Project 3: Oracle

Introduction

In this assignment, you will develop an oracle to test (possibly broken) solutions to the *Stable Marriage Problem*. You can learn more about the problem from Wikipedia, but a summary of the problem is as follows:

Assume you have two sets, A and B, where each set has *n* members.Every member in A wants to be matched with exactly one member of B and vice versa. Every member of each set ranks its preference for being matched with each member of the other set by assigning each one a unique number between 0 and n-1 (i.e. providing a total order of members of the other set).

For this assignment, imagine matching companies with candidates. Each company will keep an internal ranking of all the candidates, based on who they would prefer to hire (we assume that only one candidate will be hired per company). Each candidate also has an opinion about where they want to work and therefore ranks each company as well. A programmer is asked to design and implement a program that generates n pairings of n companies with n candidates. The program's generated "hires" are stable if there are no two members, one from each set, that would prefer each other to their current match.

There are many problems of this nature. Consider assigning TAs to classes; matching residents with hospitals; pairing students for homeworks; and much more. Of course, some of these problems represent a slight variation on the theme (maybe the companies don't rank every candidate, or are allowed to give some of them the same rank; maybe you only have partial information for making the assignment; etc.). Ultimately, however, this problem in its many guises has wide application.

Assignment Overview

Being confident that your software is correct involves more than just writing code you think is right. However, almost no software complex enough to be useful can be proved correct by hand in a reasonable amount of time. Naturally, a computer scientist's solution to this problem is to write automated testing. Your job in this assignment is to build an automated testing oracle for a hypothetical solution to the stable marriage problem.

Your oracle's job is to generate and feed test inputs to a given solution and test the correctness of the output. In the past, you did this by comparing the output to a precomputed right answer. This assumes two things: that there is only one right answer, and that it is easy for you to find it. In the real world, either of these or both can be false. (How do you know what the right answer to an arbitrary instance of the problem is if the original problem was to write a program to find it?)

You've got your work cut out for you: we give you both a correct and incorrect solution of the stable marriage problem to help you write and test your oracle. You submit your oracle, and we will evaluate its performance in classifying various implementations as correct or incorrect.

Input-Output Specification

We have added a number of functions to Ocelot that you can use in this assignment:

```
type Hire = { company: number, candidate: number }
wheat1(companies: number[][], candidates: number[][]): Hire[]
chaff1(companies: number[][], candidates: number[][]): Hire[]
```

Both `wheat1` and `chaff1` are solutions to the stable marriage problem. However, **`chaff1` has bugs**, whereas `wheat1` is a correct solution. The inputs to these functions are the preferences of each company and candidate. i.e., `company[i]` are the preferences of the *i*th company and `candidate[j]` are the preferences of the *j*th candidate. The number of companies and candidates must be the same (call the number *N*).

The preferences of a candidate are represented as an array of *N* distinct numbers from 0 to *N* - 1. For example, if a candidates preferences are the array [2, 1, 0], that means that that candidates most preferred company is company 2 and least preferred company is company 0. The preferences of a company are represented in the same way, but rank candidates instead.

For example, suppose `companies` is the array `[[0, 1], [1, 0]]`. The 0th element which is the 0th companies preference list, `[0, 1]` denotes that it prefers the 0th candidate best (because 0 is the first element of the array) and the 1st candidate second. The 1st element of the `companies` array, `[1, 0]` denotes that the 1st company prefers the 1st candidate best and 0th candidate second.

The result of both `wheat1` and `chaff1` are an array of `Hire` objects, where each object matches a company with a candidate. The set of hires returned are not required to be in any particular order. The solutions that we use for evaluation will follow the same format as these functions.

Programming Task

1. Write the following function:

   ```
   generateInput(n: number): number[][]
   ```

   This function should produce an n by n array of preferences for companies or candidates. The input generated will be used for testing a given solution. Above, we have described only the shapes of the inputs; you will have to infer the constraints we've left out. Make sure that your function always generates random values as it will be helpful to test a given solution with a broad spectrum of input.

2. Write the following function:

   ```
   oracle(f: (candidates: number[][], companies: number[][]) => Hire[]): void
   ```

   This function should test the provided implementation of stable-marriage. Remember, an implementation may sometimes produce a correct solution even if it is an incorrect implementation. At the same time, there are numerous ways for an algorithm to return an incorrect solution. A template for the `oracle` function is included below. To do well on this assignment, you will want to spend time considering all the different ways that output could be

either invalid or inconsistent with the original problem statement. Be thorough! That's the name of the game when testing.

```
function oracle(f) {
 let numTests = 20;  // Change this to some reasonably large value

 for (let i = 0; i < numTests; ++i) {
   let n = 6;  // Change this to some reasonable size
   let companies = generateInput(n);
   let candidates = generateInput(n);
   let hires = f(companies, candidates);

   test('Hires length is correct', function() {
     assert(companies.length === hires.length);
   });

   // Write your tests here
 }
}
```

You can then call oracle on the test inputs with either `oracle(wheat1)` or `oracle(chaff1)`. You can press on the "Test" button to run the tests in `oracle`. Note that this `oracle` function does not return anything. The goal of `oracle` is to have all its tests successfully pass when given a correct implementation and at least one test fail when given a faulty implementation.

Generating Random Numbers

This assignment requires you to generate random integers. This can be done using a combination of `Math.random()` and `Math.floor()`.

```
// Returns a random int i where min <= i < max
function randomInt(min, max) {
 return Math.floor(Math.random() * (max - min)) + min;
}
```

Note: For this assignment and this assignment only, we will be providing **full autograder results before the deadline**.

Additional Note: Even though your goal for this assignment is to test different implementations of an algorithm, you should still test and verify any helper functions you call within your oracle.