Assignment 2: Graphs

Data Structures and Algorithms

Document Version: 1.0

Introduction

The goal of this assignment is to write implementations of the Adjacency List and Adjacency Matrix graph implementation strategies.

Download the file `Assignment-2-Source.zip` from Brightspace. The contents of this file include the following important classes and interfaces:

- All the interfaces you require for making a Graph. In particular, the `IGraph` interface includes comments describing all of the methods that a graph implementation should contain (these are in the `graph.iface` package).
- An implementation of a Linked List, which you will need within your implementation (this is in the `graph.util` package and is named `DLinkedList`). You should not use any built-in Java data structures for this assignment (You will need to use basic Java arrays to implement the Adjacency Matrix).
- An example of a Graph implementation: `EdgeListGraph`. You should study this file carefully, as the other implementations have some similar characteristics (this is in the `graph.impl` package).
- A program called `EdgeListTest` that shows some examples of code that can test some of the methods in the graph implementation (this is in the default package).

Tasks

1. **Implement an Adjacency List graph (in a file called `AdjacencyListGraph.java`)**
2. **Implement an Adjacency Matrix graph (in a file called `AdjacencyMatrixGraph.java`)**
3. **Test these implementations (in files called `AdjacencyListTest.java` and `AdjacencyMatrixTest.java`).**

For each graph implementation, it is important that you follow the implementation strategy correctly, that your code is well structured and well documented (using appropriate comments) and is free of bugs.

For each graph type, you should also create a new **testing** class similar to `EdgeListTest` to check that your implementation is correct. **Note:** The program I have provided does not test all of the methods in the graph implementation. You should add some more tests to check other methods (e.g. removing the vertex HNL should mean that the number of incident edges on LAX to decrease by one). Testing should not be simply to copy the sample tests for the Edge List graph; more tests **must** be added.

Ideally, testing should make sure that the different operations of the graph(s) are all tested (e.g. adding and removing vertices and edges, checking that correct vertices are adjacent or not, incident edges are correct, etc.). It should also check that the consequences of these operations are correct (e.g. removing a vertex removes its incident edges also, removing an edge means that its end vertices are no longer adjacent, etc.). The testing code should automatically detect whether a problem has occurred and can inform the user.

Submission

- This is an **individual programming assignment**. Therefore, all code must be written by yourself. Assignment 1 contained some advice about avoiding plagiarism in programming assignments.

- All code should be well-formatted and well-commented to describe what it is trying to do.

- Submit a single .zip file to Brightspace, with the following contents:
  - The `AdjacencyListGraph`, `AdjacencyMatrixGraph`, `AdjacencyListTest` and `AdjacencyMatrixTest` classes.

  - If your testing code imports some other graphs from a text file, this text file may be included also.

  - Do not include any extra Java files and do not submit your entire IntelliJ project.

Grading

The following grading scheme will be used to grade the assignment:

Item	Weight
Correct implementation of Adjacency List Graph	30%
Correct implementation of Adjacency Matrix Graph	30%
Testing of Adjacency List Graph	15%
Testing of Adjacency Matrix Graph	15%
Code clarity, organisation, commenting	10%