ECE 322 Lab Assignment 3

Unit and Pairwise Testing (White-box Testing #1)

Overview

The objective of this laboratory assignment is to become familiar with the rudimentary techniques of white-box testing, specifically unit testing. Also, this assignment introduces us to the pairwise test case generation tools. This lab makes use of Python, Pychram, the unittest testing framework and allpairspy pairwise generation python package tool .

Introduction

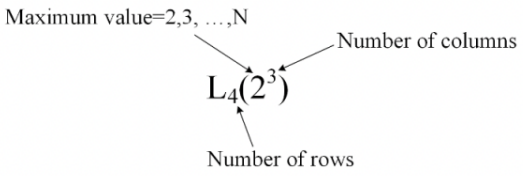
White-box Testing:

White-box testing focuses on testing the internal structure and implementation details of the application. In contrast to black-box testing, this style of testing requires the tester to have detailed knowledge of the internal structure of the software under test, usually in the form of direct access to the source code.

In this lab we will concentrate on **control flow testing** where we consider four fundamental coverage criteria: *statement coverage, branch coverage, condition coverage, and path coverage.*

Pairwise Testing: Instead of considering all possible combinations of input variables, pairwise testing provides an efficient way to construct a set of test cases that covers all combinations of the test data for each pair of variables. **The Orthogonal Array Test Strategy (OATS)** is a widely used technique of testing the pair-wise interactions of variables. An orthogonal array can be expressed in the following way:

	1	2	3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1



Frequently, the exact orthogonal array we need for the problem at hand is not available, in this scenario we can select an orthogonal array which is slightly “too large” and modify it to suit our needs, or use approximation of orthogonal arrays to generate reduced, though perhaps not minimal, test cases. Due to the popularity and effectiveness of pairwise testing, many tools have been developed to help generating pairwise test cases without the need for the proper orthogonal array to be readily available, or the correct array to be time consumingly found in a database.

One such tool, is **allpairspy** python library which computationally calculates approximations to orthogonal arrays which are then used to generate pairwise test cases.

The tool can be obtained at: https://pypi.org/project/allpairspy/
documentation is available at: https://pypi.org/project/allpairspy/, and
https://github.com/thombashi/allpairspy

A simple python script is available as an example on eclass

Note that the results produced from **allpairspy** are algorithmic approximations to orthogonal arrays and may be less efficient in terms of the number of test cases than using a true orthogonal array

Preparation: Make sure you have Pychram, Python3 and Virtualenv/Pipenv installed on your account/computer. The code for this experiment is available on the class website and can be imported into a Pychram python project once extracted. You may need to install some python packages in order for the test cases to compile and run, check the source file to find all the required python packages.

Task 1 (50 marks)

The bisection method is a simple way to solve equations in the form of $f(x) = 0$ given an interval in which the root can be found. The source for this project contains some code which executes the bisection method to find the root of an equation. The source code is available on the class website. First, **draw a control flow graph** for the algorithm, then generate test cases that comply with:

- Statement coverage
- Branch coverage
- Execution each loop body at least twice (enter the loop and repeat)

Submit your test cases as **unittest** tests and compare results based on the above coverage criteria in your written report. Constructors and error cases are included in the requirements for code coverage. For your report you must include both:

- The test cases code which can be executed against the provided lab project
- A **test case table** similar to those used in black box testing containing a meaningful description for each test case, the expected result, and the actual result
- The HTML generated coverage reports

In addition to the requirement of code coverage, keep in mind that for full marks **your test cases are required to assert meaningful things regarding the functionality of the code** and the results of your tests. For example, a successful use of the bisection method should assert that a root value is returned from the algorithm within the given error threshold. Provide a brief discussion on the effectiveness of these coverage criteria. Based only on the coverage criteria, how confident are you in the bug free nature of the code? Do tests fulfilling these criteria

adequately cover the full functionality of the algorithm? How many tests would be required to fulfil path coverage?
As always, your test cases should be executed, and the results recorded into a **test case table** to be handed in with your report. Failed test cases must be highlighted, and the reasons for failure must be identified.

Task 2 (35) :

A mock object substitutes and imitates a real object within a testing environment. It is a versatile and powerful tool for improving the quality of your tests.
One reason to use Python mock objects is to control your code's behavior during testing.

For example, if your code makes HTTP requests to external services, then your tests execute predictably only so far as the services are behaving as you expected. Sometimes, a temporary change in the behavior of these external services can cause intermittent failures within your test suite.

Because of this, it would be better for you to test your code in a controlled environment. Replacing the actual request with a mock object would allow you to simulate external service outages and successful responses in a predictable way.

Sometimes, it is difficult to test certain areas of your codebase. Such areas include except blocks and if statements that are hard to satisfy. Using Python mock objects can help you control the execution path of your code to reach these areas and improve your code coverage.

Another reason to use mock objects is to better understand how you're using their real counterparts in your code. A Python mock object contains data about its usage that you can inspect such as:

- If you called a method
- How you called the method
- How often you called the method
- Understanding what a mock object does is the first step to learning how to use one.

MathPackage is python package for mathematical operation on a list of numbers.

First :

Check the tutorial on Mocking in eclass , you should test all the code included in this tutorial but don't submit it as part of your report.

Second :

Generate test cases that comply with:

- Statement coverage
- Branch coverage

And test the logic of the App.

Third:

In this task we are just learning how to use mocks (create and call) . We are not using these mocks for testing

After you run the tutorial code create Mocks inside untitest methods (untitest test cases) for each method in MathPackage where you:

- Create the method mock
- Mock a Return Value
- Use these assertions for each method :
    - assert_called
    - assert_called_once
    - assert_called_with
    - assert_called_once_with
    - call_count
    - call_args
    - call_args_list
    - method_calls
- Call the mock

In your report explain why we need mocking in Testing and comment on how effective it is.

Task 3(15):

Assume we have a system with three independent variables (A, B, and C). Each variable has three possible values (0, 1, 2).
Your task is to:

- Identify the set of test cases for this problem based on this tool, reduced if necessary
- Compare the use of orthogonal arrays to randomly generated combinations

And

- Create a valid input for the problem
- Use the tool to generate test cases
- Comment on the effectiveness of this type of tool in test case generation. Compare the use of orthogonal arrays to the **allpairspy** estimation tool. There is no application under test for this component, this is a conceptual exercise

Include the generated test cases in your lab report as well as your observations on this method. As there is no application to test, test cases need not be executed. Comment on the effectiveness of using these types of tools to generate pairwise test cases, and on the effectiveness of pairwise testing in general. What types of errors are caught by pairwise testing, what types are missed? Are the weaknesses of the method justified by the test case efficiency increase when compared to exhaustive testing? How many test cases would be required to cover all combinations of input values in the example application?

Lab report:
Must be typed, no handwritten versions will be accepted. Follow the general format as included in the lab guideline. Submit all codes. Your report should include:

- Your write-up and any required diagrams
- The results of your test cases in table format
- The HTML generated coverage reports
- Any conclusions you have drawn from these test cases regarding the applications under test.
- Your test case code as Python files to be easily executed against the provided source for validation.

Clearly indicate all failed test cases and explain the cause of these failures.