# Project Report
# Mutation Source Code Testing

**Team Members :**

Ajay Chajjed - IMT2019006
Amitha Reddy - IMT2019023
Vismaya Solanki - IMT2019095

## Abstract

We have written code with functions in java and performed mutation testing using PITclipse which is a plugin to run the mutation testing tool against the given unit test cases. This report contains a detailed description of how this is done, what functions we have used and more about the pit tool and results.

## Introduction

Software testing is the process of reviewing and validating that a software application or product performs as intended. Testing has advantages such as bug prevention, lower development costs, and better performance.

Mutation testing is a Software Testing technique in the form of white box testing i.e it analyzes the internal structure of the code like used data structures, code design and internal structure rather than just functionality as in black box testing.

This testing involves in changing/mutating specific source code statements to see if the test cases can detect source code errors. To do this many versions of faults are introduced into the source code called mutants. The aim is to make the mutant version fail in order to show the effectiveness of the test cases. If the original program and mutant programs generate different outputs, then that means the mutant is killed by the test case and the test case is good enough to detect the change between the original and the mutant program. A mutant can be as simple as changing the operators, Ex- mutating  < to >. The mutant score is calculated based on how many mutants are killed. The purpose of mutation testing is to guarantee the quality of test cases such that they should fail the modified source code.

Mutation testing ensures the coverage of the source code and has the ability to discover faults and ambiguities in the program. But creating a mutant program is very time-consuming and complicated. So, we use automation tools for mutation. In our case, we used PITclipse to perform mutation testing.

## Source Code Description

The source code contains popular tree and graph algorithms. Following are some of them mentioned:

a) BFS,
b) DFS
c) Minimum Spanning Tree: Kruskals and Prims algorithm.
d) Floyd Warshall algorithm, Dijkstra’s algorithm.
e) Bridges in a tree.
f) Topological Sorting
g) Strongly Connected Component
h) Number of Connected Components etc.

TreeAlgorithms.java and GraphAlgorithms.java files contains the source code which have all the algorithms mentioned and more.
AlgorithmsTest.java file has unit test cases for the JUnit java class. After running this file as PIT Mutation Test, we will get the results of the mutation testing as PIT Summary which will tell us how many mutations have been killed, line coverage and the Test Strength.

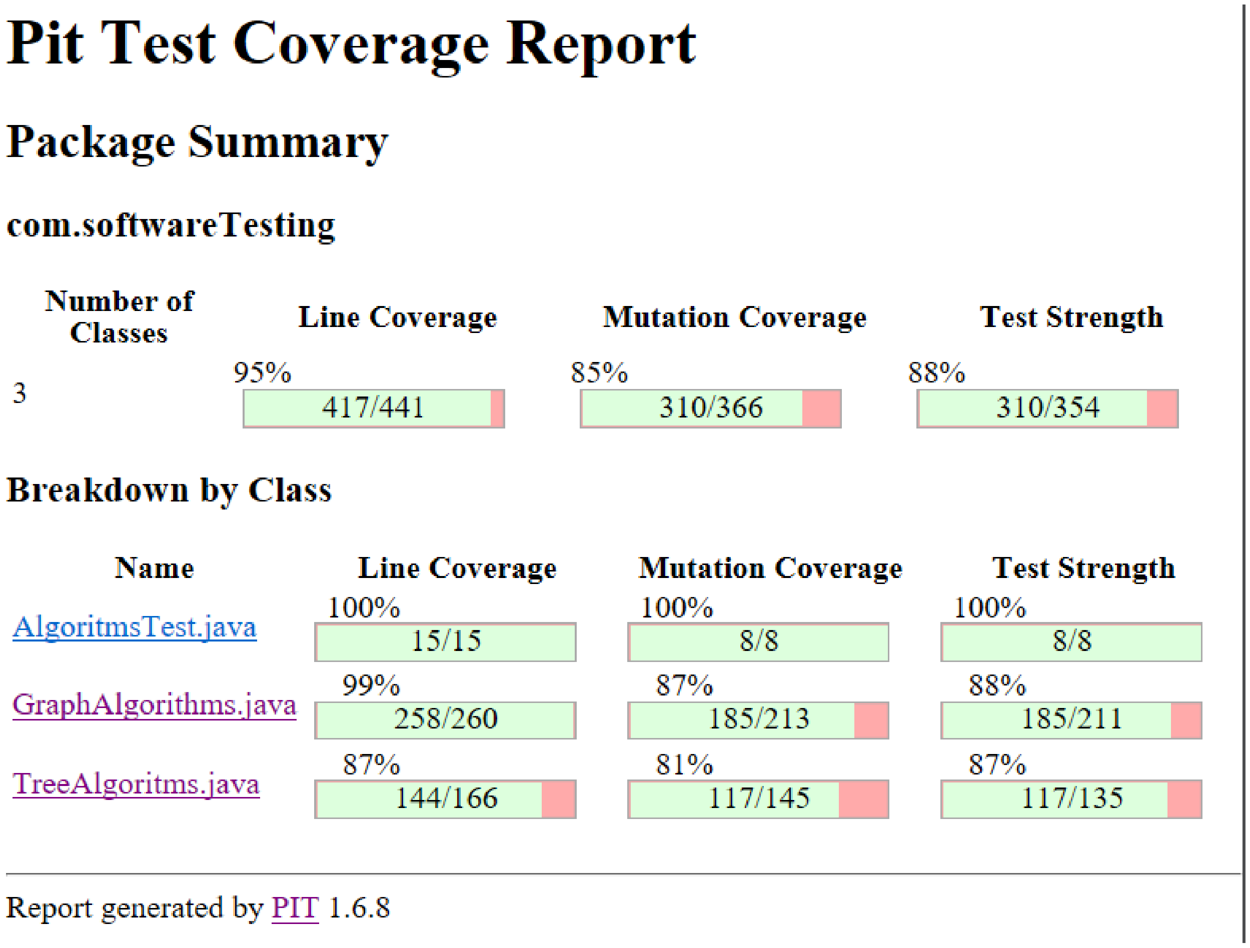
## How Mutation Testing is done:

Faults are introduced to the program by creating many versions called mutants. Each mutant is a single fault generated by using mutation operators in the source code. Test cases are now applied to both the actual program and the mutant program and the results are compared. If the outputs are different that means the test case is good enough to find the fault and the mutant is killed. If they are same, the test case failed to detect the fault and the mutant survived.

## Mutation Testing tool (PIT)

PIT is a cutting-edge mutation testing system that offers Java and the jvm gold standard test coverage. It interacts with current test and build tooling and is fast and scalable. It runs the unit tests against automatically modified versions of the application code. In the end, it gives detailed information about both line and mutation coverage. The mutation score is calculated based on percentage of mutations killed.

## Results



# Pit Test Coverage Report

## Package Summary

### com.softwareTesting

| Number of Classes | Line Coverage | Mutation Coverage | Test Strength |
|---|---|---|---|
| 3 | 95% 417/441 | 85% 310/366 | 88% 310/354 |

## Breakdown by Class

| Name | Line Coverage | Mutation Coverage | Test Strength |
|---|---|---|---|
| AlgoritmsTest.java | 100% 15/15 | 100% 8/8 | 100% 8/8 |
| GraphAlgorithms.java | 99% 258/260 | 87% 185/213 | 88% 185/211 |
| TreeAlgoritms.java | 87% 144/166 | 81% 117/145 | 87% 117/135 |

Report generated by PIT 1.6.8

## Contributions

It was a team project, all of us found the different algorithms, unit tested it and created test cases for the same and then later combined it to do the final mutation testing using PIT testing plugin on Eclipse IDE.

## References

https://www.geeksforgeeks.org/

https://www.javatpoint.com/
https://www.programiz.com/
https://stackoverflow.com/
https://leetcode.com/