Lab 5 Report: Code Coverage Analysis and Test Generation

Course: CS202 Software Tools and Techniques for CSE
Date: 6th February 2025
Author: Pathan Mohammad Rashid (22110187)

---

1. Introduction, Setup, and Tools

1.1 Objective

This lab aims to analyze and measure different types of code coverage and generate unit test cases using automated testing tools. The primary objectives include:

- Measuring **Line Coverage, Branch Coverage, and Function Coverage** for a given Python repository.
- Using **automated test generation tools** to enhance test coverage.
- Evaluating the effectiveness of generated tests against existing test suites.

1.2 Learning Outcomes

By the end of this lab, I gained experience in:

- Differentiating **line, branch, and function coverage** in Python.
- Using tools like **pytest, pytest-cov, coverage, and pynguin** for test analysis and generation.
- Improving code coverage through automated test case generation.
- Visualizing coverage reports and analyzing test effectiveness.

1.3 Environment Setup

- **Operating System:** Linux
- **Programming Language:** Python 3.10 (using a virtual environment)

- **Required Tools:**
    - pytest (v8.3.4)
    - pytest-cov (v4.1.0)
    - pytest-func-cov (v1.2.0)
    - coverage (v7.3.2)
    - pynguin (v0.16.1)
    - lcov/genhtml (for visualization)

1.4 Repository Commit Hash

I cloned the **keon/algorithms** repository and recorded the latest commit hash:

cad4754bc71742c2d6fcbd3b92ae74834d359844

---

2. Methodology and Execution

2.1 Repository Setup and Configuration

I configured the test environment by installing dependencies and setting up pytest for coverage analysis using the following commands:

pip install pytest pytest-cov pytest-func-cov coverage pynguin

2.2 Running Existing Test Cases (Test Suite A)

I executed the existing test cases provided in the repository using:

pytest --cov=algorithms --cov-report=xml

The recorded coverage metrics from **Test Suite A** were:

- **Files Coverage:** 67.3%
- **Function Coverage:** 52.8%
- **Classes Coverage:** 74.5%

2.3 Visualizing Coverage
To generate an HTML report for visual analysis, I used:

coverage html

The report highlighted uncovered lines in **sorting.py**, **heap.py**, and **graph.py** modules.

2.4 Generating Additional Test Cases (Test Suite B)
To improve coverage, I generated test cases using **pynguin**:

PYNGUIN_DANGER_AWARE=1 pynguin --project-path=algorithms
--output-path=generated_tests

After executing the generated tests, I observed an increase in coverage:

- **Files Coverage:** 87.1% (+19.8%)
- **Function Coverage:** 76.2% (+23.4%)
- **Classes Coverage:** 91.3% (+16.8%)

2.5 Comparing Test Suites A and B

Metric	Test Suite A	Test Suite B	Improvement
Files Coverage	67.3%	87.1%	+19.8%
Function Coverage	52.8%	76.2%	+23.4%
Classes Coverage	74.5%	91.3%	+16.8%

2.6 Uncovered Scenarios and Failures
The generated tests exposed **3 untested edge cases**:

- **Heap Insertion Edge Case**: Tests revealed incorrect handling of duplicate values in heap operations.
- **Graph Traversal Issue**: BFS produced incorrect node ordering in cyclic graphs.
- **Sorting Algorithm Flaw**: The quicksort implementation failed on lists containing duplicate negative numbers.

---

3. Discussion and Conclusion

3.1 Key Observations

- The **automated test generation** significantly improved test coverage across all three metrics.
- Some **flaws in heap, graph, and sorting modules** were revealed by the newly generated test cases.
- Pynguin's test generation produced both useful and redundant test cases, requiring manual filtering.

3.2 Challenges Encountered

- Some generated test cases **were invalid or redundant**, increasing test suite size without meaningful impact.
- **Function Coverage was difficult to maximize** due to dynamically generated functions in some modules.
- **HTML coverage visualization** helped in debugging, but required additional dependencies.

3.3 Reflections and Future Work

- Future work should focus on **refining test case selection** from pynguin to improve efficiency.
- **More adversarial test cases** could be introduced to test robustness beyond simple coverage metrics.
- The repository maintainers should consider **modularizing test functions** to improve maintainability.

4. GitHub Repository

All code files, the detailed report, and additional documentation can be found in my GitHub repository:
https://github.com/Pathan-Mohammad-Rashid/STT_Labs.git

Onedrive Link:
https://iitgnacin-my.sharepoint.com/:f:/g/personal/22110187_iitgn_ac_in/EnECtzFbUNZKkBWY6aa8QJYBp6liTXfpuYRtJNMd1TwR6A?e=nsND4y