SOFTWARE TESTING PROJECT

- **Vamshi Krishna Reddy Guggulla, IMT2020080**

- **Rahul Kumar Reddy Puram, IMT2020114**

Problem Overview:

The context involves projects that employ mutation testing. In this testing paradigm, mutations are introduced at the level of individual statements within methods or functions. The primary goal is to create a set of test cases that robustly identify and neutralize these mutations. For this task, a minimum of three different mutation operators should be applied.

Objective:

The overarching aim is to ensure that the introduced mutations are effectively detected and neutralized by the test suite. This involves devising test cases that can discern and eliminate variations introduced at the statement level, thus enhancing the reliability and effectiveness of the mutation testing process.

INTRODUCTION:

A type of software testing called mutation testing is used to evaluate the efficacy of current software tests and to create new ones. With the primary objective of helping testers create effective tests and identify flaws in the test data applied to the program, this approach entails making minor syntactic changes to a program. The process comprises adding legitimate changes to a software artifact and then evaluating the modified artifact. Within the domain of software artifact grammars, which yield valid strings, mutation testing makes use of derivations in these grammars to yield strings that are both valid and invalid. These valid and invalid strings are then used in the subsequent testing procedure, which is what mutation testing is all about.

Maven:

We have used Maven for building the project,it serves as a useful assistant that assembles our project, handling the organization and ensuring everything essential for our project is appropriately arranged and we have used VScode Editor for our project.

PITest:

PITest, short for "Pipeline Test," is an open-source mutation testing tool tailored for Java projects. Unlike traditional testing, mutation testing involves introducing deliberate changes to the source code to assess how well the existing test suite detects these alterations. PITest injects mutations at the bytecode level, simulating potential faults such as altering operations or introducing new errors. Integrated with Maven and Gradle, PITest seamlessly fits into continuous integration pipelines, providing automated mutation analysis during the build process. The tool generates detailed reports, helping developers identify weaknesses in their test suites and enhance overall mutation coverage, contributing to the creation of more robust software systems.

Source Code:

Link to the source code :

```
https://github.com/Vamshikrishna5944/SoftwareTesting.git
```

In the process of assembling code samples, we curated from various publicly accessible online problem sets, including but not limited to CSES problem sets, Striver's SDE sheets, and selections from GeeksforGeeks. These problem sets encompass a diverse range of topics such as graphs, binary search, trees, arrays, sorting, and searching, ensuring comprehensive coverage of fundamental programming concepts.

RESULTS:

The outcomes obtained from our code are as follows.

Pit Test Coverage Report

Package Summary

com.myproblemset.app

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
28	99%	661/665	94%	658/703	94%	658/700

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
Allocation.java	100%	27/27	93%	25/27	93%	25/27
Appartment.java	100%	20/20	92%	12/13	92%	12/13
Floodfill.java	100%	26/26	88%	29/33	88%	29/33
IncreasingArray.java	100%	7/7	90%	9/10	90%	9/10
Inversion_count.java	100%	21/21	100%	35/35	100%	35/35
Island.java	100%	19/19	100%	28/28	100%	28/28
Matrix.java	100%	19/19	100%	19/19	100%	19/19
Maximal_Rectangle.java	100%	23/23	95%	18/19	95%	18/19
Palindrome.java	100%	25/25	100%	23/23	100%	23/23
Parse_boolean.java	100%	21/21	100%	13/13	100%	13/13
Subarray_Ranges.java	100%	23/23	92%	23/25	92%	23/25
Twoknights.java	100%	30/30	91%	59/65	91%	59/65
Twosets.java	100%	49/49	94%	16/17	94%	16/17
Wordsearch.java	91%	29/32	96%	45/47	96%	45/47
frequent.java	98%	55/56	89%	66/74	93%	66/71
getdiameter.java	100%	22/22	100%	12/12	100%	12/12
llongestsequence.java	100%	17/17	100%	26/26	100%	26/26
matrixchain.java	100%	17/17	100%	19/19	100%	19/19
minimumsumpath.java	100%	12/12	100%	15/15	100%	15/15
nextgreaterelement.java	100%	10/10	90%	9/10	90%	9/10
nextsmaller.java	100%	11/11	100%	8/8	100%	8/8
overlappingintervals.java	100%	11/11	100%	12/12	100%	12/12
rottenoranges.java	100%	33/33	94%	30/32	94%	30/32
savingmoney.java	100%	16/16	89%	17/19	89%	17/19
sudokusolver.java	100%	27/27	84%	32/38	84%	32/38
threeSum.java	100%	20/20	82%	14/17	82%	14/17
tri_1.java	100%	33/33	100%	21/21	100%	21/21
tri_2.java	100%	38/38	88%	23/26	88%	23/26

We were unable to eliminate certain mutants because of the presence of equivalent mutants, specifically those generated by the CONDITIONALS_BOUNDARY Mutant operator. This operator alters conditional statements, potentially resulting in the same outcome regardless of the test cases applied. As a consequence, our test suite may not effectively distinguish between these equivalent mutants, leading to the inability to fully eliminate them. For example,

```
1  package com.myproblemset.app;
2  import java.util.*;
3  class threeSum {
4      public static List<List<Integer>> threeSum_(int[] nums) {
5          int target = 0;
6  1       Arrays.sort(nums);
7          Set<List<Integer>> s = new HashSet<>();
8          List<List<Integer>> output = new ArrayList<>();
9  2       for (int i = 0; i < nums.length; i++){
10 1            int j = i + 1;
11 1            int k = nums.length - 1;
12 2            while (j < k) {
13 2                int sum = nums[i] + nums[j] + nums[k];
14 1                if (sum == target) {
15                      s.add(Arrays.asList(nums[i], nums[j], nums[k]));
16 1                    j++;
17 1                    k--;
18 2                } else if (sum < target) {
19 1                    j++;
20                  } else {
21 1                    k--;
22                  }
23              }
24          }
25          output.addAll(s);
26 1        return output;
27      }
28 }
```

For instance, in one of our codes, the impact of the CONDITIONALS_BOUNDARY Mutant operator is evident. When applied to line 9, it produces identical outcomes. This is due to the fact that the inner loop only iterates up to the maximum length of the array. Moreover, altering the condition from '<' to '≤' at line 18 becomes inconsequential, as the IF condition wouldn't be executed in that scenario.

```
6   1. removed call to java/util/Arrays::sort → KILLED
9   1. changed conditional boundary → SURVIVED
    2. negated conditional → KILLED
10  1. Replaced integer addition with subtraction → KILLED
11  1. Replaced integer subtraction with addition → KILLED
12  1. changed conditional boundary → KILLED
    2. negated conditional → KILLED
13  1. Replaced integer addition with subtraction → KILLED
    2. Replaced integer addition with subtraction → KILLED
14  1. negated conditional → KILLED
16  1. Changed increment from 1 to -1 → KILLED
17  1. Changed increment from -1 to 1 → KILLED
18  1. changed conditional boundary → SURVIVED
    2. negated conditional → SURVIVED
19  1. Changed increment from 1 to -1 → KILLED
21  1. Changed increment from -1 to 1 → KILLED
26  1. replaced return value with Collections.emptyList for com/myproblemset/app/threeSum::threeSum_ → KILLED
```

Integration Testing:

We carefully selected modules identified by functions that call one another internally. Strategic application of integration testing operators such as EMPTY_RETURNS, TRUE_RETURNS, FALSE_RETURNS, NULL_RETURNS, and PRIMITIVE_RETURNS was employed in our PiTest testing process. Our ability to assess the smooth operation of various functions inside the codebase was made possible by these operators, which were crucial in enabling thorough integration testing. Our test suite's ability to capture complex interactions between related functions and the program's underlying mechanisms were both much enhanced by this method.

For example:

Island.java

```
1   package com.myproblemset.app;
2   import java.io.*;
3   import java.lang.*;
4   import java.util.*;
5
6   public class Island {
7
8       boolean isSafe(int M[][], int row, int col,boolean visited[][],int ROW,int COL)
9           {
10 11                return (row >= 0) && (row < ROW) && (col >= 0)&& (col < COL) && (M[row][col] == 1 && !visited[row][col]);
11          }
12
13          void DFS(int M[][], int row, int col,boolean visited[][],int ROW,int COL)
14          {
15                  int rowNbr[]
16                          = new int[] { -1, -1, -1, 0, 0, 1, 1, 1 };
17                  int colNbr[]
18                          = new int[] { -1, 0, 1, -1, 1, -1, 0, 1 };
19
20                  visited[row][col] = true;
21
22 2                for (int k = 0; k < 8; ++k)
23 3                       if (isSafe(M, row + rowNbr[k], col + colNbr[k],visited,ROW,COL))
24 3                              DFS(M, row + rowNbr[k], col + colNbr[k],visited,ROW,COL);
25          }
26
27          int countIslands(int M[][])
28          {
29          int numRows = M.length;
30          int numColumns = M[0].length;
31
32                  boolean visited[][] = new boolean[numRows][numColumns];
33                  int count = 0;
34 2                for (int i = 0; i < numRows; ++i)
35 2                       for (int j = 0; j < numColumns; ++j)
36 2                              if (M[i][j] == 1
37                                      && !visited[i][j]) // If a cell with
38                              {
39 1                                    DFS(M, i, j, visited,numRows,numColumns);
40 1                                    ++count;
41                              }
42
43 1                return count;
44          }
45
46  }
```

In the specified code segments, we observed several modifications during the integration of mutation. Specifically, in lines 24 and 39, we eliminated the invocation statement, **METHOD CALL** Mutator effectively removing a call to a particular function. Additionally, in lines 10 and 11, **TRUE RETURN** Mutator altered the original return value to a boolean 'True.' Finally, at line 43, **EMPTY RETURN** Mutator substituted the integer return value with 0. These modifications were introduced as part of the integration process to assess the effectiveness of the mutations and their impact on the overall code behavior.

Mutations

```
1.  1. changed conditional boundary → KILLED
    2. changed conditional boundary → KILLED
    3. changed conditional boundary → KILLED
    4. changed conditional boundary → KILLED
    5. negated conditional → KILLED
10  6. negated conditional → KILLED
    7. negated conditional → KILLED
    8. negated conditional → KILLED
    9. negated conditional → KILLED
    10. negated conditional → KILLED
    11. replaced boolean return with true for com/myproblemset/app/Island::isSafe → KILLED
22  1. changed conditional boundary → KILLED
    2. negated conditional → KILLED
23  1. Replaced integer addition with subtraction → KILLED
    2. Replaced integer addition with subtraction → KILLED
    3. negated conditional → KILLED
24  1. Replaced integer addition with subtraction → KILLED
    2. Replaced integer addition with subtraction → KILLED
    3. removed call to com/myproblemset/app/Island::DFS → KILLED
34  1. changed conditional boundary → KILLED
    2. negated conditional → KILLED
35  1. changed conditional boundary → KILLED
    2. negated conditional → KILLED
36  1. negated conditional → KILLED
    2. negated conditional → KILLED
39  1. removed call to com/myproblemset/app/Island::DFS → KILLED
40  1. Changed increment from 1 to -1 → KILLED
43  1. replaced int return with 0 for com/myproblemset/app/Island::countIslands → KILLED
```

Mutation Operators :

Operators used to generate mutants of our Source Code,

- **CONDITIONALS_BOUNDARY**

Original conditional	Mutated conditional
<	<=
<=	<
>	>=
>=	>

- **EMPTY_RETURNS**

Replaces return values with an ‘empty’ value for that type

- **FALSE_RETURNS**

Replaces primitive and boxed boolean return values with false.

- **INCREMENTS**

The increments mutator will mutate increments, decrements and assignment increments and decrements of local variables (stack variables). It will replace increments with decrements and vice versa.

- **INVERT_NEGS**

The invert negatives mutator inverts negation of integer and floating point variables.

- **MATH**

The math mutator replaces binary arithmetic operations for either integer or floating-point arithmetic with another operation.

- **NEGATE_CONDITIONALS**

Original conditional	Mutated conditional
==	!=
!=	==
<=	>
>=	<
<	>=
>	<=

- **NULL_RETURNS**

Replaces return values with null. Methods that can be mutated by the EMPTY_RETURNS mutator or that are directly annotated with NotNull will not be mutated.

- **PRIMITIVE_RETURNS**

Replaces int, short, long, char, float and double return values with 0.

- **TRUE_RETURNS**

Replaces primitive and boxed boolean return values with true.

- **VOID_METHOD_CALLS**

The void method call mutator removes method calls to void methods.

CONTRIBUTION:

Vamshi:

- Worked on 14 problems and have written testcase for those problems
- Worked on writing the Report

Rahul:

- Worked on 14 problems and have written testcase for those problems
- Worked on writing the Report

REFERENCES:

Pit Test :

PIT Mutation Testing

PIT is a state of the art mutation testing system, providing gold standard test coverage for Java and the jvm. It's fast, scalable and integrates with modern test and build tooling.

https://pitest.org/

Maven Installation and setup:

How to Install Maven on Linux (Ubuntu) | DigitalOcean

Technical tutorials, Q&A, events — This is an inclusive place where developers can find or lend support and discover new ways to contribute to the community.

https://www.digitalocean.com/community/tutorials/install-maven-linux-ubuntu

Striver SDE sheet:

Striver’s SDE Sheet – Top Coding Interview Problems

Check out the Most Asked Coding Interview Problem list compiled by Raja Vikramaditya AKA Striver. Striver SDE Sheet for top coding interview problems"

https://takeuforward.org/interviews/strivers-sde-sheet-top-coding-interview-problems/

CSES Problem Sheet:

CSES - CSES Problem Set - Tasks

https://cses.fi/problemset/list/