CSED232 Assignment 3

Due Friday, March 28

Background: Graphs

- A *directed graph* is a pair $G = (V, E)$, where V is a set of *vertices* (also called nodes), and $E \subseteq V \times V$ is a set of edges that connect pairs of vertices. For example, Fig. 1 shows the graph:

$$V = \{1, 2, 3, 4, 5, 6\}, \quad E = \{(1, 2), (1, 4), (2, 4), (3, 6), (6, 3)\}$$

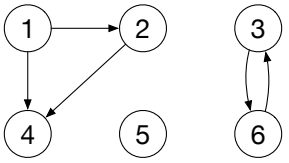

Figure 1: A directed graph

Problem 1: Black-box Test Cases for Graphs

- In this assignment, we consider a generic abstract interface for graphs, Graph<N>, where vertices are represented as elements of a given (immutable and comparable) type N.
- The goal is to write a high-quality test suite for Graph<N> according to its specifications.
  - Because only abstract specifications are available, you will write *black-box test cases* for the interface, based on equivalence partitioning.
  - E.g., for the method `addVertex`(v) of Graph<N>, there are two equivalence classes based on the description: v is already in the graph, or v is not yet in the graph.
- For each method and each equivalence class, write a test method in AbstractGraphTest<N,G>:
  - AbstractGraphTest<N,G> is the abstract test class for vertex type N and graph type G. It contains one graph of type G, eight vertices of type N, and some example test methods.
  - The abstract test class should depend only on the abstract interface, namely, Graph<N>; importing concrete implementations is not allowed.
- Your black-box test cases will be graded based on whether they clearly describe different scenarios from the specifications using equivalence partitioning.

Problem 2: Implementing Edge Lists

- In this problem, we will implement a direct graph using an *edge list representation*.[1] Implement the class EdgeListGraph<N>, which is a subclass of Graph<N>.
- You must use the following representation provided in the class EdgeListGraph<N>: a set[2] of vertices and a list[3] of edges.

```
private final @NotNull Set<N> vertices;
private final @NotNull List<Edge<N>> edges;
```

- For example, the graph in Fig. 1 can be represented as follows:

$$\textit{vertices} = \{1, 2, 3, 4, 5, 6\}, \qquad \textit{edges} = [(1, 2), (1, 4), (2, 4), (3, 6), (6, 3)]$$

[1] https://en.wikipedia.org/wiki/Edge_list
[2] https://docs.oracle.com/en/java/javase/23/docs/api/java.base/java/util/Set.html
[3] https://docs.oracle.com/en/java/javase/23/docs/api/java.base/java/util/List.html

Problem 3: Implementing Adjacency Lists

- Now we will implement a direct graph using a variation of the *adjacency list representation.*[4] Implement the class AdjacencyListGraph<N>, which is a subclass of Graph<N>.
- You must use the following representation provided in the class AdjacencyListGraph<N>, a (sorted) map from vertices to the (sorted) set of their adjacent vertices.

```
private final @NotNull SortedMap<V, SortedSet<V>> adjMap;
```

- For example, the graph in Fig. 1 is represented as the following sorted map[5]

$$\{1 \mapsto \{2,4\}, 2 \mapsto \{1,4\}, 3 \mapsto \{6\}, 4 \mapsto \{1,2\}, 5 \mapsto \emptyset, 6 \mapsto \{3\}\}$$

Problem 4: Writing White-box Test Cases

- There are two concrete test classes that extend AbstractGraphTest: DoubleEdgeListGraphTest and StringAdjacencyListGraphTest. They contain `setUp()` to initialize abstract graphs and vertices.
  - You may add more *white-box test cases* to these test classes to achieve higher code coverage for EdgeListGraph<N> and AdjacencyListGraph<N>, if needed.
- Your submitted tests must achieve at least 80% **branch coverage**. Your black-box test cases should already provide high coverage, but you may add more white-box test cases if necessary.
  - Each test method should test a single behavior using appropriate assertions. *Do not add arbitrary code to your test methods to just increase coverage.*
- After executing gradle test, you can generate coverage reports using gradle jacocoTestReport. The reports will be stored in `build/reports/jacoco/test`.

General Instruction

- The `src/main` directory contains the skeleton code. You should implement all classes and methods marked with *TODO*. Before writing code, carefully read the descriptions in the source files.
- The `src/test` directory contains test cases. Use JaCoCo to find out the coverage achieved by your tests. Upload the JaCoCo report in CSV (`build/reports/jacoco/test/jacocoTestReport.csv`).
- Do not modify the existing interface, class names, or the signatures of public methods. However, you may add additional private methods if needed.

Turning in

1. We use Github Classroom to manage homework. Click on the following link, and accept the assignment: `https://classroom.github.com/a/OnH8NUt0`
2. Your assignment repository should now be created. Clone the repository, complete your homework (including the JaCoCo coverage report), and push the changes to GitHub before the deadline.
3. The JaCoCo coverage report in CSV should be uploaded to the root directory of your repository (i.e., the same directory where `build.gradle.kts` is located).

Java Reference

- Java Language Specification: `https://docs.oracle.com/javase/specs/`
- Learn Java: `https://dev.java/learn/`
- Core Java, Volume I: Fundamentals 13th by Cay S. Horstmann, Pearson, 2024 (available online at the POSTECH digital library `http://library.postech.ac.kr`)

[4] `https://en.wikipedia.org/wiki/Adjacency_list`
[5] `https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/SortedMap.html`