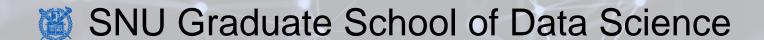
Testing and Debugging

Lecture 13

Hyung-Sin Kim



Quality Assurance (QA)

- QA check that software is working correctly
 - It is not something that is magically done after you write a program, even though you think your logic is perfect
- Before finalizing and releasing your code, you need to write another program for test your code (QA!)
 - You may not want to put enough time for testing since it seems a tedious, time-consuming job
 - But if you don't do QA and find bugs later, you will waste 10x or 100x more time
 - So... yes! QA actually saves time and improves productivity (investment!)
- Test program should be **reusable** so that anyone using your code can test it

Example – Testing above_freezing

- We want to make a function that determines if an input temperature (celsius) is above freezing point or not
 - def above freezing(celsius: float) -> bool:
 - # Some implementation
- But now, we focus more on **test cases** for the function than implementation.
- There are numerous values we can test... What are reasonable test cases?
 - Testing a value **above** freezing point: above_freezing(5.2) \improx True
 - Testing a value **below** freezing point: above_freezing(-2) \impress False
- Are they sufficient? ©

3

Example – Testing above_freezing

- Let's consider two implementations
 - def above freezing v1(celsius: float) -> bool:
 - return celsius > 0
 - def above_freezing_v2(celsius: float) -> bool:
 - return celsius ≥ 0
- Can we distinguish which is correct by using previous test cases?
 - No... We need to additionally test **above_freezing(0)!**
 - It is a boundary case since it lies on the boundary between two different behaviors
- Boundary cases are much more likely to contain bugs than other cases
- Overall, above freezing requires at least three test cases

Example – Testing running_sum

- We want to make a function that modifies a list so that it contains a running sum of the values in it
 - def running sum(L: list) -> None:
 - for i in range(len(L)):
 - L[i] = L[i-1] + L[i]
- Now instead of a return value, we need to look into the input list after the function runs

Example – Testing running_sum

• Test cases. Do you see any failure? Why?

	List input	List output (expected)
Empty list		[] (unchanged)
One-item list	[5]	[5] (unchanged)
Two-time list	[2,5]	[2, 7] (only second value changed)
Multiple items, all negative	[-1, -5, -3, -4]	[-1, -6, -9, -13] (monotonic decrease)
Multiple items, all zero	[0, 0, 0, 0]	[0, 0, 0, 0] (unchanged)
Multiple items, all positive	[4, 2, 3, 6]	[4, 6, 9, 15] (monotonic increase)
Multiple items, mixed	[4, 0, 2, -5, 0]	[4, 4, 6, 1, 1] (ups and downs)

Example – Testing running_sum

- Correct implementation ©
 - def running sum(L: list) -> None:
 - for i in range(1, len(L)):
 - L[i] = L[i-1] + L[i]

How to Choose Test Cases?

Your test cases need to have good coverage

- Size
 - Empty collection, the smallest interesting case, one item, general cases with several items
- Dichotomies
 - If a function deals with two or more different categories, make sure you test all of them
- Boundaries
 - If a function behaves differently around a particular boundary, test exactly that boundary case
- Order
 - If a function behaves differently when values appear in different orders, identify those orders and test each one of them.

8

Guidelines for Bug Hunting

- Make sure you know what the program is supposed to do
 - Manual calculation, reading documents, writing a test
- Repeat the failure (**reproducible** error)
 - Find a test case that makes the program fail reliably
- Divide and conquer
 - Try to find the first moment where something goes wrong by examining input/output of a block of code

Guidelines for Bug Hunting

- Change one thing at a time for a reason and check!
 - Replacing random bits of code that might be responsible for your problem is unlikely to do much good...

- Keep records
 - You cannot remember the results of the tests you've run

Summary

Quality assurance (QA)

Test case generation

Bug hunting

Thanks!