Mutation Testing

Raghunadh (IMT2021042)
Kushal (IMT2021035)

November 26, 2024

Abstract

Mutation testing evaluates the quality of a test suite by introducing small changes (mutants) to the code and checking if the tests can detect them. In this project, PIT (Pitest), a leading mutation testing tool for Java, was used. PIT generates mutants using various operators and runs the existing test cases against them to identify "survived mutants," indicating untested code. This approach revealed weaknesses in our test suite, highlighting areas requiring improved coverage.

1 Introduction

Testing is vital in ensuring software quality, but traditional metrics like code coverage may not fully measure the effectiveness of test suites. Mutation testing provides a deeper evaluation by introducing intentional faults (mutants) and determining if test cases can detect them. This report details the application of mutation testing using PIT (Pitest) on Java source files. PIT generates mutants through various operators, simulating real-world coding errors. The analysis identified gaps in the test suite, highlighting areas for improvement. By addressing these gaps, the project enhanced the test suite's robustness and improved the overall reliability of the codebase.

2 Project goal

The goal of the project is to implement mutation testing using PIT and evaluate strength and coverage of test suite along with using different mutation operators on the test results.

3 Methodology

3.1 Tools and Technologies Used

The following tools and technologies were used in the project:

- **Programming Language**: Java

- **Build Tool**: Maven
- **Mutation Testing Tool**: PIT
- **Testing Framework**: JUnit

3.2 Project Setup

Make sure that `pom.xml` file was configured to integrate PIT with specified dependencies.

- Junit dependency

```
 <!-- JUnit Dependency -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
</dependency>
```

- Pit dependency

```
<dependency>
    <groupId>org.pitest</groupId>
    <artifactId>pitest</artifactId>
    <version>1.9.11</version> <!-- Use the latest version available -->
    <scope>test</scope>
</dependency>
```

- Pit plugins and target classes

```
<groupId>org.pitest</groupId>
<artifactId>pitest-maven</artifactId>
<version>1.9.11</version>
<executions>
    <execution>
        <goals>
            <goal>mutationCoverage</goal>
        </goals>
    </execution>
</executions>
```

3.3 Mutation Operators

Both unit and integration-based mutators were applied to evaluate test cases. Here are some of them which we applied

Unit Test Mutators

These mutators primarily test individual units of the code, such as methods, to ensure they handle edge cases and unexpected inputs effectively:

- **CONDITIONALS_BOUNDARY:** Alters conditional boundaries, such as changing `>` to `>=`, to test edge-case behavior.
- **INLINE_CONSTS:** Modifies constant values inline, such as replacing `42` with `41`, to validate that tests detect errors in constant usage.
- **INVERT_NEGS:** Inverts negative numbers (e.g., `-5` becomes `5`) to ensure tests are sensitive to sign changes in numerical values.
- **MATH:** Alters mathematical operations (e.g., `+` to `-`) to ensure tests catch potential arithmetic errors.
- **NEGATE_CONDITIONALS:** Negates boolean conditions (e.g., `if (a > b)` to `if (a <= b)`) to verify tests handle logical inversions properly
- **REMOVE_INCREMENTS:** Removes increment operators (e.g., `++`) from the code, testing if the tests handle changes in state correctly after modifications.

Integration Mutators

These mutators focus on testing interactions between components in an integrated system, especially how changes in one component (e.g., return values, argument propagation, conditional structures) affect the overall system

- **EMPTY RETURNS:** Replaces method return values with an empty return.
- **EXPERIMENTAL ARGUMENT PROPAGATION:** Propagates arguments to different usages.
- **FALSE RETURNS:** Forces methods to return false.
- **NULL RETURNS:** Forces methods to return null.
- **PRIMITIVE RETURNS:** Forces methods to return primitive default values.
- **TRUE RETURNS:** Forces methods to return true.
- **REMOVE CONDITIONALS EQUAL ELSE:** Removes the else block when conditionals have equal branches to examine how the system behaves when conditional structures are simplified.

- **REMOVE CONDITIONALS ORDER IF:** Removes the if block when conditionals are ordered, testing how the system handles missing conditional checks between components.

3.4 Test Case Generation

Test cases are designed by using the `@Test` annotation in JUnit. This annotation marks a method as a test method, which is executed by the JUnit framework when running the tests. Here's a simple example:

```
@Test
public void testLIS()
{
    LIS lis = new LIS();
    // Test case 1
    int[] nums1 = {10, 9, 2, 5, 3, 7, 101, 18};
    assertEquals(4, lis.lengthOfLIS(nums1));
}
```

Each method annotated with `@Test` should contain code that tests a specific functionality of the application. These test methods are executed automatically by the JUnit framework when the tests are run.

4 Source code

We selected various algorithms from different topics as the source code for testing to ensure comprehensive coverage and validate their functionality. We have chosen BFS, DFS, Dijkstra, DSU from graphs. We have chosen a couple of dynamic programming problems along with binary search to cover all types of algorithms. By including a wide range of algorithms, we aimed to test diverse scenarios, evaluate their accuracy, and ensure the robustness of the implemented logic.

5 Implementation

- Modularized the source code into smaller, more manageable functions for improved readability and maintainability.
- Configured the pom.xml file to integrate PIT for mutation testing.
- Executed PIT to generate mutants and reviewed the analysis of the results.
- Enhanced the test suite to address and eliminate surviving mutants.

6 Results

The results highlight the mutation coverage and surviving mutant distribution. We show some of the results which we got from the test cases written by us below.

6.1 Summary of Mutators for SlidingPuzzle.java

```
4
5   public class SlidingPuzzle {
6
7           public int slidingPuzzle(int[][] board) {
8               // Directions for possible swaps based on '0' position
9  41           int[][] dir = {{1, 3}, {0, 2, 4}, {1, 5}, {0, 4}, {1, 3, 5}, {2, 4}};
10              String target = "123450";
11 1            Set<String> vis = new HashSet<>(); // Track visited configurations
12 1            Queue<String> q = new LinkedList<>();
13              String start = "";
14
15              // Convert 2D board to a single string
16              for (int[] row : board) {
17                  for (int col : row) {
18 6                    start += col;
19                  }
20              }
21 1            q.offer(start);
22 1            vis.add(start);
23 1            int step = 0;
24
25              // Perform BFS
26 4            while (!q.isEmpty()) {
27 1                int size = q.size();
28 6                while (size-- > 0) {
29 1                    String current = q.poll();
30
31                      // Check if target is reached
32 5                    if (current.equals(target)) return step;
33
34 3                    int zero = current.indexOf('0'); // Find position of '0'
35
36                      for (int move : dir[zero]) {
37 1                        StringBuilder next = new StringBuilder(current);
38 1                        char temp = next.charAt(zero);
39 2                        next.setCharAt(zero, next.charAt(move));
40 1                        next.setCharAt(move, temp);
41
42 5                        if (!vis.contains(next.toString())) { // Add unvisited states to the queue
43 2                            vis.add(next.toString());
44 2                            q.offer(next.toString());
45                          }
46                      }
47                  }
48 2                step++;
49              }
50 2            return -1; // Return -1 if target is unreachable
51          }
52
53  }
```

Mutations

1. Substituted 6 with 7 → SURVIVED
2. Substituted 0 with 1 → KILLED
3. Substituted 2 with 3 → SURVIVED
4. Substituted 0 with 1 → KILLED
5. Substituted 1 with 0 → KILLED
6. Substituted 1 with 0 → KILLED
7. Substituted 3 with 4 → KILLED
8. Substituted 1 with 0 → KILLED
9. Substituted 3 with 4 → SURVIVED
10. Substituted 0 with 1 → SURVIVED
11. Substituted 0 with 1 → SURVIVED
12. Substituted 1 with 0 → SURVIVED
13. Substituted 2 with 3 → KILLED
14. Substituted 2 with 3 → KILLED
15. Substituted 4 with 5 → KILLED
16. Substituted 2 with 3 → KILLED
17. Substituted 2 with 3 → KILLED
18. Substituted 0 with 1 → KILLED
19. Substituted 1 with 0 → KILLED
20. Substituted 1 with 0 → KILLED
21. Substituted 5 with 6 → KILLED
22. Substituted 3 with 4 → KILLED
23. Substituted 2 with 3 → SURVIVED
24. Substituted 0 with 1 → SURVIVED
25. Substituted 0 with 1 → KILLED
26. Substituted 1 with 0 → SURVIVED
27. Substituted 4 with 5 → KILLED
28. Substituted 4 with 5 → KILLED
29. Substituted 3 with 4 → KILLED
30. Substituted 0 with 1 → KILLED
31. Substituted 1 with 0 → KILLED
32. Substituted 1 with 0 → KILLED
33. Substituted 3 with 4 → KILLED
34. Substituted 2 with 3 → KILLED
35. Substituted 5 with 6 → KILLED
36. Substituted 5 with 6 → KILLED
37. Substituted 2 with 3 → SURVIVED
38. Substituted 0 with 1 → SURVIVED
39. Substituted 2 with 3 → KILLED
40. Substituted 1 with 0 → SURVIVED
41. Substituted 4 with 5 → SURVIVED

1. removed call to java/util/HashSet::<init> → KILLED

1. removed call to java/util/LinkedList::<init> → KILLED

1. removed call to java/lang/StringBuilder::<init> → KILLED
2. removed call to java/lang/StringBuilder::append → KILLED
3. removed call to java/lang/StringBuilder::append → KILLED
4. removed call to java/lang/StringBuilder::toString → KILLED
5. replaced call to java/lang/StringBuilder::append with receiver → KILLED
6. replaced call to java/lang/StringBuilder::append with receiver → KILLED

1. removed call to java/util/Queue::offer → KILLED

1. removed call to java/util/Set::add → SURVIVED

1. Substituted 0 with 1 → KILLED

1. negated conditional → KILLED
2. removed call to java/util/Queue::isEmpty → TIMED_OUT
3. removed conditional - replaced equality check with false → KILLED
4. removed conditional - replaced equality check with true → TIMED_OUT

1. removed call to java/util/Queue::size → TIMED_OUT

1. changed conditional boundary → KILLED
2. Changed increment from -1 to 1 → KILLED
3. negated conditional → TIMED_OUT
4. removed conditional - replaced comparison check with false → TIMED_OUT
5. removed conditional - replaced comparison check with true → KILLED
6. Removed increment -1 → KILLED

1. removed call to java/util/Queue::poll → KILLED

1. negated conditional → KILLED
2. removed call to java/lang/String::equals → KILLED
3. removed conditional - replaced equality check with false → KILLED
4. removed conditional - replaced equality check with true → KILLED
5. replaced int return with 0 for org/example/SlidingPuzzle::slidingPuzzle → KILLED

1. Substituted 48 with 49 → KILLED
2. removed call to java/lang/String::indexOf → KILLED
3. replaced call to java/lang/String::indexOf with argument → KILLED

1. removed call to java/lang/StringBuilder::<init> → KILLED

1. removed call to java/lang/StringBuilder::charAt → KILLED

1. removed call to java/lang/StringBuilder::charAt → KILLED

This section provides a detailed overview of the killed and survived mutators observed during the analysis.

Killed Mutators:

- **Substitution Mutators:**
  - Value substitution: (e.g., $0 \rightarrow 1$, $2 \rightarrow 3$, $5 \rightarrow 6$, $4 \rightarrow 5$, etc.)
  - Increment/decrement changes: (e.g., Changed increment from 1 to -1, Removed increment -1)
  - Equality checks: (e.g., replacing equality with `false` or `true`)
- **Method Call Mutators:**
  - Removed method calls:
    - `java/util/HashSet::<init>`
    - `java/util/LinkedList::<init>`
    - `java/lang/StringBuilder::<init>`
    - `java/util/Queue::offer`
    - `java/util/Queue::poll`
- **Conditional Logic Mutators:**
  - Negated conditionals.
  - Changed boundary conditions (e.g., boundary check alterations).
- **Control Flow Mutators:**
  - Replacing conditional outcomes with `false` or `true`.
  - Removing critical operations or calls that affect flow.
- **Return Value Mutators:**
  - Replaced integer return values with constant values (e.g., replacing return with 0).

Survived Mutators:

- **Substitution Mutators:**
  - $6 \rightarrow 7$, $3 \rightarrow 4$, and similar value substitutions for specific cases.
- **Set Operation Mutators:**
  - Removing the call to `java/util/Set::add`.
- **Edge Case Mutators:**
  - Certain substitutions (e.g., $0 \rightarrow 1$) survived in unique cases.

```
1    package org.example;
2
3    public class ReversePairs {
4            public int reversePairs(int[] arr) {
5  6             if (arr == null || arr.length == 0) {
6  1                 return 0;
7                 }
8  6             return mergeSortAndCount(arr, 0, arr.length - 1);
9            }
10
11           private int mergeSortAndCount(int[] arr, int left, int right) {
12 4             if (left >= right) {
13 1                 return 0;
14                }
15
16 4             int mid = left + (right - left) / 2;
17 1             int count = 0;
18
19 3             count += mergeSortAndCount(arr, left, mid);
20 5             count += mergeSortAndCount(arr, mid + 1, right);
21 3             count += mergeAndCount(arr, left, mid, right);
22
23 1             return count;
24           }
25
26           private int mergeAndCount(int[] arr, int left, int mid, int right) {
27 1             int count = 0;
28 2             int j = mid + 1;
29 4             for (int i = left; i <= mid; i++) {
30 10                while (j <= right && arr[i] > 2 * (long) arr[j]) {
31                    j++;
32                }
33 4                count += (j - (mid + 1));
34            }
35 3             int[] temp = new int[right - left + 1];
36 1             int i = left, k = 0;
37 2             j = mid + 1;
38
39 8             while (i <= mid && j <= right) {
40 4                 if (arr[i] <= arr[j]) {
41 4                     temp[k++] = arr[i++];
42                } else {
43 4                     temp[k++] = arr[j++];
44                }
45            }
46
47 4             while (i <= mid) {
48 4                 temp[k++] = arr[i++];
49            }
50
51 4             while (j <= right) {
52 4                 temp[k++] = arr[j++];
53            }
54
55 4             for (i = left; i <= right; i++) {
56 1                 arr[i] = temp[i - left];
57            }
58
59 1             return count;
60           }
61   }
```

Mutations

5	1. negated conditional → KILLED 2. negated conditional → KILLED 3. removed conditional - replaced equality check with false → KILLED 4. removed conditional - replaced equality check with false → SURVIVED 5. removed conditional - replaced equality check with true → SURVIVED 6. removed conditional - replaced equality check with true → KILLED
6	1. Substituted 0 with 1 → NO_COVERAGE
8	1. Substituted 0 with 1 → KILLED 2. Substituted 1 with 0 → KILLED 3. Replaced integer subtraction with addition → KILLED 4. removed call to org/example/ReversePairs::mergeSortAndCount → KILLED 5. replaced call to org/example/ReversePairs::mergeSortAndCount with argument → KILLED 6. replaced int return with 0 for org/example/ReversePairs::reversePairs → KILLED
12	1. changed conditional boundary → KILLED 2. negated conditional → KILLED 3. removed conditional - replaced comparison check with false → KILLED 4. removed conditional - replaced comparison check with true → KILLED
13	1. Substituted 0 with 1 → KILLED
16	1. Substituted 2 with 3 → SURVIVED 2. Replaced integer subtraction with addition → KILLED 3. Replaced integer division with multiplication → KILLED 4. Replaced integer addition with subtraction → KILLED
17	1. Substituted 0 with 1 → KILLED
19	1. Replaced integer addition with subtraction → SURVIVED 2. removed call to org/example/ReversePairs::mergeSortAndCount → KILLED 3. replaced call to org/example/ReversePairs::mergeSortAndCount with argument → KILLED
20	1. Substituted 1 with 0 → KILLED 2. Replaced integer addition with subtraction → KILLED 3. Replaced integer addition with subtraction → KILLED 4. removed call to org/example/ReversePairs::mergeSortAndCount → KILLED 5. replaced call to org/example/ReversePairs::mergeSortAndCount with argument → KILLED
21	1. Replaced integer addition with subtraction → KILLED 2. removed call to org/example/ReversePairs::mergeAndCount → KILLED 3. replaced call to org/example/ReversePairs::mergeAndCount with argument → KILLED
23	1. replaced int return with 0 for org/example/ReversePairs::mergeSortAndCount → KILLED
27	1. Substituted 0 with 1 → KILLED
28	1. Substituted 1 with 0 → KILLED 2. Replaced integer addition with subtraction → KILLED
29	1. changed conditional boundary → KILLED 2. negated conditional → KILLED 3. removed conditional - replaced comparison check with false → KILLED 4. removed conditional - replaced comparison check with true → KILLED
30	1. changed conditional boundary → KILLED 2. changed conditional boundary → KILLED 3. Substituted 2 with 3 → KILLED 4. Replaced long multiplication with division → KILLED 5. negated conditional → KILLED 6. negated conditional → KILLED 7. removed conditional - replaced comparison check with false → KILLED 8. removed conditional - replaced comparison check with false → KILLED 9. removed conditional - replaced comparison check with true → KILLED 10. removed conditional - replaced comparison check with true → KILLED
33	1. Substituted 1 with 0 → KILLED 2. Replaced integer addition with subtraction → KILLED 3. Replaced integer subtraction with addition → KILLED 4. Replaced integer addition with subtraction → KILLED
35	1. Substituted 1 with 0 → KILLED 2. Replaced integer subtraction with addition → SURVIVED 3. Replaced integer addition with subtraction → KILLED
36	1. Substituted 0 with 1 → KILLED
37	1. Substituted 1 with 0 → KILLED 2. Replaced integer addition with subtraction → KILLED
39	1. changed conditional boundary → KILLED 2. changed conditional boundary → KILLED 3. negated conditional → KILLED 4. negated conditional → KILLED 5. removed conditional - replaced comparison check with false → KILLED 6. removed conditional - replaced comparison check with false → KILLED 7. removed conditional - replaced comparison check with true → KILLED 8. removed conditional - replaced comparison check with true → KILLED
	1. changed conditional boundary → SURVIVED

6.2 Summary of Mutators for ReversePairs.java

This section provides a detailed overview of the killed and survived mutators observed during the analysis.

Killed Mutators:

- **Negated Conditional:**
  - Negating conditionals led to test failures, indicating that the conditional checks are critical for the function's correctness
- **Removed Conditional - Replaced Equality Check with False:**
  - Replacing equality checks with false caused incorrect logic and failed tests.
- **Changed Conditional Boundary:**
  - Altering boundary conditions in conditionals led to errors in logic and failed tests.
- **Removed Increment 1:**
  - Removing the increment caused the logic to break, resulting in test failures.

Survived Mutators:

Replaced Integer Addition with Subtraction:

- In certain cases, replacing addition with subtraction did not cause test failures.

Changed Increment from 1 to -1:

- Some changes in increment from 1 to -1 did not affect the outcome, allowing tests to pass.

7 Pit report

The Pit Test Coverage Report analyzes the code coverage of the Java project. This is the measured line coverage, mutation coverage, and test strength of all functionalities present in our source code. It gives the summary of line coverage, mutation coverage and test strength of the whole code.

Pit Test Coverage Report

Package Summary

org.example

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
16	97%	525/542	87%	1375/1586	88%	1375/1557

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
CutStick.java	100%	13/13	90%	38/42	90%	38/42
DestinationWays.java	100%	32/32	81%	66/81	81%	66/81
FloodFill.java	100%	26/26	89%	81/91	89%	81/91
LIS.java	100%	19/19	92%	47/51	92%	47/51
MakingLargeIsland.java	97%	57/59	76%	138/181	77%	138/180
MatchSticks.java	96%	22/23	81%	47/58	84%	47/56
MaxConnectedComponents.java	93%	65/70	88%	190/217	89%	190/213
MaximumXor.java	100%	44/44	95%	97/102	95%	97/102
MedianOfTwoSortedArrays.java	93%	26/28	82%	68/83	85%	68/80
PartitionArray.java	100%	14/14	93%	37/40	93%	37/40
RabinKarp.java	92%	22/24	90%	70/78	90%	70/78
RestrictedPaths.java	100%	35/35	85%	83/98	85%	83/98
ReversePairs.java	97%	31/32	91%	95/104	92%	95/103
SlidingPuzzle.java	100%	30/30	84%	76/90	84%	76/90
Statistics.java	92%	47/51	80%	110/137	92%	110/119
SumOfSubArrayRanges.java	100%	42/42	99%	132/133	99%	132/133

Report generated by PIT 1.9.11

8 Conclusion

This project successfully implemented mutation testing using PIT, demonstrating its effectiveness in evaluating and strengthening test cases. By applying both standard and experimental mutation operators, valuable insights were gained into the strengths and weaknesses of the test suite. The results highlighted areas for improvement, emphasizing the need for refining test cases to enhance coverage and robustness. Ultimately, this approach contributed to increasing the overall reliability and fault tolerance of the software.

9 Contribution

- **Raghunadh (IMT2021042)**:
  - Solved 8 different problems and made sure to create tests that covered all possibilities in graphs and dynamic programming.

  - Helped put together the project report, summarizing what we learned from the testing.

- **Kushal (IMT2021035)**:
  - Worked on 8 problems, creating tests to check how well they work in different situations related to binary search, graphs, and strings.
  - Also, played a part in writing the report, where we shared what we found out during the testing.

10 References

- PIT Documentation: `https://pitest.org/`
- JUnit Documentation: `https://junit.org/junit4/`
- Maven Documentation: `https://maven.apache.org`
- Source code on github: `https://github.com/Raghunadh2004/MutationTesting`