Problem Solving in Computer Science

Jacob Steinhardt

The major focus of SCT is on the USA Computing Olympiad, or USACO. Some of the problems can be pretty intimidating, but always remember, they are made by humans for humans. Up to the gold division, at least, every problem is definitely accessible if you know how to think about it the right way. The goal of this lecture is to help you think the right way.

1 Problem 1: Hypnotic Milk Improvement (2002 USACO Green Division)

Problem: FJ hires a dairy hypnotist to improve milk production using hypnotic suggestions. FJ knows how much milk each cow currently has for milking (an integer amount) and has asked the hypnotist to manipulate the cows to maximize total milk production. FJ has arranged 36 cows on a 6x6 grid.

The hypnotist can spin his watch so that it is seen by the six cows in any one row, column, or the major diagonal (upper left to lower right). His power of suggestion can increase or decrease the amount of milk a cow can give by one liter. Of course, if he asks for too much milk (more than 9 liters), the cow gets confused and will give 0 liters. Similarly, a cow asked to reduce from 0 liters will be confused and give 9 liters instead. Using an unlimited number of hypnotic suggestions, maximize the amount of milk that FJ can expect.

Step 1: Naieve Approach. It's always best to start a problem by trying the most obvious thing. Unfortunately, here there is not even any clear approach, because we are allowed "an unlimited number of hypnotic suggestions." So how do we know when we are done? This is definitely a problem! As we can't get anywhere without doing something about this, we should start by focusing our efforst here.

Step 2: Make observations to limit the search space. When a problem appears difficult to approach algorithmically, it's often best to try to make some observations about the particular situation that allows one to consider less cases than before. Here, we note that the order of hypnotisms doesn't matter (since addition is commutative) and furthermore that ten of the same hypnotism is the same as none at all, so we at least now have a finite number of cases to check. Returning to the naieve approach, we should see exactly how many cases there are. There are 10 possibilities for each row, column, and the major diagonal, for a total of 10^{13} possibilities. This is definitely too much, so we should see what else we can do.

Step 3: Try to improve on the naieve approach. 10^{13} is way too much, but 10^7 is invitingly within our limit on number of operations. Note that we don't actually need to try all the possibilities for the rows, since once we have made our choices for the columns and main diagonal, the rows behave independently. In fact, as there are only 10^6 possibilities for what a row looks like, we can optimize further by precomputing, for each row, what the best achievable value is. This gets us down to the 10^7 operations we wanted.

Step 4: Optimize further. However, 10^7 is close to the limit, so we make one more optimization just in case. Note that we really haven't exploited any of the symmetry of the situation yet. We do

so now. Since applying a hypnotism to every column is essentially the same as applying 9 to each row, we can assume that no hypnotisms are applied to the first column (else apply 9 hypnotisms on every row and 1 on every column until we have cancelled out the hypnotisms that we had initially applied in the first column). This lets us cut off another factor of 10, so that we now only use 10^6 operations, which is well within our bounds.

Ideas used: Naieve approach, reduce to a manageable search space, optimize through precomputation, optimize through symmetry.

Comment: The above problem is said to be one of the toughest USACO problems, but it really wasn't all that bad, just required multiple observations. So remember, never be afraid of a problem, there's always a reasonable solution.

2 Problem 2: Topcoder High School SRM Hard Problem, RangeFixer (SRM39)

Problem statement: Given a a, b, c, return the smallest x, $b \leq x \leq c$, such that the *bit difference* between a and x is minimal (among all x in the range $[b, c]$. The bit difference of two numbers is the number of places in which their binary representations differ. Note: a, b, c can be as large as $2^{30} - 1$.

Step 1: naieve approach. Here the naieve approach is pretty obvious: loop through all x in the range $[b, c]$, calculate the bit difference, and return the lowest one. The problem with this is that b and c can both be huge! We thus need a way to look only at the "important" numbers in the range.

Step 2: reduce search space. We would ideally like to be able to look at only the smallest number in $[b, c]$ with any given bit difference, or something along these lines, as that would limit our search space. The reason I chose the above reduction to try for is that it still effectively solves the problem but is a smaller step than solving it outright, so is probably easier to design an algorithm for. In fact, I think that you can convince yourself that it's an even better idea to fix both the number of ones and zeroes we change as separate entities.

Your turn! With this suggestion for how to try approaching the problem, it's now your turn to see what you can do. Feel free to work on the problem in groups to solve it. After you solve it, make a list of the key ideas used.