Contest Strategy

Andre Kessler

USACO is about algorithms, algorithms, algorithms, data structures, and 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 look away from your computer screen.
   - 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; this generally goes
     (a) Problems you've done before
     (b) Problems you can easily see how to solve
     (c) Other
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? Are you SURE your algorithm will 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.
   - Do NOT start coding until you've solved everything that you think 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. I reiterate: this is coding a CORRECT program as quickly as possible. Spending more time on coding initially may allow you to avoid a bug you discover five minutes before the end of the contest that makes your program totally incorrect.
   - Test code while you're writing it – it's easier to check small parts than to debug a huge multipart program. This especially holds true for computational geometry problems, in which you need many small functions to be correct. Try to alleviate this "correctness creep" at all costs; if you're a C++ user, you should probably be using <complex> on any computational geometry that requires cross products, rotations, lots of distance calculations, etc.
   - Comment stuff that you don't think you'll remember in a few hours since you may well need to remember it then.
   - Before optimizing a non-trivial portion of your program, save a copy of the old version (prog-name.old.cc or something).

4. Testing

    - Try to leave the last 30-45 minutes of the contest to testing your solutions. If it's a 5-hour contest, try to leave as much as an hour.
    - 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. An example of a script to do so is below:

    ```
    #!/bin/bash
    for i in ‘seq 1 10000‘
    do
        ./prog.generate
        ./prog.slow
        mv prog.out prog.slow.out
        ./prog
        if diff prog.out prog.slow.out
        then
            echo $i OKAY
        else
            echo $i FAIL
            exit
        fi
    done
    ```

    - 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 or some other weirdness.

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 finding where segfaults happen.
    - Use assert () in your code.

2 Additional Tips

- Try to solve problems completely; one full solution and two hacks are better than three bad solutions. That being said, if you can get partial credit solutions, by all means submit them!
- **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. After a USACO contest, send your log to one of the officers, so we can go through it with you and tell you how to improve your strategy.
- Take a short break about halfway through the contest! Get up, have a drink, whatever – it'll clear your mind and you can re-evaluate how to best spend the rest of your time.