Algorithm from 332.

```
Dijkstra(Graph G, Vertex source)
     initialize distances to ∞, source.dist to 0
     mark all vertices unprocessed
     initialize MPQ as a Min Priority Queue
     add source at priority 0
     while(MPQ is not empty){
          u = MPQ.removeMin()
          foreach(edge (u,v) leaving u){
               if(u.dist+weight(u,v) < v.dist){
                    if(v.dist == ∞) //if v not in MPQ
                         MPQ.insert(v, u.dist+weight(u,v))
                    else
                         MPQ.decreaseKey(v, u.dist+weight(u,v))
                         v.dist = u.dist+weight(u,v)
                         v.predecessor = u
               }
```

```
        }
        mark u as processed
}
```

Consider the following claim:

For all n, the n^{th} time a vertex, v, is removed from `MPQ`, `v.dist` contains the true shortest path distance from `source` to v.

(a) Prove the claim by induction.

(b) Prove the claim by contradiction. **Hint:** It's *extreme*ly helpful to think about the closest vertex where something is wrong, rather than just any old iteration.
During your proof, you may use the following fact without proof: Every non-infinite value of `v.dist` is the length of a path from `v.dist` (not necessarily the shortest path).

6. Practice A Reduction

You have a set of r riders and h horses, but unfortunately, $2h < r < 3h$, i.e. there are many more riders than horses. You wish to setup a set of 3 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 k horses and k riders. Each horse has a preference list of all k riders, and each rider has a preference list of all k horses.
Output: A stable matching among the k horses and k riders.

(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)