Test Your Algorithm

/run/media/lijunjie/资料/Learning Video/writing-running-fixing-code/01_writing-code/02_more-about-steps-1-4/06_revisiting-step-4_instructions.html

Once you have generalized your Algorithm, it is time to test it out. To test it out, you should work it on *different* instances of the problem than the one(s) you used to come up with it. The goal here is to find out if you mis-generalized before you proceed. We have already seen one instance of mis-generalization in our rectangle problem, in which our algorithm was too specific to the examples from which we had built it (always using r1's bottom, r2's left, *etc* ...). Testing on these same examples would not have revealed any problems.

In doing this testing, you want to strike a balance—enough testing to give you confidence that your algorithm is correct before you proceed, but not an excessive amount of testing. Note that in this testing, you perform your steps by hand, so it may be somewhat slow for a long or complex algorithm. You can do more extensive testing after you translate your algorithm to code. The tradeoff there is that the computer will execute your test cases (which is fast), but if your algorithm is not correct, you have spent time implementing the wrong algorithm.

Here are some guidelines to help you devise a good set of test cases to use in Step 4:

- Try test cases that are qualitatively different from what you used to design your algorithm. In the case of our rectangle example, the two examples we used to build the algorithm were both fairly similar, but the third example (which we used to show the flaw) was noticeably different—the rectangles overlapped in a very different way.
- Try to find *corner cases* —inputs where your algorithm behaves differently. If your algorithm takes a list of things as an input, try it with an empty list. If you count from 1 to N (inclusive), try N=0 (you will count no times) and N=1 (you will count only one time).
- Try to obtain *statement coverage* —that is, between all of your test cases, each line in the algorithm should be executed at least once. We will discuss various forms of test case coverage later in the course.
- Examine your algorithm and see if there are any apparent oddities in its behavior (such as it always answers "true", it never answers "0" even though that seems like a plausible answer), and think about whether or not you can get a test case where the right answer is something that your algorithm cannot give as its answer