Software Testing

Final Project
Mutation Testing (Unit & Integration)

Problem Statement:

Mutation testing: Projects that use mutation testing, based on mutation operators applied at the level of a statement within a method or a function and at the integration level. The mutated program needs to be strongly killed by the designed test cases. At least three different mutation opera- tors should be used at the unit level and three different mutation operators at the integration level should be used.

Overview:

Wikipedia - "Mutation testing (or *mutation analysis* or *program mutation*) is used to design new software tests and evaluate the quality of existing software tests. Mutation testing involves modifying a program in small ways. Each mutated version is called a *mutant*. Tests detect and reject mutants by causing the behavior of the original version to differ from the mutant. This is called *killing* the mutant. Mutants are based on well-defined *mutation operators* that either mimic typical programming errors (such as using the wrong operator or variable name) or force the creation of valuable tests (such as dividing each expression by zero). Mutation testing is a form of *white-box testing*."

Mutation Unit Testing involves evaluating the effectiveness of a testset at the unit level on mutated code. **Mutation integration testing** involves evaluating how the testset copes with interactions and collaborations between different units or components in the system.

Codebase:

The codebase under scrutiny consists of a diverse range of data structures and algorithms, showcasing a comprehensive demonstration of Object-Oriented Programming (OOP) principles. This code repository serves as a collection for multiple Java classes that encompass various data structures like graphs, trees, sorting algorithms, dynamic programming algorithms, linked lists, etc. We have also managed to incorporate several OOPs concepts such as inheritance and polymorphism in order to demonstrate integration testing as well.

 - https://github.com/yashk0311/Mutation-testing.git

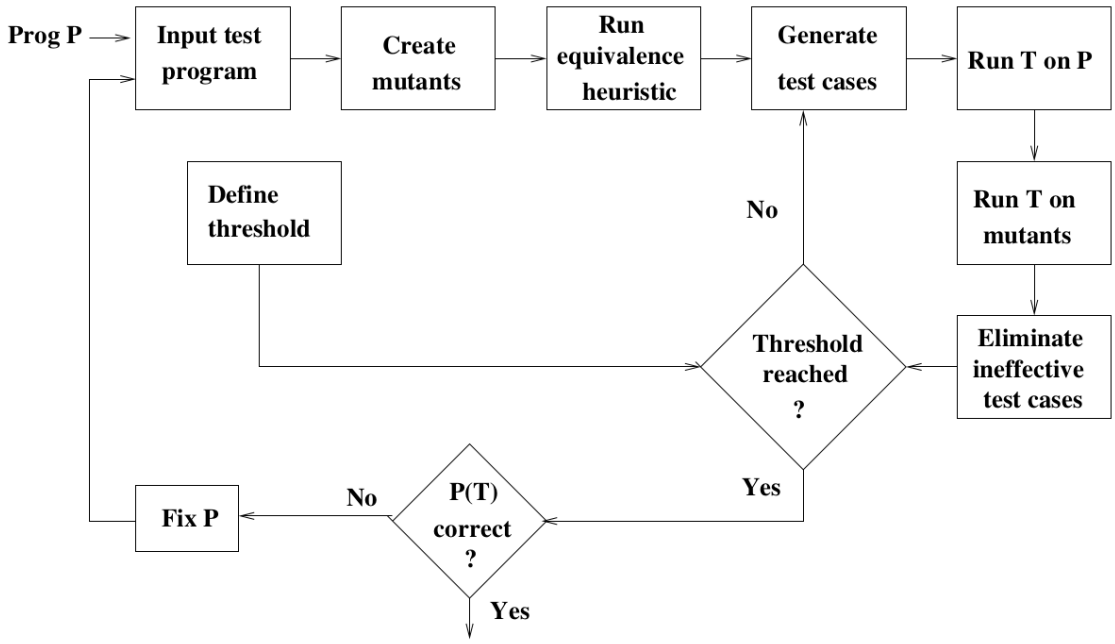
Tools Used:

Maven - A build automation tool primarily used for managing the project's build lifecycle, handling dependencies, and creating project documentation. It uses a Project Object Model (POM) to define the project structure, dependencies, and build process configuration.

JUnit - Java testing framework specifically designed for unit testing. It provides a simple and effective way to write and execute repeatable automated tests for Java applications. JUnit helps in creating test cases, defining test methods, and asserting expected outcomes against actual results.

PITest - Mutation testing tool for Java applications. It helps in assessing the effectiveness of existing test suites by creating mutants or modified versions of the codebase and checking if the tests can detect these mutations. PITest generates mutation reports that identify areas of weakness in the test suite, highlighting where improvements can be made to enhance test coverage.

Mutation testing: Process



Unit Testing:

Active mutators

- INCREMENTS_MUTATOR
- VOID_METHOD_CALL_MUTATOR
- RETURN_VALS_MUTATOR
- MATH_MUTATOR
- NEGATE_CONDITIONALS_MUTATOR
- INVERT_NEGS_MUTATOR
- CONDITIONALS_BOUNDARY_MUTATOR

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
19	100%	536/536	92%	426/464

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
dsa.LinkedListTest	6	100%	183/183	90%	95/106
dsa.algorithms	13	100%	353/353	92%	331/358

Report generated by PIT 1.4.3

Pit Test Coverage Report

Package Summary

dsa.LinkedListTest

Number of Classes	Line Coverage		Mutation Coverage	
6	100%	183/183	90%	95/106

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
FindKthElement.java	100%	43/43	93%	27/29
LLRemoveDuplicates.java	100%	29/29	82%	9/11
MiddleElement.java	100%	24/24	94%	17/18
PalindromeList.java	100%	33/33	75%	9/12
ReverseLinkedList.java	100%	23/23	93%	13/14
SumLists.java	100%	31/31	91%	20/22

Report generated by PIT 1.4.3

Pit Test Coverage Report

Package Summary

dsa.algorithms

Number of Classes	Line Coverage		Mutation Coverage	
13	100%	353/353	94%	335/358

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
BitsCounter.java	100%	16/16	100%	14/14
Factorial.java	100%	18/18	95%	20/21
Knapsack.java	100%	13/13	100%	23/23
LongestConsecSeq.java	100%	34/34	97%	33/34
MinPathSum.java	100%	14/14	100%	18/18
MissingNumber.java	100%	16/16	100%	10/10
MoveZeros.java	100%	28/28	96%	22/23
PairSum.java	100%	19/19	100%	10/10
RemDupElements.java	100%	19/19	100%	16/16
RotateMatrix.java	100%	29/29	82%	18/22
SetZeros.java	100%	22/22	100%	20/20
SplitArray.java	100%	63/63	83%	57/69
StrassenMM.java	100%	62/62	95%	74/78

Report generated by PIT 1.4.3

Integration Testing:

Active mutators

- CONSTRUCTOR_CALL_MUTATOR
- INCREMENTS_MUTATOR
- EXPERIMENTAL_MEMBER_VARIABLE_MUTATOR
- RETURN_VALS_MUTATOR
- ARGUMENT_PROPAGATION_MUTATOR
- MATH_MUTATOR
- NEGATE_CONDITIONALS_MUTATOR
- NON_VOID_METHOD_CALL_MUTATOR

Pit Test Coverage Report

Package Summary

org.softtest

Number of Classes	Line Coverage		Mutation Coverage	
11	97%	327/337	88%	310/352

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
BinaryTree.java	92%	67/73	84%	74/88
Graph.java	100%	61/61	95%	71/75
MergeSort.java	100%	33/33	100%	27/27
QuickSort.java	100%	18/18	100%	14/14
ShortestPathBFS.java	97%	30/31	82%	18/22
TopologicalSortDFS.java	100%	25/25	96%	23/24
Tree.java	100%	20/20	81%	21/26
TreeNode.java	100%	3/3	100%	1/1
Trie.java	94%	30/32	83%	25/30
WordSearch.java	96%	22/23	77%	30/39
dummyMain.java	100%	18/18	100%	6/6

Steps to Run the code:

1) Clone the GitHub repository
2) Open it in a maven-supported IDE (such as IntellIJ).
3) Inside the /Unit-Testing directory, run “mvn clean test” to obtain the report for unit testing (can be viewed in the index.html file which shall get formed under pit-reports in the /target directory).
4) Inside the /Integration-Testing directory, run “maven clean test” to obtain the report for integration testing (can be viewed in the index.html file which shall get formed under pit-reports in the /target directory

Contributions:

Yash Koushik Kocherla (IMT2020033) : Mutation Unit Testing

Chaitanya Manas Cheedella (IMT2020053) : Mutation Integration Testing