CPSC 320 Notes, The Stable Marriage Problem

September 7, 2016

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 **M&M's**, **Resse's peanut butter cups**, and **Snickers** and, separately, among the baked good **brownies**, **Nanaimo bars**, and **tenacious death spirals**.

Wait at this point for a class activity. **While you're waiting**, tell the story of your best experience in a course to your group. 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:

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:

(**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?

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 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).

10 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**: 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.
2. **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.
3. **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.)
4. **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.