**COSC 3P95- Software Analysis & Testing**

**Assignment 1**

**Due date**: Monday, Oct 16th, 2023, at **23:59** (11:59 pm)

**Delivery method:** This is an individual assignment. Each student should submit one PDF through Brightspace.

**Attention:** This assignment is worth 10% of the course grade. Please also check the Late Assignment Policy.

**Name: \_\_\_\_\_\_\_Yifan Zhu\_\_\_\_\_\_ Student ID: \_\_\_\_\_7345671\_\_\_\_\_\_\_\_\_\_\_**

**Questions:**

1. Explain the difference between "sound" and "complete" analysis in software analysis. Then, define what true positive, true negative, false positive, and false negative mean. How would these terms change if the goal of the analysis changes, particularly when "positive" means finding a bug, and then when "positive" means not finding a bug. **(10 pts)**

A sound analysis means all programs can pass the analysis while a complete analysis means the analysis covers all cases in debugging.

**True positive (TP)** means the analysis/test result is true while the actual result is also true.

**False positive (FP)** means the analysis/test result is true while the actual result is false.

**True negative (TN)** means the analysis/test result is false while the actual result is true.

**False negative (FN)** means the analysis/test result is false while the actual result is also false.

When positive means finding a bug:

TP: The analysis finds a bug and it is actually a bug of the program.

FP: The analysis finds a bug but it’s actually not a bug in the program.

TN: The analysis doesn’t find a bug while there is a bug in the program.

FN: The analysis doesn’t find a bug while there is no bug in the program.

When positive means not finding a bug:

TP: The analysis doesn’t find a bug and there is no bug in the program.

FP: The analysis doesn’t find a bug but there is a bug in the program.

TN: The analysis finds a bug and it is a bug in the program.

FN: The analysis finds a bug while there is no bug in the program.

1. Using your preferred programming language, implement a random test case generator for a sorting algorithm program that sorts integers in ascending order. The test case generator should be designed to produce arrays of integers with random lengths, and values for each sorting method.

A) Your submission should consist of:

* 1. Source code files for the sorting algorithm and the random test case generator.
  2. Explanation of how your method/approach works and a discussion of the results (for example, if and how the method was able to generate or find any bugs, etc.). You can also include bugs in your code and show your method is able to find the input values causing that.
  3. Comments within the code for better understanding of the code.
  4. Instructions for compiling and running your code.
  5. Logs generated by the print statements, capturing both input array, output arrays for each run of the program.
  6. Logs for the random test executions, showing if the test was a pass and the output of the execution (e.g., exception, bug message, etc.).

B) Provide a context-free grammar to generate all the possible test-cases. **(18 + 8 = 26 pts)**

**A)** See **Question2.py** in the submission.

**b)**: the generator keeps generating arrays with random length and random values and use these arrays as the input of the sorting algorithm. After sorting, the test function will check whether the result is correct or not. In the example, the sorting algorithm is quicksort, if we modify anything in the sorting algorithm therefore cause a bug. The program will report an error and show the input array and output array. If there is a runtime error, the program will catch it and print the input array.

**B)**

test\_case := events\*

event := sort(arrays\*) | …

arrays := [] | [element, element] | [element, element, element] | …

element := Integer.min\_value | … | 0 | … | Integer.max\_value

1. A) For the following code, manually draw a control flow graph to represent its logic and structure.

*def filterData(data, limit, exceptions):*

*filtered\_data = []*

*index = 0*

*while index < len(data):*

*item = data[index]*

*if item in exceptions:*

*modified\_item = item + "\_EXCEPTION*

*elif item > limit:*

*modified\_item = item \* 2*

*else:*

*modified\_item = item / limit*

*filtered\_data.append(modified\_item)*

*index += 1*

*return filtered\_data*

The code is supposed to perform the followings:

* 1. If an item is in the exceptions list, the function appends "\_EXCEPTION" to the item.
  2. If an item is greater than a given limit, the function doubles the item.
  3. Otherwise, the function divides the item by 2.

B) Explain and provide detailed steps for “random testing” the above code. No need to run any code, just present the coding strategy or describe your testing method in detail. **(8 + 8 = 16 pts)**

A diagram of a flowchart

Description automatically generated

**B)** There are three input parameters for this program. A list data, a value limit and another list exceptions. First, we generate random inputs including random lists as data, random values of limit and random lists of exceptions. Then we test the function filterData() with permutations of random inputs above and record inputs and the results. In the end, we write another function for validation. If any result is not correct, we report a bug with our random inputs and result we got.

1. A) Develop 4 distinct test cases to test the above code, with code coverage ranging from 30% to 100%. For each test-case calculate and mention its code coverage.

Test case1: filterData([], 0, [])

Function coverage: 100%

Statement coverage: 30%

Branch coverage: 0%

Test case2: filterData([1,2,3,4,5,6,7], 5, [1])

Function coverage: 100%

Statement coverage: 100%

Branch coverage: 100%

Test case3: filterData([1,2,3], 4, [])

Function coverage: 100%

Statement coverage: 83.3%

Branch coverage: 33.3%

Test case4: filterData([1,2,3,4,5,7,6], 100, [1,2,4])

Function coverage: 100%

Statement coverage: 91.7%

Branch coverage: 66.7%

B) Generate 6 modified (mutated) versions of the above code.

Version1:

*def filterData(data, limit, exceptions):*

*filtered\_data = []*

*index = 0*

*while index < len(data):*

*item = data[index]*

*if item in exceptions:*

*modified\_item = item + "\_EXCEPTION*

*elif item > limit:*

*modified\_item = item \* 3*

*else:*

*modified\_item = item / limit*

*filtered\_data.append(modified\_item)*

*index += 1*

*return filtered\_data*

Version2:

*def filterData(data, limit, exceptions):*

*filtered\_data = []*

*index = 0*

*while index > len(data):*

*item = data[index]*

*if item in exceptions:*

*modified\_item = item + "\_EXCEPTION*

*elif item > limit:*

*modified\_item = item \* 2*

*else:*

*modified\_item = item / limit*

*filtered\_data.append(modified\_item)*

*index += 1*

*return filtered\_data*

Version3:

*def filterData(data, limit, exceptions):*

*filtered\_data = []*

*index = 0*

*while index < len(data):*

*item = data[index]*

*if item in exceptions:*

*modified\_item = item + "\_EXCEPTION*

*elif item > limit:*

*modified\_item = item \* 2*

*else:*

*modified\_item = item*

*filtered\_data.append(modified\_item)*

*index += 1*

*return filtered\_data*

Version4:

*def filterData(data, limit, exceptions):*

*filtered\_data = []*

*index = 0*

*while index < len(data):*

*item = data[index]*

*if item in exceptions:*

*modified\_item = item*

*elif item > limit:*

*modified\_item = item \* 2*

*else:*

*modified\_item = item / limit*

*filtered\_data.append(modified\_item)*

*index += 1*

*return filtered\_data*

Version5:

*def filterData(data, limit, exceptions):*

*filtered\_data = []*

*index = 0*

*while index < len(data):*

*item = data[index]*

*if item not in exceptions:*

*modified\_item = item + "\_EXCEPTION*

*elif item > limit:*

*modified\_item = item \* 2*

*else:*

*modified\_item = item / limit*

*filtered\_data.append(modified\_item)*

*index += 1*

*return filtered\_data*

Version6:

*def filterData(data, limit, exceptions):*

*filtered\_data = []*

*index = 0*

*while index < len(data):*

*item = data[index]*

*if item in exceptions:*

*modified\_item = item + "\_EXCEPTION*

*elif item > limit:*

*modified\_item = item \* 2*

*else:*

*modified\_item = item / limit*

*filtered\_data.append(modified\_item)*

*index += 2*

*return filtered\_data*

C) Assess the effectiveness of the test cases from part A by using mutation analysis in conjunction with the mutated codes from part B. Rank the test-cases and explain your answer.

The mutation score of the test cases from partA:

1: 0/6

2: 6/6

3: 4/6

4: 5/6

Rank: 2>4>3>1

Reason: the rank is not only given by the mutation score but also according to the statement coverage rate. Test case 2 has the highest statement coverage rate and the highest mutation score. Test case 4 has the second-highest analysis rate and so on. Therefore the rank of test cases is 2>4>3>1.

D) Discuss how you would use path, branch, and statement static analysis to evaluate/analyse the above code. **(4 \* 8 = 32 pts)**

**Path static analysis**: There are three possible paths for the program above: the item is in exceptions, the item is not in exceptions but is larger than the limit and the item in not in exceptions and is smaller than the limit or equal to the limit. To do the path static analysis, first we need to check are all the paths reachable. Then we check if there are any variables not initialized before using and initialized but not being used. In the end, we check if there is an infinite loop.

**Branch static analysis**: There are one loop and three if-else branches. To do the branch static analysis, we first check if there is any unreachable branch. Then we check the variable domain. Is there any variable being modified in the local domain by accident.

**Statement static analysis**: To do statement static analysis, we need to check spellings of variables and functions. Whenever we have a statement that calls a function, we check the format of this function (parameters).

1. The code snippet below aims to switch uppercase characters to their lowercase counterparts and vice versa. Numeric characters are supposed to remain unchanged. The function contains at least one known bug that results in incorrect output for specific inputs.

*def processString(input\_str):*

*output\_str = ""*

*for char in input\_str:*

*if char.isupper():*

*output\_str += char.lower()*

*elif char.isnumeric():*

*output\_str += char \* 2*

*else:*

*output\_str += char.upper()*

*return output\_str*

In this assignment, your tasks are:

1. Identify the bug(s) in the code. You can either manually review the code (a form of static analysis) or run it with diverse input values (a form of manual random testing). If you are unable to pinpoint the bug using these methods, you may utilize a random testing tool or implement random test case generator in code. Provide a detailed explanation of the bug, identify the line of code causing it, and describe your strategy for finding it.
2. Implement Delta Debugging, in your preferred programming language to minimize the input string that reveals the bug. Test your Delta Debugging code for the following input values provided.
   * 1. “abcdefG1”
     2. “CCDDEExy”
     3. “1234567b”
     4. “8665”

Briefly explain your delta-debugging algorithm and its implementation and provide the source code in/with your assignment. **(4 + 12 = 16 pts)**

1. The bug is at the line 8:  *output\_str += char \* 2*

According to the requirement, we need to keep numeric characters unchanged. However, with the input value “123”, the output value is “112233”, which is not expected. For more details see **Question5.py** in the submission.

1. The codes are in the file **Question5.py**. Since we already know what’s the bug, in the delta-debugging, we checked whether the length of input and length of output are the same to identify the bug. We use a binary search with recursion.
2. Extra Credit Assignment: Create a GitHub repository to host all the elements of this assignment. This includes source codes, test data, and any screenshots or logs you have generated. Submit the GitHub link along with your main submission through Brightspace. **(5 pts)**

**Link of GitHub: https://github.com/zyfeleven/3P95-A1**

Marking Scheme:

*Marks will be awarded for completeness and demonstration of understanding of the material. It is important that you fully show your knowledge when providing solutions in a concise manner. Quality and conciseness of solutions are considered when awarding marks. Lack of clarity may lead you to lose marks, so keep it simple and clear.*

***Submission:***

*The submission is expected to contain a sole word-processed document. The document can be in either* ***DOC or PDF*** *format; it should be a single column, at least single-spaced, and at least in font 11. It is strongly recommended to use the assignment questions to facilitate marking: answer the questions just below them for easier future reference.*

***Late Assignment Policy:***

*A one-time penalty of 25% will be applied on late assignments. Late assignments are accepted until the Late Assignment Date, four days after the Assignment Due Date. No excuses are accepted for missing deadlines. However, deadline extensions may be granted under extenuating circumstances, such as medical or physical conditions; please note that granting the extension is under the instructor’s discretion. However, deadline extensions may be granted under extenuating circumstances, such as medical or physical conditions; please note that granting the extension is under the instructor’s discretion.*

***Plagiarism:***

*Students are expected to respect academic integrity and deliver evaluation materials that are only produced by themselves. Any copy of content, text or code, from other students, books, web, or any other source is not tolerated. If there is any indication that an activity contains any part copied from any source, a case will be open and brought to a plagiarism committee’s attention. In case plagiarism is determined, the activity will be canceled, and the author(s) will be subject to university regulations. For further information on this sensitive subject, please refer to the document below:* ***https://brocku.ca/node/10909***