Stable Matching Sample Problem

1. 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.
Solution:

Create an instance where each horse has 3 copies (one per ride) and extra "fake" riders to balance the sides. Modify the preference list to represent what the agents want in the problem.

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

We will create a `Basic` instance with $3h$ riders and $3h$ horses. For each horse h_i in the original instance, create three horses $h_i^{(1)}, h_i^{(2)}, h_i^{(3)}$. Each copy of the horse starts with h_i's original list.

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 $3h$, 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 lists of all horses (in any order).

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

[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.

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

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

(d) Write a proof of correctness.
Solution:

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 a blocking 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 blocking 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 a blocking pair for the `Basic` instance. But the algorithm produces a stable matching, which by definition has no such blocking pairs, a contradiction!

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

$\Theta(h^2)$. 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 $\Theta(h)$).