**CS731 SOFTWARE TESTING**

**PROJECT REPORT**

**MUTATION TESTING ON SOURCE CODE OF STANDARD ALGORITHMS**

**SUBMITTED BY:**

**IMT2019004 ADRIJ SHARMA**

**IMT2019010 ANKIT AGRAWAL**

*Project Aim*

The aim of the project is to use **Mutation Testing** to test a real-world software project with the help of open-source tools.

*Code Tested*

Algorithms form an important aspect of any program and are used everywhere to achieve specific tasks. For example, the TCP/IP protocol is used to reliably transfer data between computers over the Internet. They are also used in complex systems such as rocket launches. Given the importance of algorithms, it becomes important to ensure that they are correct and work exactly as intended, while producing outputs that are correct. Hence, we chose to perform mutation testing on algorithms. We used the popular GitHub repository [TheAlgorithms](https://github.com/TheAlgorithms) to test the code. To be more specific, we used the algorithms implemented in Java ([GitHub repository link](https://github.com/TheAlgorithms/Java)) to perform the testing.

A complete list of algorithms that are implemented in Java language (in the above GitHub repository) can be found [here](https://github.com/TheAlgorithms/Java/blob/bdfecbe1c1b80761c2a061a3304ba873095131d0/DIRECTORY.md). The long list of the algorithms implemented show the sheer amount of code that is present in this repository.

*Testing Strategy and Tools Used*

Mutation testing is a form of testing where small modifications are made to the source code (and each modification is called a mutant). The aim of mutation testing is to “kill” these mutants – that is show that making small changes to the code can alter the execution of the program and hence the result that it produces. There are 2 ways to “kill” a mutant:

1. Kill a mutant **weakly**: Here, the memory state of the program after the execution of the mutated statement is different from the memory state of the program when the statement was not mutated and executed. Notice that in this case, the output of the program on a test case can remain the same, irrespective of whether a program statement was mutated or not.
2. Kill a mutant **strongly**: Here, the output of the program on a test case, when a statement was mutated and not mutated, must change. Notice that when we kill a mutant strongly, the error propagates through the program and we notice this by seeing different outputs in the presence and absence of the mutant.

We chose **mutation testing** as our testing strategy, with the aim to kill the mutants **strongly**.

The tools that we used for mutation testing are as follows:

1. [Eclipse IDE for Java](https://www.eclipse.org/downloads/packages/): A fantastic Java IDE, which allows adding plugins to add extra features to the IDE.
2. [PIT Mutation Testing Tool](https://pitest.org/): An easy-to-use mutation testing tool, that works for Java. We used the [PITclipse](https://marketplace.eclipse.org/content/pitclipse) plugin for Eclipse to integrate this tool into the Eclipse IDE.

*Mutations Used*

PIT, by default provides a set of mutation operators. These operators are listed below:

|  |  |
| --- | --- |
| BOOLEAN\_FALSE\_RETURN | INCREMENTS\_MUTATOR |
| BOOLEAN\_TRUE\_RETURN | INVERT\_NEGS\_MUTATOR |
| CONDITIONALS\_BOUNDARY\_MUTATOR | MATH\_MUTATOR |
| EMPTY\_RETURN\_VALUES | NEGATE\_CONDITIONALS\_MUTATOR |
| PRIMITIVE\_RETURN\_VALS\_MUTATOR | VOID\_METHOD\_CALL\_MUTATOR |
| NULL\_RETURN\_VALUES | |

More information about the mutation operators present in PITest can be found [here](https://pitest.org/quickstart/mutators/).

*Tabulated Testing Results*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm  Domain** | **Algorithm Name** | **Mutations  Killed** | **%  Mutations  Killed** | **Lines of Code  in Algorithm  (approx.) \*\*** |
| Dynamic  Programming | [Coin Change](https://github.com/TheAlgorithms/Java/blob/master/src/main/java/com/thealgorithms/dynamicprogramming/CoinChange.java) | 22/24 | 92 | 40 |
| [Edit Distance](https://github.com/TheAlgorithms/Java/blob/master/src/main/java/com/thealgorithms/dynamicprogramming/EditDistance.java) | 26/29 | 90 | 45 |
| [Fibonacci Numbers](https://github.com/TheAlgorithms/Java/blob/master/src/main/java/com/thealgorithms/dynamicprogramming/Fibonacci.java) | 27/30 | 90 | 50 |
| [0-1 Knapsack](https://github.com/TheAlgorithms/Java/blob/master/src/main/java/com/thealgorithms/dynamicprogramming/Knapsack.java) | 23/23 | 100 | 27 |
| [Longest Common Subsequence](https://github.com/TheAlgorithms/Java/blob/master/src/main/java/com/thealgorithms/dynamicprogramming/LongestCommonSubsequence.java) | 48/53 | 91 | 60 |
| [Longest Increasing Subsequence](https://github.com/TheAlgorithms/Java/blob/master/src/main/java/com/thealgorithms/dynamicprogramming/LongestIncreasingSubsequence.java) | 37/41 | 90 | 75 |
| [Longest Palindromic Subsequence](https://github.com/TheAlgorithms/Java/blob/master/src/main/java/com/thealgorithms/dynamicprogramming/LongestPalindromicSubsequence.java) | 14/14 | 100 | 45 |
| [Longest Palindromic Substring](https://github.com/TheAlgorithms/Java/blob/master/src/main/java/com/thealgorithms/dynamicprogramming/LongestPalindromicSubstring.java) | 19/20 | 95 | 55 |
| [Longest Valid Parentheses](https://github.com/TheAlgorithms/Java/blob/master/src/main/java/com/thealgorithms/dynamicprogramming/LongestValidParentheses.java) | 26/27 | 96 | 35 |
| [Minimum Sum Partitioning](https://github.com/TheAlgorithms/Java/blob/master/src/main/java/com/thealgorithms/dynamicprogramming/MinimumSumPartition.java) | 36/39 | 92 | 50 |
| [Palindromic Partitioning](https://github.com/TheAlgorithms/Java/blob/master/src/main/java/com/thealgorithms/dynamicprogramming/PalindromicPartitioning.java) | 33/34 | 97 | 50 |
| [Rod Cutting](https://github.com/TheAlgorithms/Java/blob/master/src/main/java/com/thealgorithms/dynamicprogramming/RodCutting.java) | 11/11 | 100 | 15 |
| [Subset Counting](https://github.com/TheAlgorithms/Java/blob/master/src/main/java/com/thealgorithms/dynamicprogramming/SubsetCount.java) | 36/36 | 100 | 35 |
| [Subset Sum](https://github.com/TheAlgorithms/Java/blob/master/src/main/java/com/thealgorithms/dynamicprogramming/SubsetSum.java) | 26/30 | 87 | 20 |
| [Unique Paths](https://github.com/TheAlgorithms/Java/blob/master/src/main/java/com/thealgorithms/dynamicprogramming/UniquePaths.java) | 29/29 | 100 | 30 |
| Divide &  Conquer | [Binary Search](https://www.geeksforgeeks.org/binary-search/) | 11/12 | 92 | 20 |
| [Closest Pair](https://www.geeksforgeeks.org/closest-pair-of-points-using-divide-and-conquer-algorithm/) | 98/109 | 90 | 215 |
| [Rabin Karp](https://www.geeksforgeeks.org/rabin-karp-algorithm-for-pattern-searching/) | 38/41 | 93 | 45 |
| [Strassen Matrix Multiplication](https://www.geeksforgeeks.org/strassens-matrix-multiplication/) | 67/75 | 89 | 85 |
| [Binary Search in 2D Array](https://www.geeksforgeeks.org/search-element-sorted-matrix/) | 38/52 | 73 | 50 |
| Trees | [Ceil in Binary Search Tree](https://www.geeksforgeeks.org/floor-and-ceil-from-a-bst/) | 17/19 | 90 | 60 |
| [Check Symmetric Tree](https://www.geeksforgeeks.org/symmetric-tree-tree-which-is-mirror-image-of-itself/) | 25/26 | 96 | 30 |
| [K-dimensional Tree](https://www.geeksforgeeks.org/k-dimensional-tree/) | 60/108 | 58 | 270 |
| [Segment Tree](https://www.geeksforgeeks.org/lazy-propagation-in-segment-tree/) | 50/55 | 91 | 85 |
| String  Algorithms | [Anagram Checker](https://www.geeksforgeeks.org/check-whether-two-strings-are-anagram-of-each-other/) | 37/37 | 100 | 80 |
| [Letter Combinations of  Phone Number](https://www.geeksforgeeks.org/iterative-letter-combinations-of-a-phone-number/) | 6/6 | 100 | 30 |
| [String to Number - Atoi](https://www.geeksforgeeks.org/java-program-to-write-your-own-atoi/) | 30/34 | 88 | 70 |
| [Palindrome](https://www.geeksforgeeks.org/c-program-check-given-string-palindrome/) | 25/28 | 89 | 25 |
| Graph | [Hamiltonian Cycle](https://www.geeksforgeeks.org/hamiltonian-cycle-backtracking-6/) | 28/30 | 93 | 60 |
| [Bipartite](https://www.geeksforgeeks.org/bipartite-graph/) | 16/17 | 94 | 20 |

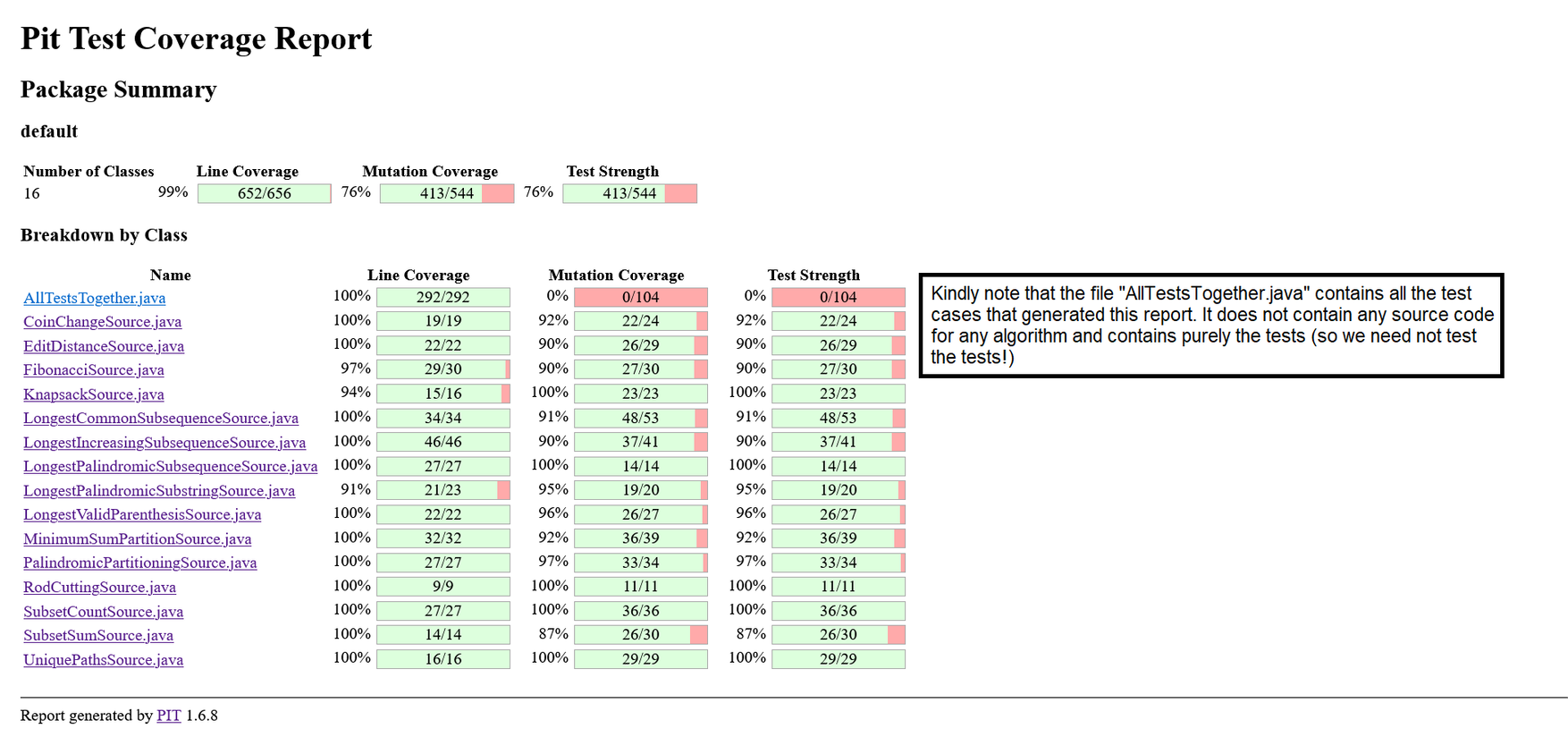
We are able to achieve at least 90% mutation kills in almost all of the functions

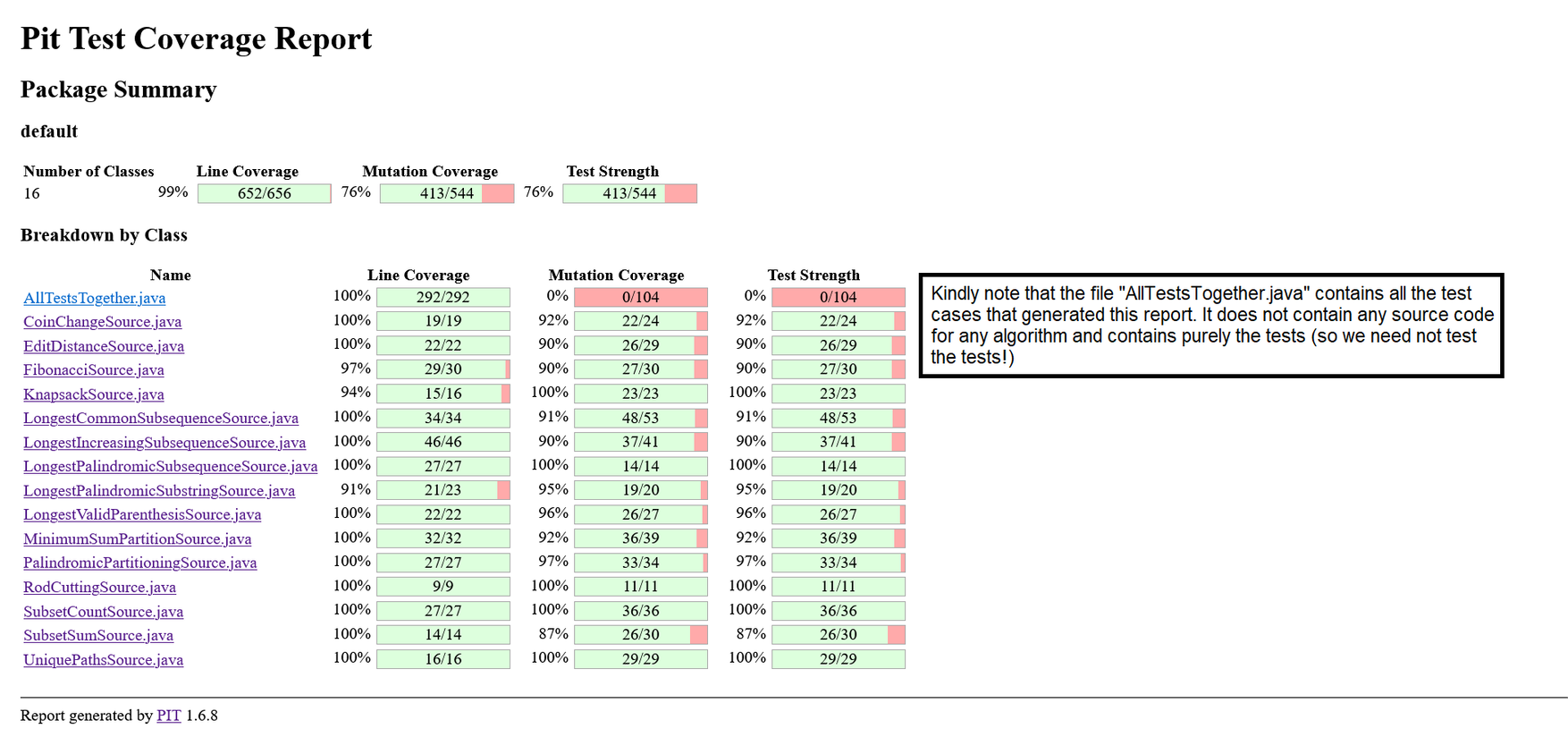
\*DP stands for Dynamic Programming.

\*\* Lines of Code in Algorithm include little/no documentation.

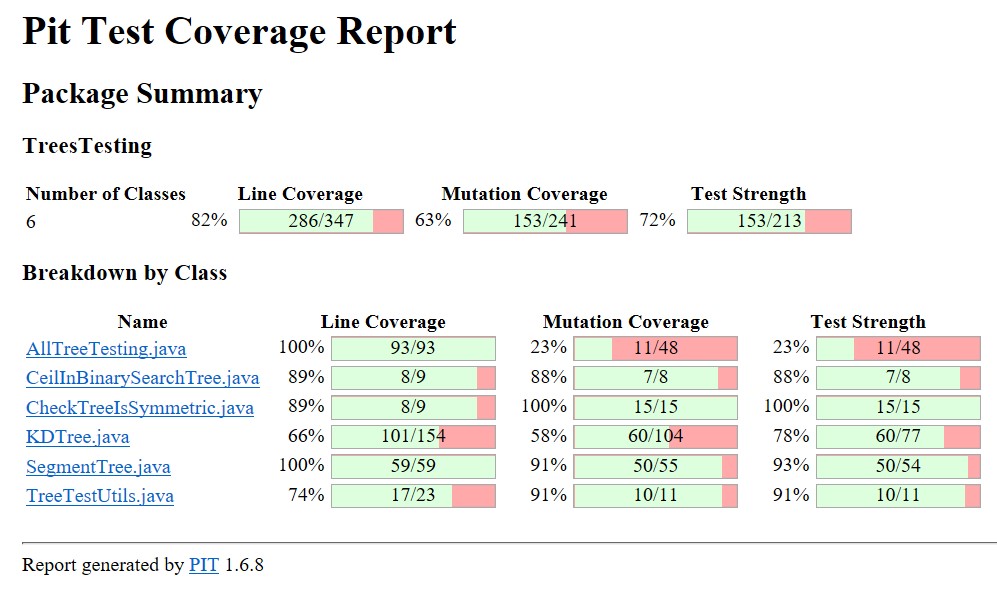
*Testing Summaries*

1. *Dynamic Programming*

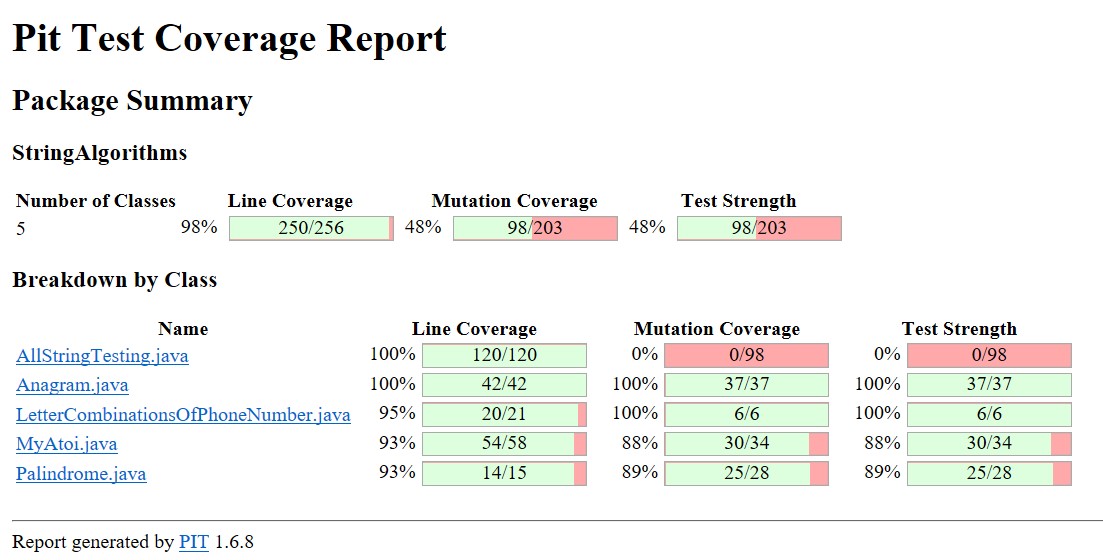




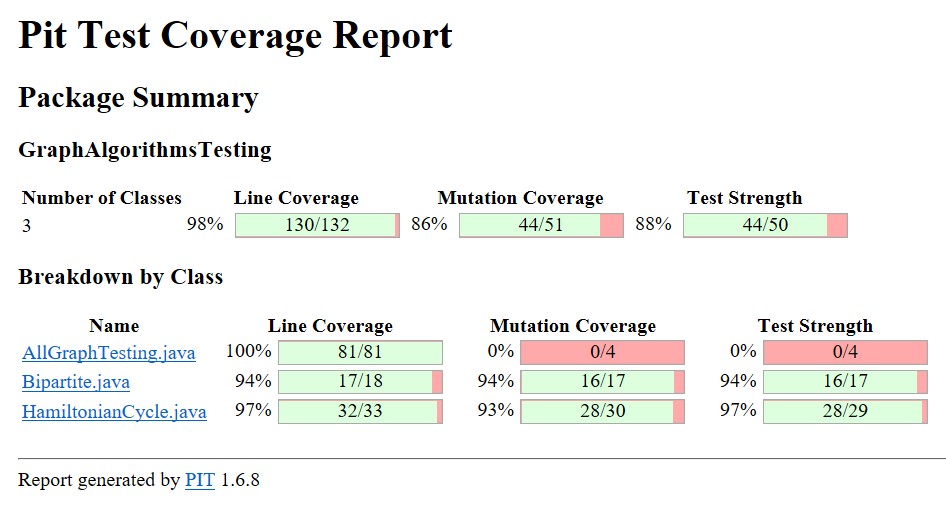
1. *Trees*



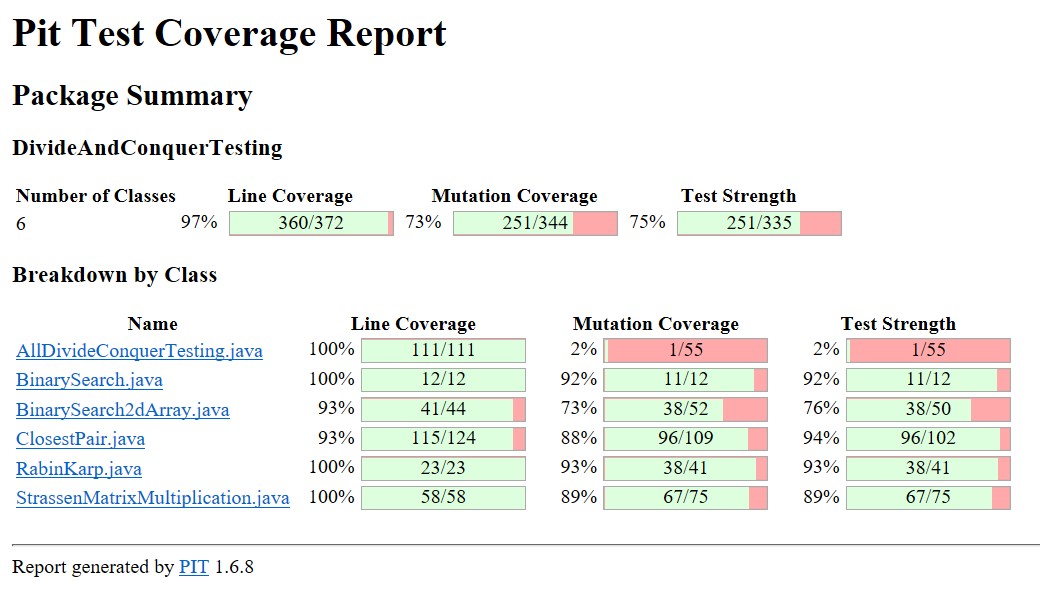
1. *Strings*



1. *Graphs*



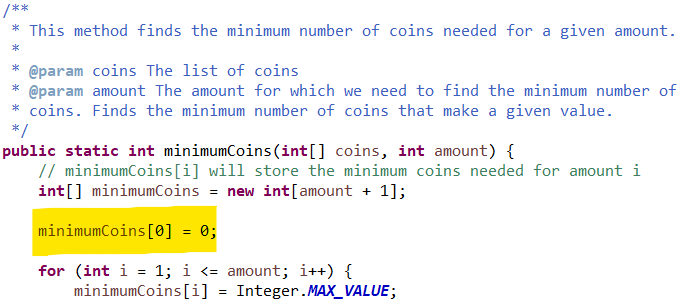
1. *Divide & Conquer*



*Bugs Found*

Because the code in the repository comes from a diverse group of coders, the presence of bugs in the implementation of algorithms is obvious. Below are some interesting highlights that we found out while testing the code:

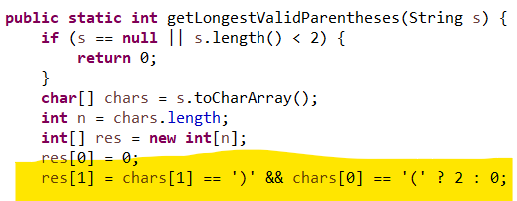
***Bug 1:* Handling Edge Cases Incorrectly**



Out-Of-Bounds Access Possible (Program: Coin Change)

In the above code, we found that the above function assumes the presence of at least one coin. In the real world, this may be true, but during testing, an out-of-bounds memory access can lead to segmentation faults.

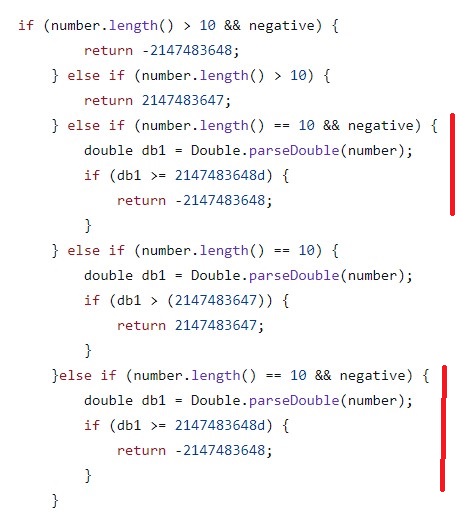
***Bug 2:* No Input Sanity Check**



No Input Sanity Check (Program: Longest Valid Parentheses)

The function finding longest valid parentheses assumes that the input string consists of only characters ‘(‘ and ‘)’. However, an unaware user could also use square brackets (which are ‘[‘ and ‘]’) and curly brackets (which are ‘{‘ and ‘}’). Hence, the function should also make sure that its input string consists solely of ‘(‘ and ‘)’.

***Bug 3:*** Presence of Extra redundant code which reduces the % of mutants killed



The above image has the highlighted duplicate pieces of code in the string function myAtoi().

*References*

1. Mutation Testing Theory:
   1. <https://www.geeksforgeeks.org/software-testing-mutation-testing/>
   2. <https://www.guru99.com/mutation-testing.html>
2. Mutation Testing Tutorial:
   1. <https://youtu.be/wZeZMtqVmck>
   2. <https://youtu.be/lDeTsMIN8As>
3. PITclipse: <https://github.com/pitest/pitclipse>
4. JUnit Assert: <https://junit.org/junit4/javadoc/4.13/org/junit/Assert.html>
5. PITest: <https://medium.com/geekculture/mutation-testing-for-maven-project-using-pitest-f9b8fef03a05>