USACO Contest Strategy

Andre Kessler

October 23, 2009

What is USACO about? Coding, algorithms or data structures? Answer: algorithms, then data structures, then coding. Hence, a significant portion of the contest should be devoted to thinking about the problems. How much depends on time constraints, but on a three hour contest it is usually advisable to spend a good 30-45 minutes thinking.

1 The Contest

1. Reading
   - Print the problems and turn off your computer screen (or a least look away)
   - Read ALL of the problems first so that you can have a rough estimate of how long the contest will take and what order you should solve the problems in
2. Solving
   - Come up with a naive/brute force solution first so you have something to fall back on
   - If you come up with multiple algorithms, pick the simplest that works (in time)
   - Check the correctness of your algorithm on the sample input and small cases
   - Will your algorithm run in time?
   - Come up tricky cases and try to break your algorithm
   - Don't be seduced by "obvious" greedy algorithm solutions - half of the time, they're wrong. Try to find counter-examples.
   - DON'T start coding until you've solved everything that you'll solve - once you start coding, you won't be able to think clearly anymore
   - Before coding, try to estimate how many lines of code each problem will be to determine the order in which you code the problems
3. Coding
   - You want to code a correct program as quickly as possible. This is not coding as quickly as possible. This is coding a CORRECT program as quickly as possible. Spending more time on coding initially may allow you to avoid a bug that you discover five minutes before the end of the contest that makes your program totally incorrect.
   - Test your code while you're writing it - it's easier to check small parts than debug a huge multipart program
   - Comment stuff that you don't think you'll remember in a few hours since you WILL need to remember it then
   - Before optimizing a non-trivial portion of your program, save a copy of the old version

4. Testing

   - Try to leave the last 30-45 minutes of the contest to testing your solutions
   - Write a slow but obviously correct solution to the problem, and write a script to generate test cases
   - Test your fast program against your slow program's output for a few thousand testcases or so with diff
   - Test edge cases
   - Check your program against a few very large testcases to make sure you aren't producing nonsensical output due to integer overflow

5. Debugging

   - You don't want to do much of this, so try not to make bugs in the first place by testing components of your program as you write them. Example: computational geometry, programs with lots of data structures
   - If you've spent 45 minutes debugging a program, look at the clock and see if it will be more beneficial for you to keep going - or cut your losses and move on
   - Use gdb for segfaults
   - Use assert () in your code

2 Additional Tips

- Solve problems completely; one full solution and two hacks are better than three bad solutions
- **Keep a log of each contest** - this allows you to see how much time you've spent in each area, and you can refer back to the log to see how you could have improved your strategy
- Take a short break about halfway through the contest! It's less than 3% of the total time on a three hour contest anyways.

Example:

```
5:00 - #1 is confusing
4:55 - #2 is a modified dijkstra
4:54 - #3 is a classic DP / or rabin-karp
4:35 - Thought of a fast way O(NlogN) for #3, and it should have a favorable constant - ‘‘manually’’ check for length = 2, then
binary search on longest string.  use rabin karp for fast comparison.
4:25 - fully wrote out functions for #3 (rabin karp, binary search, strcmp)
4:00 - thought about #1, it seems to be by far the messiest, and pretty annoying.  #2 is not all that hard and works on N = 100,
A_(i,j) ≈ 2000000000 in < 0.04 s, moving on to #3
3:59 - done thinking.  order is #2, slow #3, fast #3, #1.
3:40 - done with himtn
3:39 - debugging himtn
3:32 - done debugging himtn, starting to test
3:20 - found a bug, spent some time debugging, fixed bug.
3:13 - himtn works on N = 100, A_(i,j) ≈ 2000000000 in < 0.04 s, moving on to #3
2:35 - still working on slow patterns, though the fast one will be able to use a lot of the slow one's code.
2:26 - done with slow, debugging slow
2:23 - darn, it's finding pairs of patterns, not the number of patterns
2:00 - plan for rest of the contest:  finish fast #3 with rabin karp, test against slow one.  if there's time, try to find a DP
for #3, code that, test it, and submit it if it's faster than the rabin karp.
1:55 - got back from lunch.  will fix hashing next.
1:45 - figured out I can make my slow solution pretty fast with a binary search
0:55 - rabin karp is a bit messy but seems to work
0:15 - suffixes work, will submit fast solution after some testing
0:06 - fast #3 works, going back to do a bit more #2 testing
```