MP4: Debugging

1 Introduction

When some software tests fail, developers need to start the debugging process to remove the bug(s) from software systems. To speed up the debugging process, researchers have proposed various automated debugging techniques. In this task, you will look into a simplistic Spectrum-Based Fault Localization (SBFL) technique (also called Coveraged-based Fault Localization), Ochiai (http://repositorium.sdum.uminho.pt/bitstream/1822/37990/1/1750.pdf).

The basic idea of SBFL is very simple – statements that are primarily covered by failed tests shall be more suspicious (i.e., more likely to be buggy) than statements primarily covered by passed tests. In this way, we can easily rank all the statements in the descending order of their suspisiousness values so that the developers can start with more suspicious statements for debugging.

In this MP, you are going to compute the suspiciousness value for each statement s of Jsoup (for a given buggy version) based on the following Ochiai formula:

$$susp(s) = \frac{fail(s)}{\sqrt{totalfail * (fail(s) + pass(s))}}$$

.

Which follows the following notations:

- *fail(s)*: the number of failed tests that cover s
- *pass(s)*: the number of passed tests that cover s
- *totalfail*: the number of all failed tests (regardless of covering s or not)
- *totalpass*: the number of all passed tests (regardless of covering s or not)

You have been provided with a template Maven-based project for this MP (https://github.com/uiuc-cs427-f23/cs427-hw4), you only need to modify class `edu.illinois.cs.debugging.SBFL` to add your implementation to the locations with the TODO comments. Please do not change/add other files (including the pom.xml build file). Note that you do not need anything from MP1 and other earlier MPs to complete this MP. In case you want to access the source code of the Jsoup version to better understand the Jsoup library, you can follow MP1 to clone the Jsoup code with `git clone https://github.com/uiuc-cs427-f23/jsoup`.

To help you understand the debugging context and test your implementations, we have provided five tests (in `SBFLTest.java`) based on five real-world bugs introduced by Jsoup developers in the history. Note that all the required information has already been provided in each test for computing SBFL with respect to its corresponding bug, including the statement coverage information, failed test list, and buggy line. Also note that to simplify this MP, we assume that all three buggy versions share the same coverage file, while in practice different bugs may incur different minor changes in coverage.

There are 658 tests with coverage information, your total number of tests will be 658. For each bug, the number of passed tests will be 658 minus the size of the failed test list. For example, for the first bug, the failed test list includes only 1 test, thus the number of passed tests will be 658-1=657. That said, all you need to do is to change `SBFL.java` to pass all the three provided tests (which will fail before your implementation).

2 Instructions

- Understand method `readXMLCov` used to parse the coverage file collected for you (in XML) **using Jsoup**. Yes, you can use Jsoup to help debug itself! :) Note that you can find how to access the example coverage file in the JUnit tests. The coverage file (`src/test/resources/debug-info/cov.xml`) includes the covered lines for each test. Each test is represented by <test>, which includes the test name (represented by <name>) and the lines covered by the test separated via ", " (represented by <coveringLines>).
- Implement method `Ochiai` to compute the suspiciousness of each covered statement based on the Ochiai technique.
- Implement another two simple methods `getSusp` and `getRank` to return the suspiciousness value and rank of a buggy line. All the covered statements shall be ranked in the descending order of their suspiciousness values (so that developers can go through the list and start with inspecting more suspicious ones). Note that for the tied statements with same suspiciousness values, your `getRank` method should return the *lowest possible* rank. For example, if three statements both have the same highest values, they should all be ranked as 3rd.
- After your implementation, make sure that all three tests pass (via `mvn test`).

3 Deliverables

You are expected to upload a zip file including only the `src/main/java/edu/illinois/cs/softeng/SBFL.java` file you completed (no folders please). The name of the zip file should be your NetID. Please make sure that you do not change any other places.

Warning: you may lose all your points if you violate the following rules:

- Please make sure that your zip file is named with your NetID and only includes the specified file: **DO NOT include any folders in the zip file**, and **DO NOT change anything (including names) of the file except the TODO parts**.
- Please **DO NOT use any absolute paths** in your solution since your absolute paths will not match our grading machines. Also, please **only use "/"** as the file separator in your relative paths (if needed for this MP) since Java will usually make the correct conversions if you use the Unix file separator "/" in relative paths (for both Windows and Unix-style systems).
- Please **DO NOT fork any assignment repo** from our GitHub organization, `uiuc-cs427-f23`, or share your solution online.

4 Grading rubric

The autograder will run tests on your submitted code and your final score for this MP will be based on the success rate of test execution (including the 5 provided tests plus 5 hidden tests). The overall score is 5pt:

- Each public test has a corresponding hidden test to detect hard-coded solutions. For each of the five test pairs, you will get 1 pt if you pass both tests; otherwise, you will get 0 pt if you fail any test of the pair.

5 Addendum

In case you are interested in playing with the Jsoup project to reproduce the three bugs (captured by the three tests), here are some detailed instructions (you do not need to read the following contents to finish the MP):

- The first bug can be injected in line 432 of file `org/jsoup/parser/HtmlTreeBuilder.java`:

  - The change to reproduce the bug:

```
---} else if (("td".equals(name) || "th".equals(name) && !last)) {
+++} else if (("td".equals(name) || "td".equals(name) && !last)) {
```

  - Failed tests (1 in total, stored in `src/test/resources/debug-info/failedtests1.txt`):
    * org.jsoup.parser.HtmlParserTest.testReinsertionModeForThCelss

- The second bug can be injected in line 768 of file `org/jsoup/parser/HtmlTreeBuilderState.java`:
  - The change to reproduce the bug:

```
---String name = t.asEndTag().normalName;
+++String name = t.asEndTag().name();
```

  - Failed tests (2 in total, stored in `src/test/resources/debug-info/failedtests2.txt`):
    * org.jsoup.parser.HtmlParserTest.caseSensitiveParseTree
    * org.jsoup.parser.HtmlParserTest.caseInsensitiveParseTree

- The third bug can be injected in line 219 of file `org/jsoup/parser/TokeniserState.java`:
  - The change to reproduce the bug:

```
---} return;
+++} r.advance(); return;
```

  - Failed tests (24 in total, stored in `src/test/resources/debug-info/failedtests3.txt`):
    * org.jsoup.helper.W3CDomTest.simpleConversion
    * org.jsoup.helper.DataUtilTest.discardsSpuriousByteOrderMark
    * ...