

**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](mailto:1160800218@stu.hit.edu.cn) |
| 手机号码 | 15636703603 |

**目录**

[1 实验目标概述 1](#_Toc507927440)

[2 实验环境配置 1](#_Toc507927441)

[3 实验过程 2](#_Toc507927442)

[3.1 Poetic Walks 2](#_Toc507927443)

[3.1.1 Get the code and prepare Git repository 2](#_Toc507927444)

[3.1.2 Problem 1: Test Graph <String> 2](#_Toc507927445)

[3.1.3 Problem 2: Implement Graph <String> 2](#_Toc507927446)

[3.1.3.1 Implement ConcreteEdgesGraph 2](#_Toc507927447)

[3.1.3.2 Implement ConcreteVerticesGraph 3](#_Toc507927448)

[3.1.4 Problem 3: Implement generic Graph<L> 4](#_Toc507927449)

[3.1.4.1 Make the implementations generic 4](#_Toc507927450)

[3.1.4.2 Implement Graph.empty() 4](#_Toc507927451)

[3.1.5 Problem 4: Poetic walks 5](#_Toc507927452)

[3.1.5.1 Test GraphPoet 5](#_Toc507927453)

[3.1.5.2 Implement GraphPoet 6](#_Toc507927454)

[3.1.5.3 Graph poetry slam 6](#_Toc507927455)

[3.1.6 Before you’re done 6](#_Toc507927456)

[3.2 Re-implement the Social Network in Lab1 7](#_Toc507927457)

[3.2.1 FriendshipGraph类 7](#_Toc507927458)

[3.2.2 Person类 7](#_Toc507927459)

[3.2.3 客户端main() 7](#_Toc507927460)

[3.2.4 测试用例 7](#_Toc507927461)

[3.2.5 提交至Git仓库 8](#_Toc507927462)

[3.3 The Transit Route Planner（选做，额外给分） 8](#_Toc507927463)

[4 实验进度记录 9](#_Toc507927464)

[5 实验过程中遇到的困难与解决途径 9](#_Toc507927465)

[6 实验过程中收获的经验、教训、感想 9](#_Toc507927466)

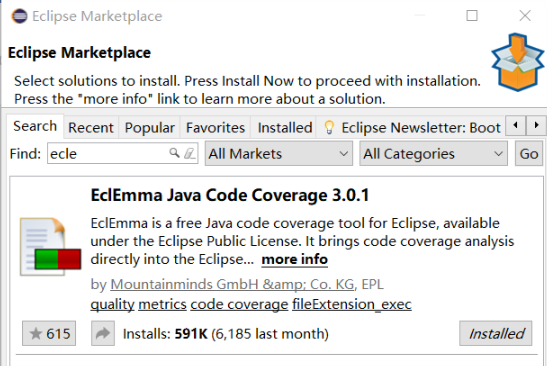
# 实验目标概述

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。

# 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

# 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！）。

## 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。

### 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。

### Problem 1: Test Graph <String>

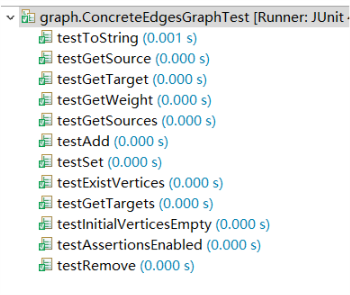
以下各部分，请按照MIT页面上相应部分的要求，逐项列出你的设计和实现思路/过程/结果。

### Problem 2: Implement Graph <String>

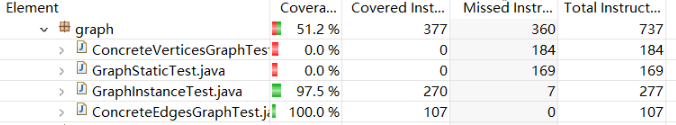
以下各部分，请按照MIT页面上相应部分的要求，逐项列出你的设计和实现思路/过程/结果。

#### 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：

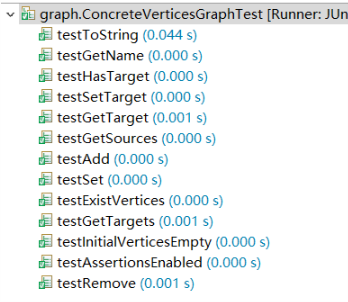


* 覆盖度截图：

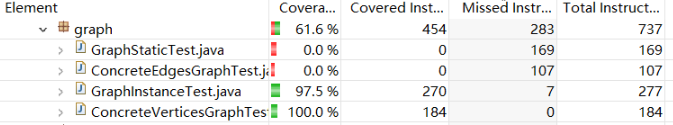


#### Implement ConcreteVerticesGraph

* 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：



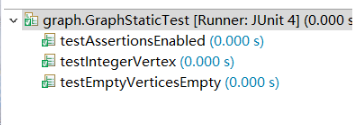
### Problem 3: Implement generic Graph<L>

#### 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。

#### Implement Graph.empty()

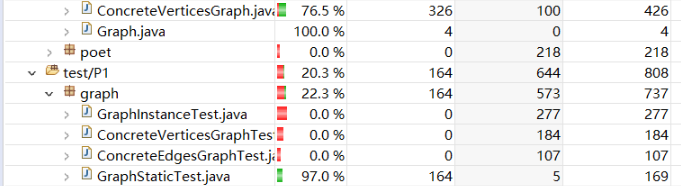
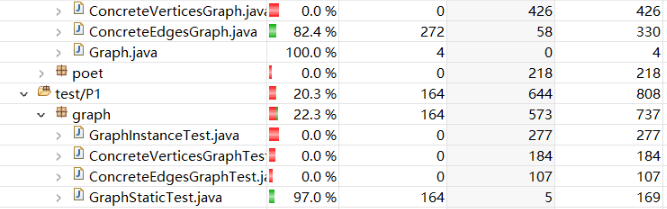
* 调用Graph.empty()生成ConcreteEdgesGraph或ConcreteVerticesGraph，无论生成的是何种图，用户在进行操作的过程和获得的结果应该都是一致的；
* 设计L为Integer的功能测试，对两种不同的Graph.empty()进行测试，测试结果应一致且通过。
* JUnit测试结果：



* 覆盖度：

ConcreteEdgesGraph:

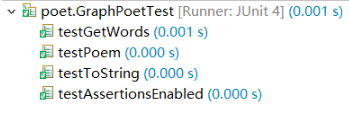
ConcreteVerticesGraph:



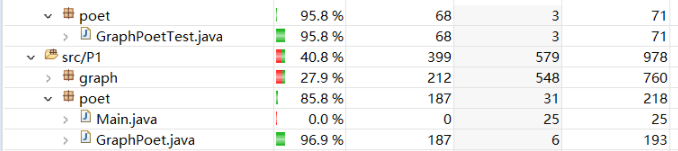
### Problem 4: Poetic walks

#### 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：



#### 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。

#### Graph poetry slam

I didn’t do this..

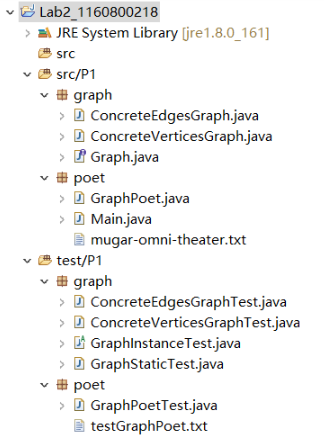
### Before you’re done

请按照[http://web.mit.edu/6.031/www/sp17/psets/ps2/#before\_youre\_done](http://web.mit.edu/6.031/www/sp17/psets/ps2/" \l "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。



## 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。

### FriendshipGraph类

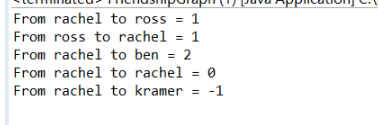
* 使用ConcreteEdgesGraph<Person>类实现图，顶点为Person类型；
* Social Network中的对象存于图的vertices集合中；
* addVertex()：检查vertices中是否含有待加入结点的名字，若已存在，则输出错误提醒并结束程序，若不存在，将结点加入vertices中，使用Graph的add()方法实现；
* addEdge()：检查edges集合中是否存在待加入的边，使用Graph中的set()方法实现；
* getDistance()：首先判断起点结点与目标结点是否相同，相同则返回0，不同则执行下一步。创建队列， bfs按顺序将结点入队并将队首结点出队，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。

### 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。

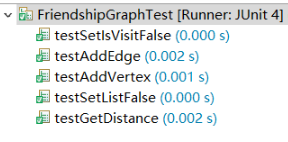
### 客户端main()

* 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。

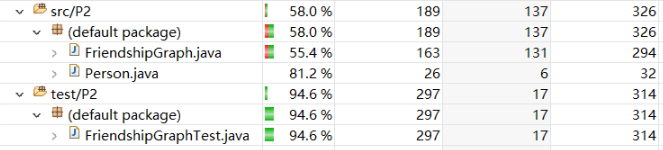


### 测试用例

* 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）

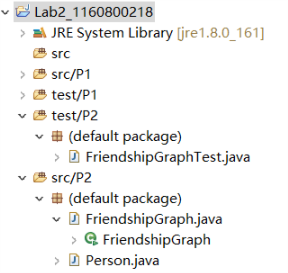


### 提交至Git仓库

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。



## The Transit Route Planner（选做，额外给分）

未完成。

# 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 |

# 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。

# 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。

1. 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。

1. 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。

1. 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。

1. 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？
2. 关于本实验的工作量、难度、deadline。

工作量较大，难度适中，deadline适当