哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2018 年春季学期

计算机学院大二软件构造课程

Lab 2 实验报告

姓名	林子杰
学号	1160800218
班号	136101
电子邮件	1160800218@stu.hit.edu.cn
手机号码	15636703603

目录

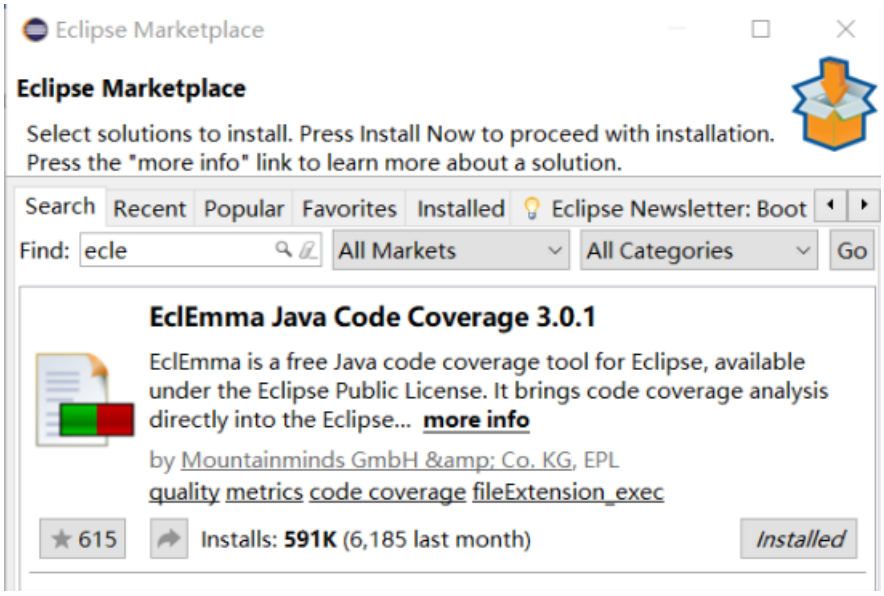
1 实验目标概述

本次实验训练抽象数据类型（ADT）的设计、规约、测试，并使用面向对象编程（OOP）技术实现 ADT。具体来说：

- 针对给定的应用问题，从问题描述中识别所需的 ADT；
- 设计 ADT 规约（pre-condition、post-condition）并评估规约的质量；
- 根据 ADT 的规约设计测试用例；
- ADT 的泛型化；
- 根据规约设计 ADT 的多种不同的实现；
- 针对每种实现，设计其表示（representation）、表示不变性（rep invariant）、抽象过程（abstraction function）
- 使用 OOP 实现 ADT，并判定表示不变性是否违反、各实现是否存在表示外泄（rep exposure）；
- 测试 ADT 的实现并评估测试的覆盖度；
- 使用 ADT 及其实现，为应用问题开发程序；
- 在测试代码中，能够写出 testing strategy 并据此设计测试用例。

2 实验环境配置

从 Eclipse Marketplace 获取 EclEmma



在这里给出你的 GitHub Lab2 仓库的 URL 地址（Lab2-学号）。

（获取实验库失败，自建了一个）
https://github.com/1160800218/Lab2_1160800218.git

3 实验过程

请仔细对照实验手册，针对三个问题中的每一项任务，在下面各节中记录你的实验过程、阐述你的设计思路和问题求解思路，可辅之以示意图或关键源代码加以说明（但千万不要把你的源代码全部粘贴过来！）。

3.1 Poetic Walks

该实验要求我们按照 MIT 上的操作步骤逐步完成自己的第一个泛型 Graph<L>的设计和测试，并应用该泛型实现 poetic walks 功能。

3.1.1 Get the code and prepare Git repository

从 https://github.com/rainywang/Spring2018_HITCS_SC_Lab2/tree/master/P1 获取实验库，在 GitHub 账号上新建一个库，并将实验内容 push 到上面。

3.1.2 Problem 1: Test `Graph <String>`

以下各部分，请按照 MIT 页面上相应部分的要求，逐项列出你的设计和实现思路/过程/结果。

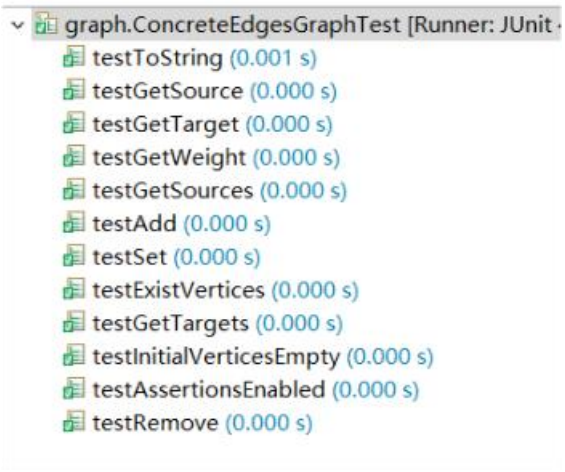
3.1.3 Problem 2: Implement `Graph <String>`

以下各部分，请按照 MIT 页面上相应部分的要求，逐项列出你的设计和实现思路/过程/结果。

3.1.3.1 Implement `ConcreteEdgesGraph`

- 该图类给出的对象有 vertices 和 edges，适用邻接矩阵实现图的功能；
- 根据 MIT 上对 Graph<L>的各种方法的说明完成各方法的功能；
- 设计 Edge<String>类，类中对象应包含边的起点、终点和权重，为防止 rep exposure 应都为 private final，同时需要实现获取各对象的方法，以便 Graph<String>方法的实现；
- 设计 set()方法时设置 hasedge 值用来记录图中是否已经存在待设置的边，并实现不同情况的功能；

- 设计 toString()方法，使用 Collections 类的 stream()方法和 Edge<String>类中设计的 toString()方法转化并输出字符串；
- 设计对 Graph<String>和 Edge<String>类中的方法的测试策略和测试用例，并保证具有高覆盖度，详见 test strategy。
- JUnit 测试结果截图：



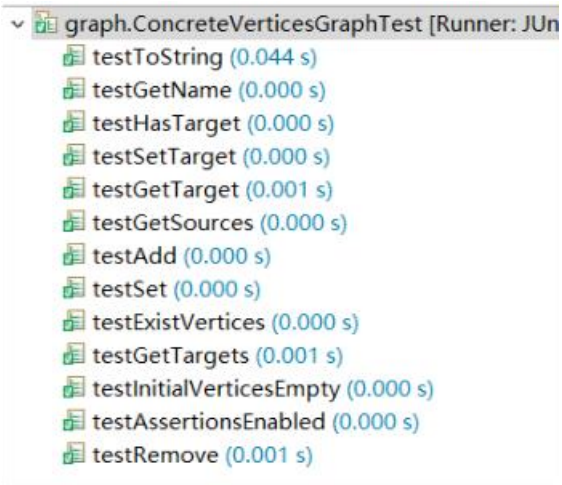
- 覆盖度截图：

Element	Covera...	Covered Inst...	Missed Instr...	Total Instruct...
graph	51.2 %	377	360	737
ConcreteVerticesGraphTes	0.0 %	0	184	184
GraphStaticTest.java	0.0 %	0	169	169
GraphInstanceTest.java	97.5 %	270	7	277
ConcreteEdgesGraphTest.ja	100.0 %	107	0	107

3.1.3.2 Implement ConcreteVerticesGraph

- 该图类给出的对象只有 vertices，适用邻接表实现图的功能；
- 根据 MIT 上对 Graph<L>的各种方法的说明完成各方法的功能；
- 设计 Vertex<String>类，类中对象应包含顶点名称以及顶点的边表，为防止 rep exposure 应都为 private final，需要实现获取顶点名、判断边表中是否存在指定顶点、设置边权值、输出边表等的方法，以便 Graph<String>方法的实现；
- 设计 set()方法，使用 Vertex<String>类中设计的 hasTarget()方法判断待设置边是否已经存在，以实现不同情况的功能，并使用 Vertex<String>类中设计的 set()方法设置边权值；
- 设计 toString()方法，使用 Collections 类的 stream()方法和 Vertex<String>类中设计的 toString()方法转化并输出字符串。
- 设计对 Graph<String>和 Vertex<String>类中的方法的测试策略和测试用例，并保证具有高覆盖度，详见 test strategy。

- JUnit 测试结果截图：



- 覆盖度截图：

Element	Covera...	Covered Inst...	Missed Instr...	Total Instruct...
graph	61.6 %	454	283	737
GraphStaticTest.java	0.0 %	0	169	169
ConcreteEdgesGraphTest.ja	0.0 %	0	107	107
GraphInstanceTest.java	97.5 %	270	7	277
ConcreteVerticesGraphTes	100.0 %	184	0	184

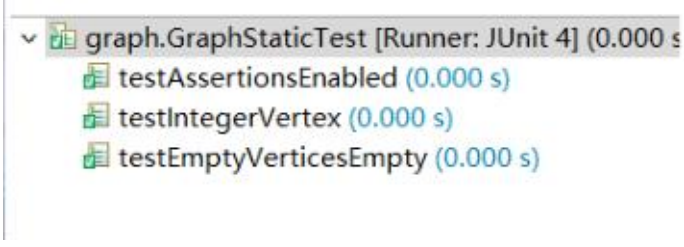
3.1.4 Problem 3: Implement generic `Graph<L>`

3.1.4.1 Make the implementations generic

- 将 3.1.3 中的除了 toString 部分外的 String 类型的对象都改成 L 类型。
- 继承 Graph<L>类时，应根据 L 表示的具体数据类型重写 toString()、equals()、hashCode()方法。

3.1.4.2 Implement `Graph.empty()`

- 调用 Graph.empty()生成 ConcreteEdgesGraph 或 ConcreteVerticesGraph，无论生成的是何种图，用户在进行操作的过程和获得的结果应该都是一致的；
- 设计 L 为 Integer 的功能测试，对两种不同的 Graph.empty()进行测试，测试结果应一致且通过。
- JUnit 测试结果：



- 覆盖度：

ConcreteEdgesGraph:

ConcreteVerticesGraph.java	0.0 %	0	426	426
ConcreteEdgesGraph.java	82.4 %	272	58	330
Graph.java	100.0 %	4	0	4
poet	0.0 %	0	218	218
test/P1	20.3 %	164	644	808
graph	22.3 %	164	573	737
GraphInstanceTest.java	0.0 %	0	277	277
ConcreteVerticesGraphTes	0.0 %	0	184	184
ConcreteEdgesGraphTest.ja	0.0 %	0	107	107
GraphStaticTest.java	97.0 %	164	5	169

ConcreteVerticesGraph:

ConcreteVerticesGraph.java	76.5 %	326	100	426
Graph.java	100.0 %	4	0	4
poet	0.0 %	0	218	218
test/P1	20.3 %	164	644	808
graph	22.3 %	164	573	737
GraphInstanceTest.java	0.0 %	0	277	277
ConcreteVerticesGraphTes	0.0 %	0	184	184
ConcreteEdgesGraphTest.ja	0.0 %	0	107	107
GraphStaticTest.java	97.0 %	164	5	169

3.1.5 Problem 4: Poetic walks

3.1.5.1 Test GraphPoet

- 测试从文件按词获取字符串的功能、poem()方法、以及 toString()方法，详见 test strategy。
- JUnit 测试结果（完成 GraphPoet 后）：

- 覆盖度：

poet	95.8 %	68	3	71
GraphPoetTest.java	95.8 %	68	3	71
src/P1	40.8 %	399	579	978
graph	27.9 %	212	548	760
poet	85.8 %	187	31	218
Main.java	0.0 %	0	25	25
GraphPoet.java	96.9 %	187	6	193

3.1.5.2 Implement GraphPoet

- 设计 getWordsFromFile()，使用 Scanner 类按词读取文件；
- 设计 getCorpusWords()，获取单词表；
- 设计 connectWords()，每两个相邻单词之间形成一条边；
- 设计 poem()，根据输入，查找单词表，若两个单词之间空了一个单词，则插入该单词。使用 Gaph 类的 getTargets()方法实现查找两个单词之间是否空有单词，使用 Collections 类的 stream().collect()方法输出字符串；
- 设计 toString，按顺序输出单词表。

3.1.5.3 Graph poetry slam

I didn't do this..

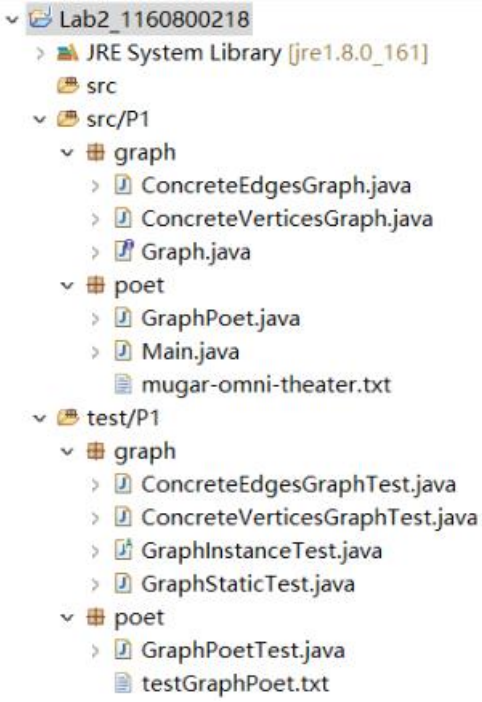
3.1.6 Before you're done

请按照 http://web.mit.edu/6.031/www/sp17/psets/ps2/#before_youre_done 的说明，检查你的程序。

如何通过 Git 提交当前版本到 GitHub 上你的 Lab2 仓库。

```
git push origin master
```

在这里给出你的项目的目录结构树状示意图。



3.2 Re-implement the Social Network in Lab1

该实验要求我们使用前一个问题完成的 Graph<L>泛型实现 Lab1 中的 Social Network，只改变结点的类型和部分方法的实现方案，确保功能不变。

3.2.1 FriendshipGraph 类

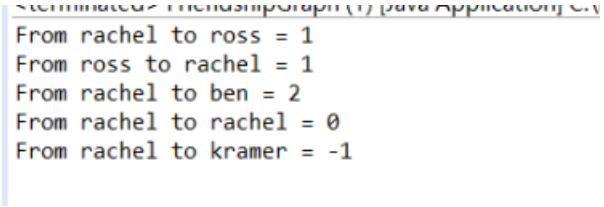
- 使用 ConcreteEdgesGraph<Person>类实现图，顶点为 Person 类型；
- Social Network 中的对象存于图的 vertices 集合中；
- addVertex()：检查 vertices 中是否含有待加入结点的名字，若已存在，则输出错误提醒并结束程序，若不存在，将结点加入 vertices 中，使用 Graph 的 add()方法实现；
- addEdge()：检查 edges 集合中是否存在待加入的边，使用 Graph 中的 set()方法实现；
- getDistance()：首先判断起点结点与目标结点是否相同，相同则返回 0，不同则执行下一步。创建队列， bfs 按顺序将结点入队并将队首结点出队，设置该结点的 isVisit 值为为 true，调用 getTargets()方法获得该结点的目标结点表，遍历是否存在目标结点，若不存在，则 distance 加一，若存在，则返回（distance + 1）。若在遍历结束后没有返回，则说明起点结点与目标结点直接无连接，返回-1。需要注意的是在每一次有返回值之前要设置遍历过的结点的 isVisit 值为 false 以便下一次调用 getDistance()函数不会受到本次调用的影响；
- 将双向图变为单向图后，测试用例结果符合预期。

3.2.2 Person 类

- 设置对象属性有：personname、isVisit（是否被访问，在 getDistance()用到）；
- 设计 getName()方法，输出结点名称。

3.2.3 客户端 main()

- 在实验要求上的例子基础上添加了单向连线和多路径选最短路径的测试用例。程序执行结果符合设想。

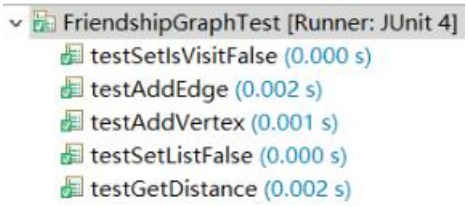


3.2.4 测试用例

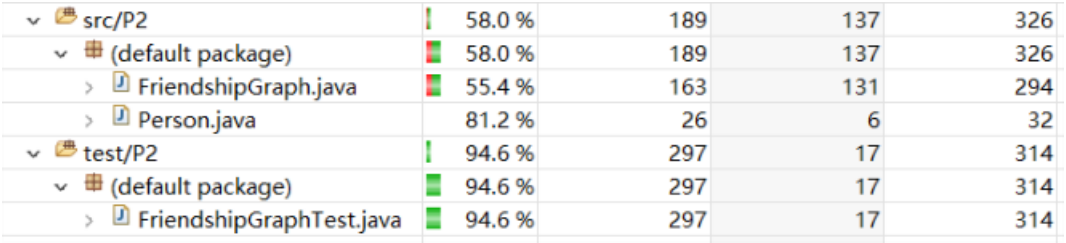
- 与 Lab1 测试用例基本相同，详见 test strategy；
- 在 testAddVertex()中对添加重名的顶点进行测试时会遇到 system.exit(0)而强行终止程

序导致测试不成功，在查阅相关资料后，增加捕捉并屏蔽 system.exit(0)发出的信号。

- JUnit 测试结果：



- 覆盖度：(FriendshipGraph 中的 main 方法未设计测试用例)

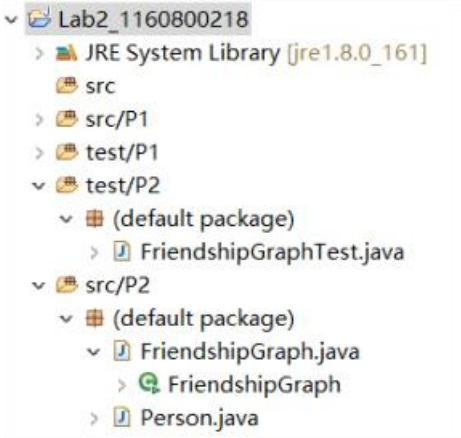


3.2.5 提交至 Git 仓库

如何通过 Git 提交当前版本到 GitHub 上你的 Lab3 仓库。

```
git push origin master
```

在这里给出你的项目的目录结构树状示意图。



3.3 The Transit Route Planner（选做，额外给分）

未完成。

4 实验进度记录

请尽可能详细的记录你的进度情况。

日期	时间段	计划任务	实际完成情况
2018.3.24	14:00-17:30	完成 ConcreteEdgesGraph 和 ConcreteVerticesGraph 方法	未完成
2018.4.1	10:00-15:00	完成 ConcreteEdgesGraph 和 ConcreteVerticesGraph 方法	按时完成
2018.4.2	9:00-12:00	完成 AF、RI、safety of RE、checkRep	按时完成
2018.4.3	8:00-12:00	完成 toString 和 testToStrin 以及两个图类的测试用例	延时完成
2018.4.6	9:00-12:00 14:00-18:00	完成 GraphPoet、GraphStaticTest、GraphInstanceTest 、P2，	按时完成
2018.4.7	19:00-22:00	完成 test strategy	按时完成
2018.4.8	8:30-12:00	完成实验报告	延时完成

5 实验过程中遇到的困难与解决途径

MIT 上的实验指导书为全英文，一开始理解起开困难且有许多偏差，在前两周实验进度拖沓不前。后在查阅老师的课程 ppt 和请教同学后逐渐对实验内容有了较为清晰的认识。在实现 Graph<L>泛型的过程中，遇到许多困难，有遍历错误、条件判断失效、不熟悉数据结构导致的错误、specification 编写不熟练等问题，后都通过查阅博客、资料和询问同学一一解决。

6 实验过程中收获的经验、教训、感想

本节除了总结你在实验过程中收获的经验和教训，也可就以下方面谈谈你的感受（非必须）：

(1) 面向 ADT 的编程和直接面向应用场景编程，你体会到二者有何差异？
面向 ADT 的编程更注重代码的普适性，直接面向应用场景编程更注重具体功能的实现。

(2) 使用泛型和不使用泛型的编程，对你来说有何差异？
使用泛型，代码结构性更强，更易读。

(3) 在给出 ADT 的规约后就开始编写测试用例，优势是什么？你是否能够适

应这种测试方式？

目的性指导性较强，还不太适应。

(4) 本实验设计的 ADT 在三个应用场景下使用，这种复用带来什么好处？

减少代码的复杂性，提高代码的可重用性。

(5) 为 ADT 撰写 specification, invariants, RI, AF，时刻注意 ADT 是否有 rep exposure，这些工作的意义是什么？你是否愿意在以后编程中坚持这么做？

(6) 关于本实验的工作量、难度、deadline。

工作量较大，难度适中，deadline 适当