The Unequal Stable Marriage Problem

September 9, 2016

So far, women have had an unfair disadvantage in what we might call the Chauvinistically Optimal Men-Propose Gale-Shapley Algorithm.[1] Let's shake things up a bit.

Imagine a similar problem to SMP, except there are at least as many men as women, and maybe **many** more. Characterize and solve the problem, i.e., finding the equivalent of SMP's stable perfect matchings but adjusted for this context. We'll call this the "Unequal Stable Marriage Problem" or USMP for short.

1 Trivial and Small Instances

1. Write down all the **trivial** instances of USMP. 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 USMP. Here's your first:

   The other can be even smaller, but not trivial:

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

[1]If you need more of a break from patriarchy, check out Fresh Air's awesome Gloria Steinem interview.

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 makes a solution **valid** and **good**:

3. Go back up to your trivial and small instances and write out one or more solutions to each using these names.

4. Go back up to your drawn representation of an instance and draw at least one solution.

4 Similar Problems

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?

We have a way to test if something with the form of a solution (i.e., looks like a solution but may not be valid or good) is actually **valid** and **good**. (From the "Represent the Solution" step.)

1. Sketch an algorithm to produce everything with the "form" of a solution. (It will help to **give a name** to your algorithm and its parameters, *especially* if your algorithm is recursive.)

2. Choose an appropriate variable to represent the "size" of an instance.

3. Exactly or asymptotically, how many such "solution forms" are there? (It will help to **give a name** to the number of solutions as a function of instance size.)

4. Exactly or asymptotically, how long will it take to test whether a solution form is valid and good with a naïve approach? (Write out the naïve algorithm if it's not simple!)

5. Will brute force be sufficient for this problem for the domains we're interested in?

6 Lower-Bound

In terms of instance size, exactly or asymptotically, how "big" is an instance? (That is, how long will it take for an algorithm just to read the input to the problem?)

(We won't do more for now to lower-bound the problem.)

7 Promising Approach

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

8 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:

9 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).