CS35X: Competitive Programming

Lecture 6: Debugging

Joshua Brody

Quiz 1

Contest Strategies

- **Identify Easy Problems**
  - Practice identifying which problems are easiest
  - Solve easy problems fast.
  - Ignore harder problems unless your goal is to solve everything
- **Upsolving Problems**
  - Solve a problem after contest completes, without time pressure
- **Technical Resource Document (TRD)**
  - Printout of notes about different data structures, algorithms
  - Good for obscure, not-often-used algorithms you've already implemented.

Debugging Strategies

Before Submitting

- At a ***minimum***, make sure your program compiles, runs on sample inputs.
- Perform sanity checks:
  - Does program work on minimal inputs e.g. n==0, n==1?
  - Does program use all of given input?
  - Have you removed all debugging print statements?
  - (C++ integers): does your answer fit inside `int` or `long long`?
  - Is the output formatted correctly? (reread output spec)

Debugging Strategies

Before Submitting

- Compile/test your program locally.
  - `g++ -std=c++11 -o solution solution.cpp`
  - `./solution`
- Redirect input/output
  - `./solution < in.txt > out.txt`

Debugging Strategies By Verdict

Wrong Answer (WA)

- WAs are very common and hard to debug :(
- Can be useful to try to identify type of bug
  - **Optimization** Problems: program found suboptimal solution or a solution that was too good to be true
  - **Counting** Problems: program over- or under-counted the answer
  - **Constructive** Problems: program failed to find a construction where one existed, or provided incorrect construction
- Bugs can be generally classified as errors in *implementation* or in *reasoning*.

Debugging Strategies By Verdict

Time Limit Exceeded (TLE) — there are two main reasons for TLEs:

- Incorrect asymptotic complexity
- Program's runtime has poor constant factor
- Data structures like **map** or **set** have high constant factor
- C++ printing a lot of output can be slow.
  - Make sure you're using `'\n'` instead of `endl`.
- Generate a large sample input and time your code running on it.

Stress Testing

- After getting a WA result, write two programs:
  - A slow solution you know is correct (e.g. brute force)
  - A program that prints random (small!) test cases
- Then, use the program to generate several test cases, and run both your WA and the slow solution to compare differences
- Power Stress Testing: write a *shell script* to automate process.

Kattis problem: fadingwind