CPSC 320 Notes, The Stable Matching Problem

Get into a group of three people and come up with the names of three fictional companies and three applicants. Also come up with the preferences of the companies and the applicants. One way to do this is after you have the names of the company and applicants, each of you can write down the preference for one of the companies and one of the applicants

Wait at this point for a class activity. **While you're waiting**, get to know your group by telling them your name, major, and the most exciting thing you're planning to do this term. Once we're done with the activity, we'll explore the stable matching problem (SMP) using your tasty preferences.

1 Trivial and Small Instances

1. Write down all the **trivial** instances of SMP. We think of an instance as "trivial" roughly if its solution requires no real reasoning about the problem.

2. Write down two **small** instances of SMP. One should use the preference lists your group gave for the employer/applicant example above:

   The other can be even smaller, but not trivial:

3. Hold this space for another instance, in case we need more.

Fun fact: people tend to stop at the end of the page instead of going on. **GO ON UNTIL YOU'RE STUCK!**

2 Represent the Problem

1. What are the quantities that matter in this problem? Give them short, usable names.

2. Go back up to your trivial and small instances and rewrite them using these names.

3. Use at least one visual/graphical/sketched representation of the problem to draw out the largest instance you've designed so far:

4. Describe using your representational choices above what a valid instance looks like:

3 Represent the Solution

1. What are the quantities that matter in the solution to the problem? Give them short, usable names.

2. Describe using these quantities what a **valid** solution looks like:

3. We haven't said what a **good** solution to this problem looks like. Brainstorm some different possible definitions of a good solution.

4. There are many reasonable ways we could define a **good** solution to this problem. However, for the remainder of this worksheet we will use **one particular** definition. **WAIT HERE** for us to provide that definition!

5. Go back up to your trivial and small instances and write out one or more good (as we've defined "good") solutions to each using these names.

6. Draw out a solution to your graphical/visual/sketched representation from earlier.

4 Similar Problems

As the course goes on, we'll have more and more problems we can compare against, but you've already learned some. So...

Give at least one problem you've seen before that seems related in terms of its surface features ("story"), problem or solution structure, or representation to this one:

5 Brute Force?

You should usually start on any algorithmic problem by using "brute force": the most straightforward, non-optimized approach you can think of. In this case (and in my cases), we want to generate all possible solutions and test each one to see if it is, in fact, **the** solution we're looking for.

1. A possible SMP solution takes the form of a perfect matching: a pairing of each applicant with exactly one employer. We'll call a perfect matching a "valid" (but not necessarily good) solution.

   It's more difficult than the usual brute force algorithm to produce all possible perfect matchings; instead, we'll count how many there are. Imagine lining all the employers up in a row in a particular order. How many different ways we can line up (permute) the applicants next to them?

   (This is the number of "valid solutions". Note that to solve this, you must also choose a way to measure the size of an instance and give it a name. **Naming things** is incredibly important. Do it!)

2. Once we have a possible solution, we must test whether it's the solution we're looking for. Informally, we'll refer to this as asking whether it's a "good" solution.

   A perfect matching is a good solution if it has no instabilities. Design an algorithm that—given an instance of SMP and a perfect matching—determines whether that perfect matching contains an instability.

3. Exactly or asymptotically, how long would a brute force algorithm for the SMP problem take? (Recall: a brute force algorithm would generate every possible perfect matching, and then use the algorithm you designed in question 2 to check if the solution is stable.)

4. Brute force would generate each valid solution and then test whether it's good. Will brute force be sufficient for this problem for the domains we're interested in?

6 Promising Approach

Unless brute force is good enough, describe—in as much detail as you can—an approach that looks promising.

7 Challenge Your Approach

1. **Carefully** run your algorithm on your instances above. (Don't skip steps or make assumptions; you're debugging!) Analyse its correctness and performance on these instances:

2. Design an instance that specifically challenges the correctness (or performance) of your algorithm:

8 Repeat!

Hopefully, we've already bounced back and forth between these steps in today's worksheet! You usually *will* have to. Especially repeat the steps where you generate instances and challenge your approach(es).