CPSC 320 Notes, The Stable Marriage Problem

August 25, 2017

The major goal of CPSC 320 is, of course, romantic advice. That's a heavy topic over which to meet your classmates. So, we use candy and baked goods to stand in for love (a surprisingly common proxy).

Get in a group of three. Each of you write down your preferences among the candies **Werthers originals**, **Mars bars**, and **Smarties** and, separately, among the baked good **brownies**, **rose spirals** (gluten-free cornstarch cookies), and **thin mints** (homemade version of the Girl Guide cookies).

Wait at this point for a class activity. **While you're waiting**, get to know your group by telling the story of your best experience in a course. Once we're done with the activity, we'll explore the stable marriage 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 be your candy/baked goods example above:

   The other can be even smaller, but not trivial:

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:

(**Still and always** go to the next page if you finish early! There are even challenge problems at the end.)

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

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": 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 woman with exactly one man. 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 men up in a row in a particular order. How many different ways we can line up (permute) the women 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 a (brute force!) algorithm that—given an instance of SMP and a perfect matching—determines whether that perfect matching contains an instability. (As always, it helps to **give a name** to your algorithm and its parameters, *especially* if your algorithm is recursive. Remember, for brute force: generate each possible solution (possible instability, in this case) and then test whether it really is a solution. Be careful: a possible instability is two people who would rather be married to each others than their partners, not an already-married couple.)

3. Exactly or asymptotically, how long does your algorithm take? (Again, you should explicitly name the size of an instance and perform your analysis in temrs of that name!)

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).

9 Challenge Problems

These are just for fun and are rated easy/medium/hard as a scale of difficulty **for challenge problems with no guidance**. "Easy" is already plenty hard enough.

1. **Easy** (and well worth doing!): Design an algorithm to generate each possible perfect matching between n men and n women. (As always, it will help tremendously to start by giving your algorithm and its parameters names! Your algorithm will almost certainly be recursive.)
2. **Easy**: Prove that a man willing to pay another man to lie about his preferences can improve his own result in the "man-oriented" G-S algorithm.
3. **Medium**: A "local search" algorithm might pick a matching and then "repair" instabilities one at a time by pairing the couple causing the instability and their spurned partners. Use the smallest possible instance to show how bad this algorithm can get.
4. **Medium**: Design a scalable SMP instance that forces the G-S algorithm to take its maximum possible number of iterations. How many is that? (A "scalable instance" is really an algorithm that takes a size and produces an instance of that size, just like the "input" in worst case analysis is scalable to any n.)
5. **Hard**: Prove that no set of men can collaborate to lie about their preferences and improve **all** of their results in the man-oriented G-S algorithm.