

**2021年春季学期  
计算学部《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 沈城有 |
| 学号 | 1190200526 |
| 班号 |  |
| 电子邮件 |  |
| 手机号码 |  |

**目录**

[1 实验目标概述 1](#_Toc73898050)

[2 实验环境配置 1](#_Toc73898051)

[3 实验过程 2](#_Toc73898052)

[3.1 Poetic Walks 2](#_Toc73898053)

[3.1.1 Get the code and prepare Git repository 2](#_Toc73898054)

[3.1.2 Problem 1: Test Graph <String> 3](#_Toc73898055)

[3.1.3 Problem 2: Implement Graph <String> 3](#_Toc73898056)

[3.1.3.1 Implement ConcreteEdgesGraph 3](#_Toc73898057)

[3.1.3.2 Implement ConcreteVerticesGraph 6](#_Toc73898058)

[3.1.4 Problem 3: Implement generic Graph<L> 9](#_Toc73898059)

[3.1.4.1 Make the implementations generic 9](#_Toc73898060)

[3.1.4.2 Implement Graph.empty() 9](#_Toc73898061)

[3.1.5 Problem 4: Poetic walks 10](#_Toc73898062)

[3.1.5.1 Test GraphPoet 10](#_Toc73898063)

[3.1.5.2 Implement GraphPoet 10](#_Toc73898064)

[3.1.5.3 Graph poetry slam 12](#_Toc73898065)

[3.1.6 使用Eclemma检查测试的代码覆盖度 12](#_Toc73898066)

[3.1.7 Before you’re done 13](#_Toc73898067)

[3.2 Re-implement the Social Network in Lab1 14](#_Toc73898068)

[3.2.1 FriendshipGraph类 14](#_Toc73898069)

[3.2.2 Person类 16](#_Toc73898070)

[3.2.3 客户端main() 17](#_Toc73898071)

[3.2.4 测试用例 17](#_Toc73898072)

[3.2.5 提交至Git仓库 17](#_Toc73898073)

[4 实验进度记录 19](#_Toc73898074)

[5 实验过程中遇到的困难与解决途径 19](#_Toc73898075)

[6 实验过程中收获的经验、教训、感想 19](#_Toc73898076)

[6.1 实验过程中收获的经验和教训 19](#_Toc73898077)

[6.2 针对以下方面的感受 20](#_Toc73898078)

# 实验目标概述

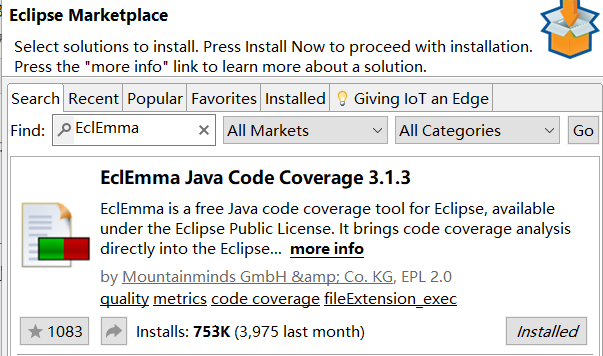
本次实验训练抽象数据类型（ADT）的设计、规约、测试，并使用面向对象编程（OOP）技术实现 ADT。具体来说：

* 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泄露（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。

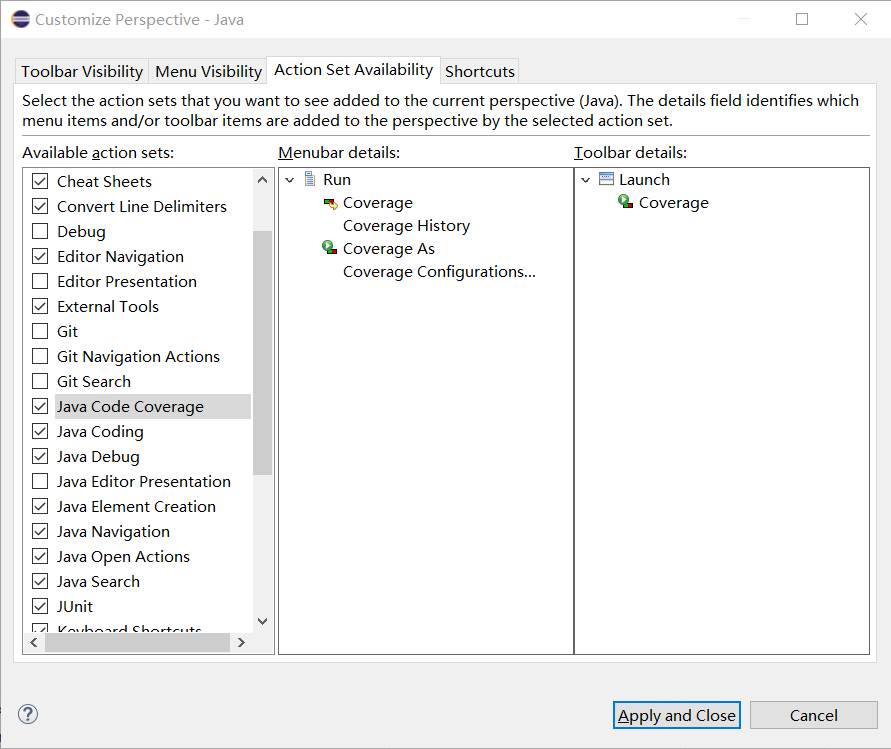
# 实验环境配置

本次实验需要在Eclipse IDE中配置安装EclEmma，我通过浏览EclEmma官网找到了安装教程。

首先在Eclipse中找到Help-Eclipse Marketplace，在其中搜索EclEmma，结果如下图：



显示已经安装，但我没有在工具栏看到其图标。打开Window-Perspective-Customize Perspective，弹出一个页面并选择Action Set Availability，勾选Java Code Coverage（如下图）并保存设置，发现工具栏出现图标，配置完成。



GitHub Lab2仓库的URL地址：

<https://github.com/ComputerScienceHIT/HIT-Lab2-1190200526>

# 实验过程

## Poetic Walks

该任务要求首先实现并测试Graph<String>，其包括边图、点图两种具体实现，随后从具体扩展到泛型Graph<L>，最后使用编写的ADT完成简单的应用。

### Get the code and prepare Git repository

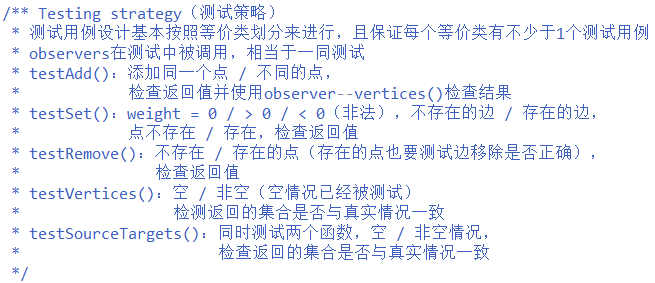
我首先通过git clone命令（SSH方式）将GitHub上用于实验提交的代码库克隆至本地，这同时也完成了本地库与远程库的关联。随后使用GitHub网页端下载了包含任务代码的ZIP包，调整结构后放入本地git仓库。最后使用Eclipse IDE在此目录创建新工程，调整工程buildpath及部分import后将初始化的项目推送至GitHub。

注：此步完成后的项目结构见GitHub描述为“Lab2 init.”的提交。

### Problem 1: Test Graph <String>

要求我们针对Graph<String>设计和实现对应的测试并写明测试策略，规定在GraphInstanceTest中的测试用例必须是符合Graph规约的客户端代码。

我的思路是按照等价类划分情况，并针对每个情况设置至少一个测试用例，具体测试策略如下：



注：保留了testInitialVerticesEmpty()用于测试testVertices()未测试的空情况。

测试代码较长，此处略。

### Problem 2: Implement Graph <String>

此部分要求重写Graph里的方法，分别实现以点为基础、以边为基础的图。

#### Implement ConcreteEdgesGraph

要求使用给定的私有属性完成ConcreteEdgesGraph的编程，且Edge类要求必须是不可变(immutable)的。

首先设计实现Edge类，过程如下：

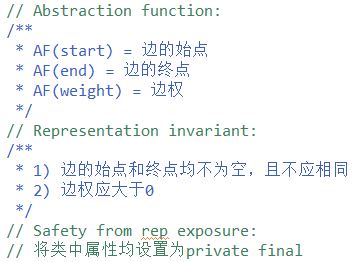
1. Edge类应保存有向边的始点、终点和边权信息，故定义以下私有属性：



1. 需要实现的方法如下：

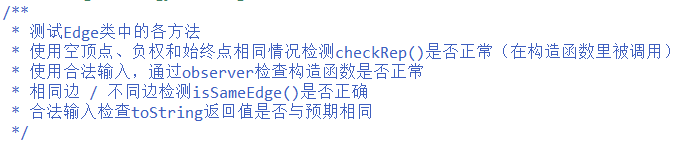
* Edge()：构造方法
* checkRep()：检查表示不变性
* getStart()、getEnd()、getWeight()：获取边信息（始点、终点、边权）
* isSameEdge()：判断两边是否为同一条（用于辅助图实现，简单比较两边的始、终顶点是否分别相同）
* toString()：返回有向边的可读表示

1. AF、RI和Safety from rep exposure：



1. 设计Edge测试策略并编写测试：

测试策略：



具体测试此处略去。

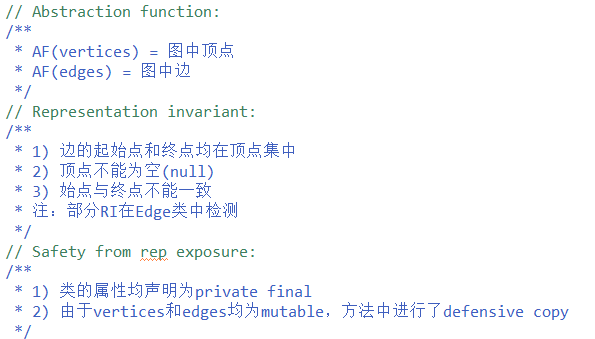
1. 实现Edge类（此处略，见源代码文件ConcreteEdgesGraph.java）。

然后设计实现ConcreteEdgesGraph类，具体过程如下：

1. 使用实验要求的私有属性：



1. AF、RI与Safety from rep exposure：

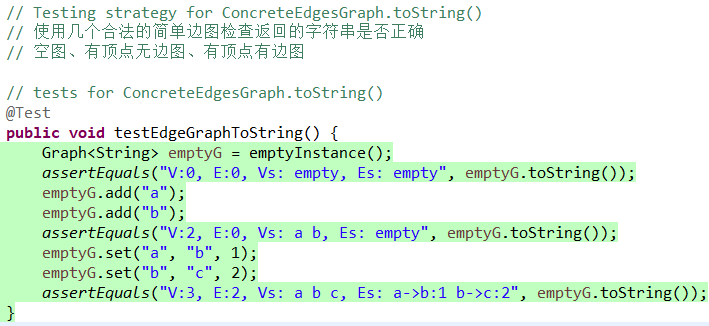


1. 实现各方法，主要思路：、

除toString()方法外，均根据AF、RI、Safety from rep exposure及Graph.java文件中的规约来实现具体的方法，在源代码文件ConcreteEdgesGraph.java中有较详细的注释说明，故代码此处略去；对于toString()方法，我的思路为输出图顶点数、边数，然后输出每条边的信息（调用Edge类的toString方法，并连接字符串），具体实现见下图：

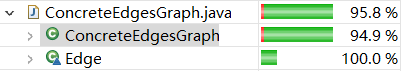


1. 编写测试策略，设计测试用例：由于其他方法的测试在Problem 1中已经进行了编写，此处仅编写toString()方法的测试策略并实现其测试，如下图：

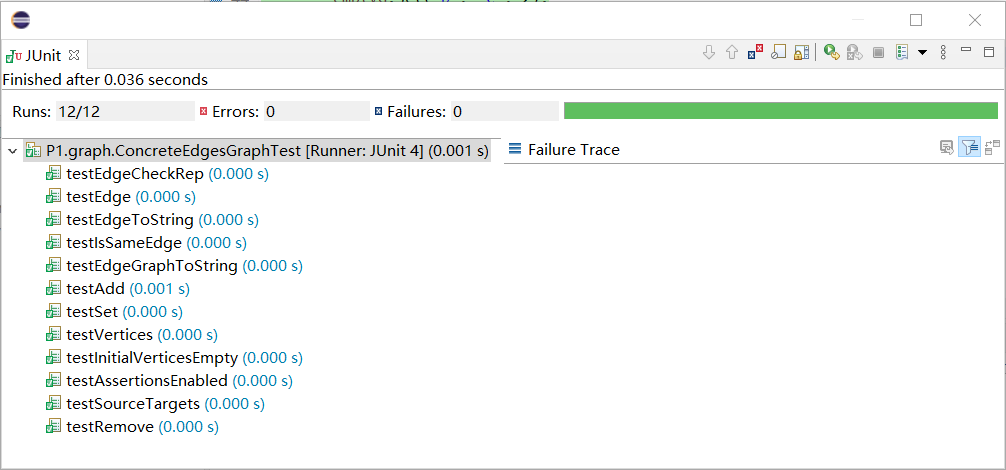


最后运行测试，得到的结果如下图：

代码覆盖率情况：



测试用例通过情况：

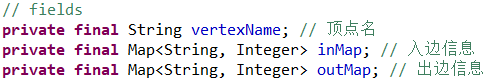


#### Implement ConcreteVerticesGraph

要求使用给定的私有属性完成ConcreteVerticesGraph的编程，且Vertex类要求必须是可变(mutable)的。

首先设计Vertex类，过程如下：

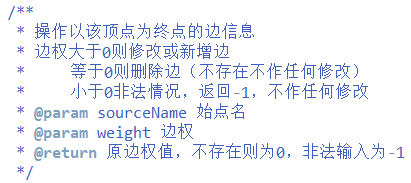
1. Vertex类应保存顶点及顶点间边关系的相关信息，故定义以下私有属性：

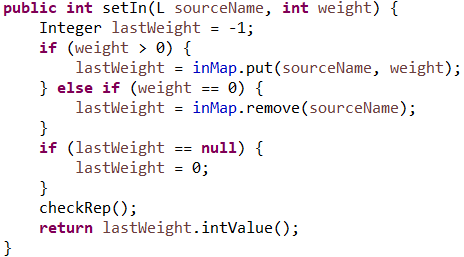


1. 需要实现的方法如下：

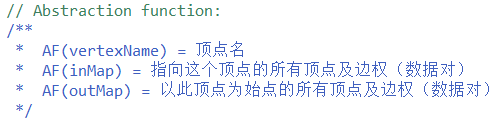
* Vertex()：构造方法
* checkRep()：检查不变性
* getName()、getIns()、getOuts()：访问私有属性信息
* setIn()、setOut()：设置顶点入边/出边信息（整合了新增、修改与删除），用于辅助ConcreteVerticesGraph类set()、remove()等方法的实现
* toString()：返回Vertex类对象可读表示

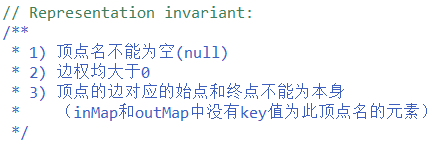
这里展示较关键的方法setIn()，setOut()与其类似：

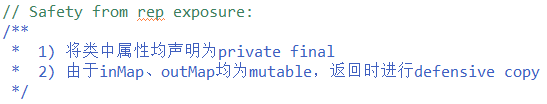




1. AF、RI和Safety from rep exposure：

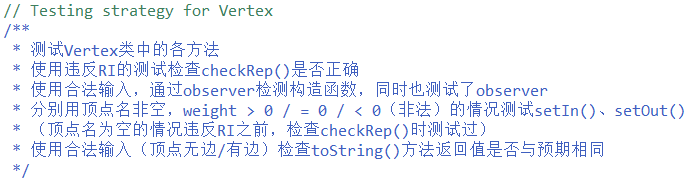






1. 设计Vertex测试策略并编写测试：

测试策略：



具体测试代码略。

1. 按照设计实现Vertex类：

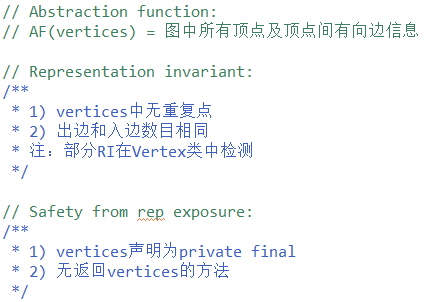
见源代码文件。

随后设计实现ConcreteVerticesGraph类，具体过程如下：

1. 使用实验要求的私有属性：



1. AF、RI与Safety from rep exposure：



1. 实现各方法，主要思路：除toString()方法外，均根据AF、RI、Safety from rep exposure及Graph.java文件中的规约来实现具体的方法，在源代码文件ConcreteVerticesGraph.java中有较详细的注释说明，对于toString()方法，实现与ConcreteEdgesGraph中类似。故代码此处略去；
2. 编写测试策略，设计测试用例：

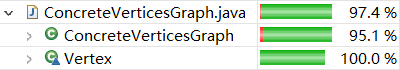
测试策略如下（与ConcreteEdgesGraph类似）：



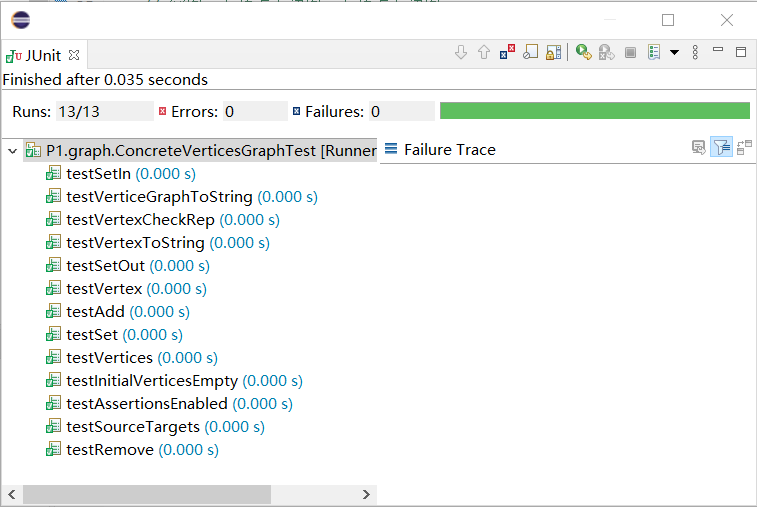
具体测试代码略。

最后运行测试，得到的结果如下图：

代码覆盖率情况：



测试用例通过情况：



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

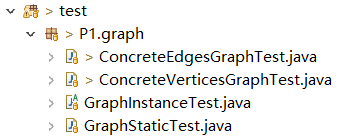
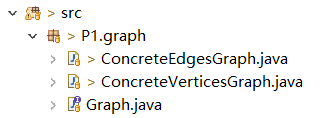
#### Make the implementations generic

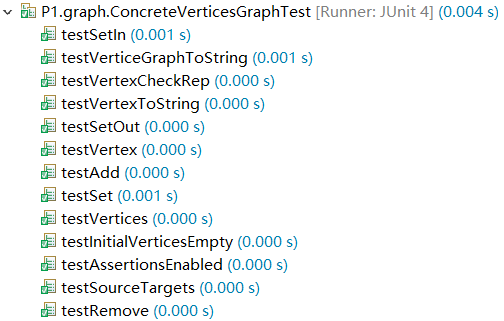
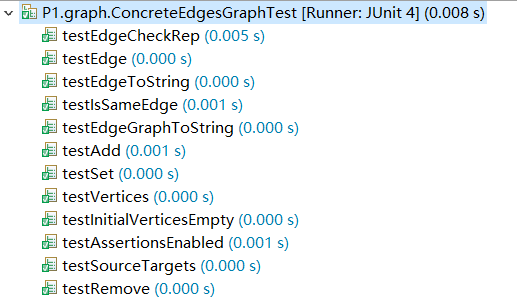
要求将之前实现的ConcreteEdgesGraph、ConcreteVerticesGraph及相关的Edge类和Vertice类转换为泛型实现。

按照实验要求，先修改几个类的声明，随后可按照Eclipse IDE的错误提示进行对应的修改。

首先修改声明和私有属性，随后按照错误提示修改类内部的类型，最后按照警告提示修改对应的测试文件，具体过程略去，可查看GitHub仓库中描述为“P1 Problem 3.1 completed.”的提交中的代码变化。

完成全部修改后，所有相关源代码文件均不再显示警告或错误，且能通过测试，如下图：





#### Implement Graph.empty()

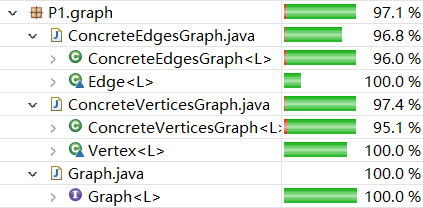
这里我选择ConcreteEdgesGraph供客户端使用，首先实现Graph.empty()，如下图：



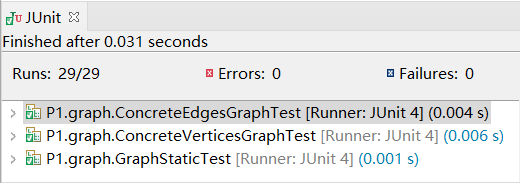
随后在GraphStaticTest.java中补充对不同immutable类型L的测试，这里我添加了对Integer和Character（分别是int和char的封装对象类型）的测试。具体测试代码此处略，测试策略类似于简化版的GraphInstanceTest，初始化相应类型的对象并进行了一些简单的方法操作，检测结果是否正确。

最后运行测试，结果如下图：

代码覆盖率情况：



测试用例通过情况：



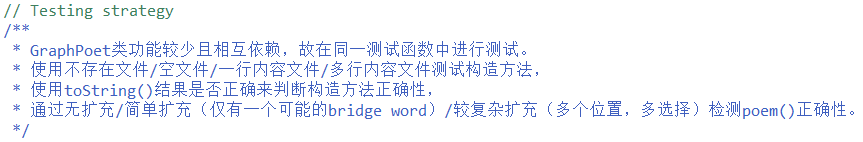
### Problem 4: Poetic walks

该任务要求使用Graph类来实现GraphPoet类，此类使用读入的语料库(corpus)初始化图结构，并运用这一图结构搜索词之间的关系，自动扩充输入的语句（使之诗歌化）。

#### Test GraphPoet

这一步要求在GraphPoetTest.java中设计对于GraphPoet类的测试策略并编写测试用例。允许在GraphPoet类中添加额外的方法及加强所需方法的规约，但不能修改所要求的方法的签名或减弱其规约强度。

测试策略如下：



具体测试代码略。

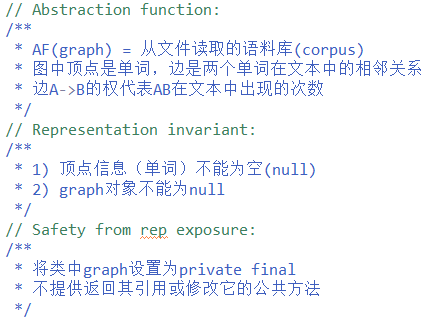
#### Implement GraphPoet

这一步要求实现GraphPoet类，我按照实验要求实现了类中的各方法，没有修改原来的规约，也没有增加新的方法。具体过程如下：

1. 使用要求的私有属性：



1. AF、RI及Safety from rep exposure

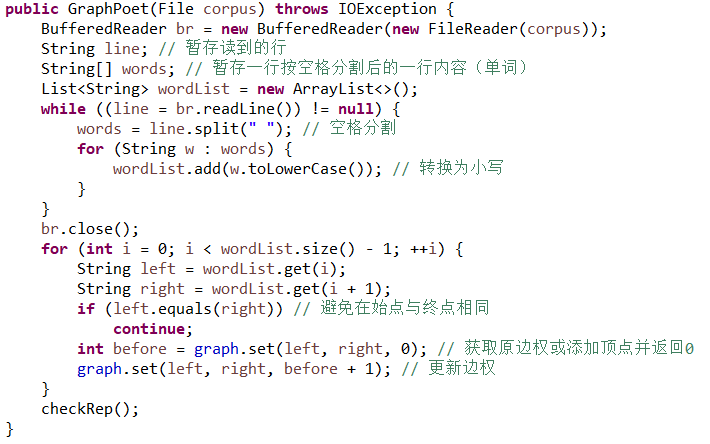


1. 实现各方法：

需实现的方法及简要思路：

* GraphPoet()：构造函数，读取文件，构建图结构。文件操作与Lab1类似，使用String示例的split方法得到单词，将单词转换为小写表示，并对每对单词调用两次图结构的set方法来分别获取原权值和更新权值。**一个关键点是两个单词不能相等，完成编程后测试时才发现这一问题，这样会违反图的RI从而导致程序异常终止。**

具体代码如下：



* checkRep()：检查不变性；
* poem()：生成新文本(poetry)，利用图结构的sources和targets方法，查找输入文本中每两个相邻单词间可能的bridge word，并在有多个可能的bridge word时选择最早出现、边权最大的一个。核心部分代码如下：

（见下页图）

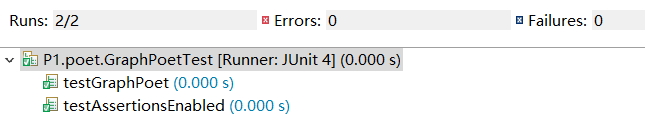


* toString()：将类信息转换为可读字符串返回，主要使用了graph类的toString方法。

注：具体实现见源代码文件GraphPoet.java。

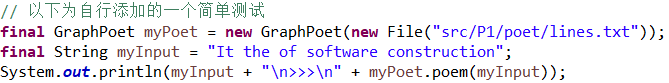
1. 进行测试：

测试结果如下：

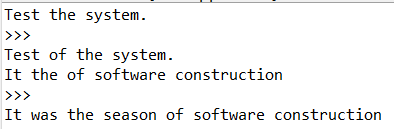


#### Graph poetry slam

在Main.java中添加了一个简单的例子，如下图：

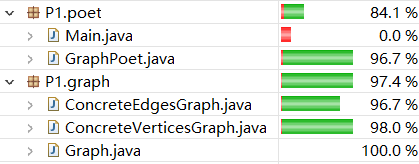


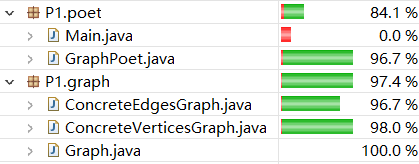
运行结果如下图所示：



### 使用Eclemma检查测试的代码覆盖度

运行P1部分的所有测试，代码覆盖度结果如下：



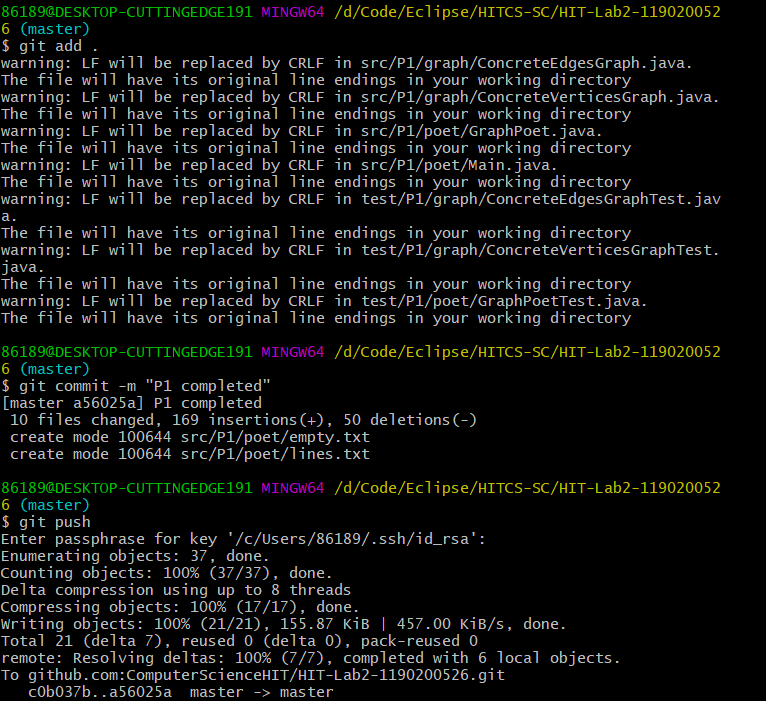


注：Main.java是客户端代码，不进行JUnit测试，可直接运行输出结果。

### Before you’re done

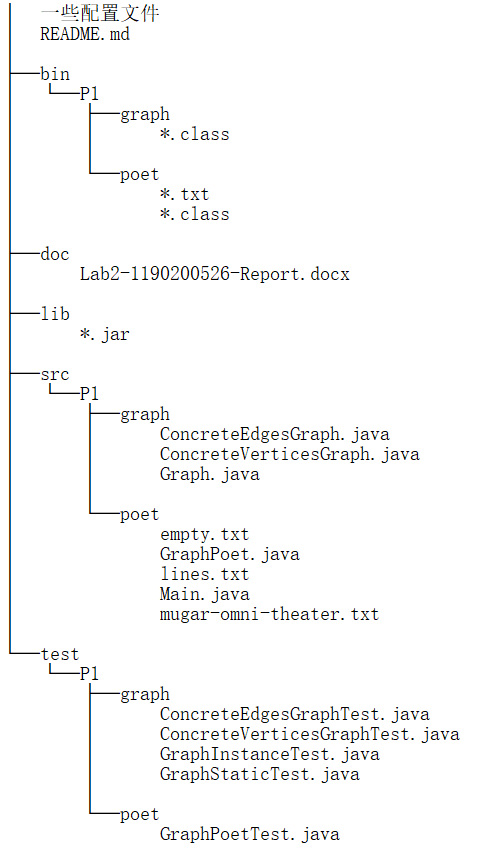
请按照[http://web.mit.edu/6.031/www/sp17/psets/ps2/#before\_youre\_done](http://web.mit.edu/6.031/www/sp17/psets/ps2/" \l "before_youre_done)的说明，检查你的程序：2021.6.1 完成检查及相应的代码修改完善工作。

如何通过Git提交当前版本到GitHub上你的Lab2仓库：



在这里给出你的项目的目录结构树状示意图。

(见下页)



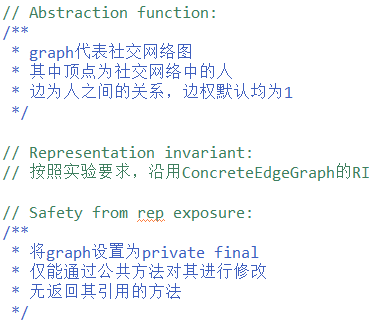
## Re-implement the Social Network in Lab1

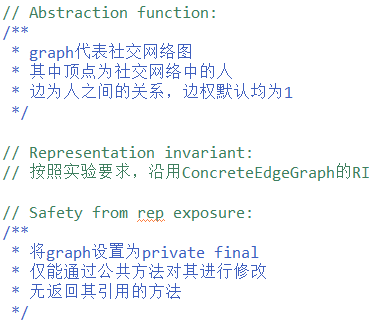
该任务要求基于在3.1节Poetic Walks中定义的Graph<L>及其两种实现（这里的L即为Person），重新实现Lab1中3.3节的FriendshipGraph类，并通过Lab1中编写的测试用例。

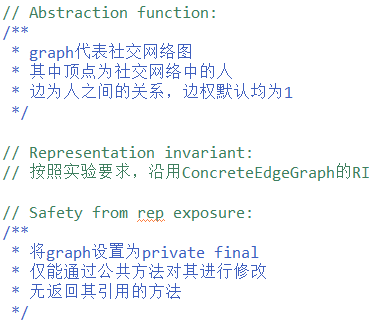
### FriendshipGraph类

我选择使用3.1中实现的边图来完成FriendshipGraph类的实现。

补充了AF、RI和Safety from rep exposure：

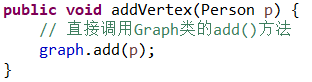




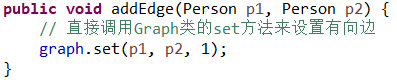


需要实现的方法的思路及具体实现如下：

1. addVertex()：



1. addEdge()：



1. getDistance()：

使用与Lab1实现相同的BFS思路，只需调整队列的元素类型、记录顶点是否访问过的方式和循环体，具体实现如下：



注：以上部分检查两点是否相同或有一点不在图中。

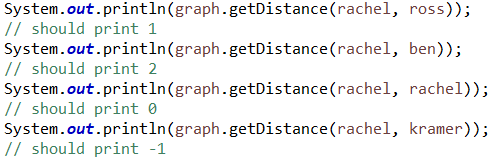


注：以上部分是该方法的核心，运用BFS求距离。



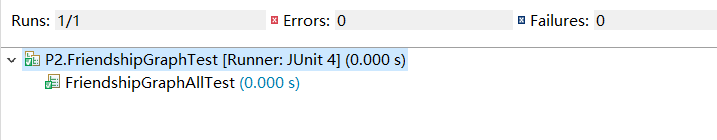
注：以上部分为结果的返回，若BFS不可达，disMap中则无对应记录，返回-1，若可达，输出disMap中记录的结果。

至此，要求的三个函数已经全部实现，运行客户端main()代码，结果与预期结果一致，如下图：



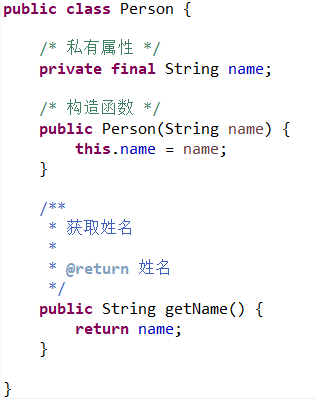


Junit测试用例通过，结果如下图：



### Person类

实现与Lab1相同，定义一个私有属性name保存人的姓名，构造函数初始化姓名，公共getName()方法获取姓名：



### 客户端main()

保留了Lab1中原有的客户端代码，运行结果与预期一致（见3.2.1节）。

### 测试用例

基本沿用了Lab1中的测试用例，但对于addVertex()和addEdge()方法，新实现不再抛出异常（在Graph类中进行了检测），故删去了对这些非法抛出异常的测试（根据Lab1中CMU的实验指导，对于违反spec的非法情况的处理可以自行决定，故这样的修改不会影响程序正确性）。

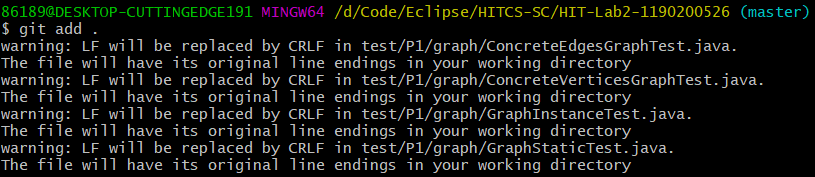
具体测试代码见FriendshipGraphTest.java。

### 提交至Git仓库

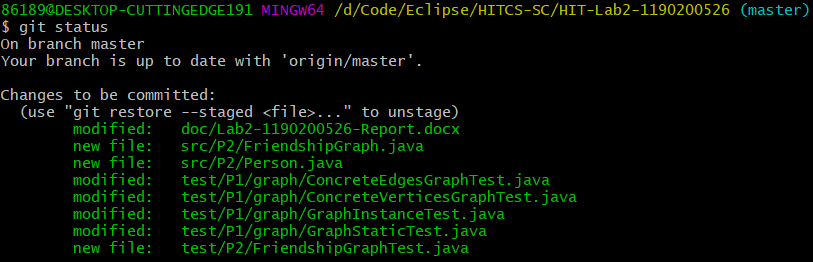
如何通过Git提交当前版本到GitHub上你的Lab3仓库。

过程与3.1.7节类似（git add . + git commit -m “描述” + git push），如下图：

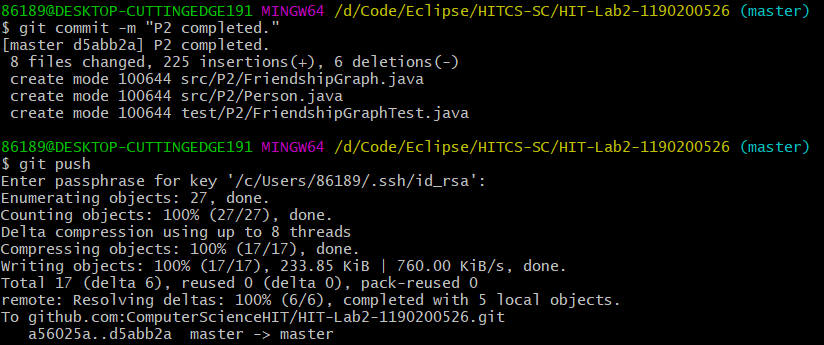
第一步：



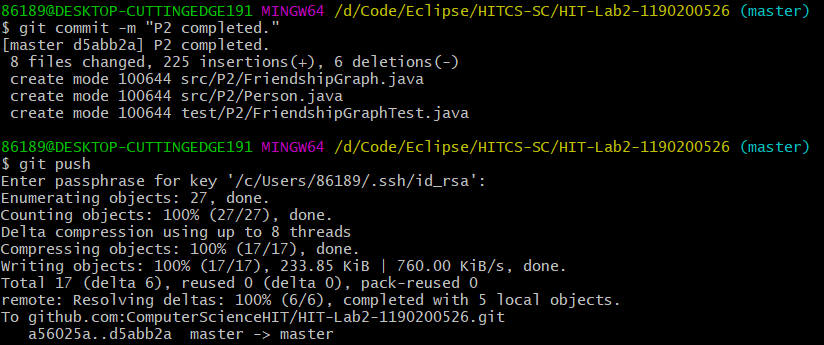
查看暂存情况：



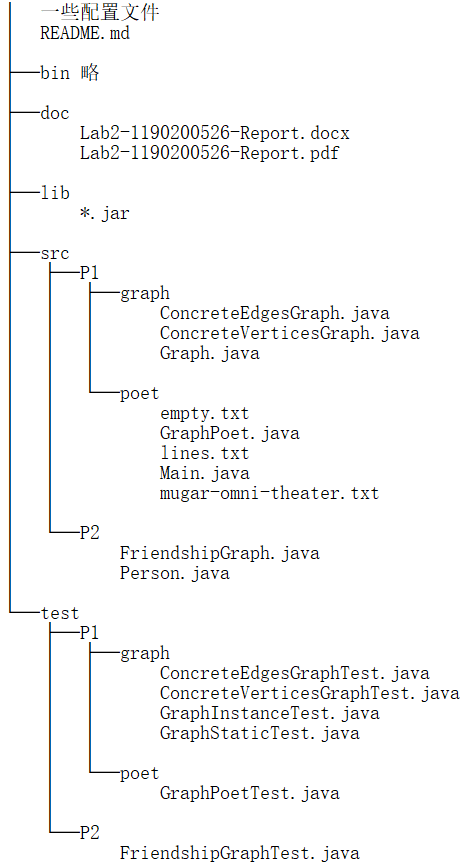
第二步：



第三步：



在这里给出你的项目的目录结构树状示意图：



# 实验进度记录

请使用表格方式记录你的进度情况，以超过半小时的连续编程时间为一行。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 2021.5.25 | 18:00 - 20:00 | 完成P1 Problem 1 | 按计划完成 |
| 2021.5.27 | 14:30 - 15:30 | 初步设计P1 ConcreteEdgesGraph | 按计划完成 |
| 2021.5.27 | 18:30 - 22:30 | 完成P1 ConcreteEdgesGraph的编程及对应部分的报告 | 按计划完成 |
| 2021.5.29 | 14:40 - 17:00 | 完成P1 ConcreteVerticesGraph的编程 | 按计划完成 |
| 2021.5.29 | 19:00 - 20:40 | 完成P1 Problem 2所有内容 | 按计划完成 |
| 2021.5.30 | 11:50 - 12:40 | 完成P1 Problem 3的3.1 | 按计划完成 |
| 2021.5.30 | 14:40 - 15:30 | 完成P1 Problem 3所有内容 | 按计划完成 |
| 2021.5.31 | 16:00 - 17:20 | 完成P1 Problem 4的4.1 | 按计划完成 |
| 2021.5.31 | 18:00 - 20:30 | 完成P1全部内容，完善代码 | 按计划完成 |
| 2021.6.1 | 15:00 - 17:00 | 修改P1部分代码，完善报告，完成P2 | 按计划完成 |
| 2021.6.1 | 17:50 - 18:30 | 总结实验，在线构建，提交 | 按计划完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 实验初期对AF、RI和表示泄露等概念理解不清，不知道如何说明和编写相关代码。 | 课堂听课、课后复习课件并与其他同学探讨。 |
| 不适应测试优先的编程方式，在没有具体实现前不知如何设计测试。 | 只能在完成实验的过程中逐渐适应。现在能做到根据规约设计测试用例，再进行具体实现，但测试有时还要进行调整和补充。 |
| 不了解如何设计实现checkRep()检查RI。 | 在实验课上请教老师。 |

# 实验过程中收获的经验、教训、感想

## 实验过程中收获的经验和教训

1. 深入理解了AF、RI、表示泄露等概念；
2. 理解了设计ADT、将ADT泛型化的过程；
3. 学会了如何避免表示泄露及如何检查表示不变性；
4. 认识到养成测试优先编程习惯的重要性和意义，锻炼了编写规约及根据规约设计测试用例的能力；
5. 学会了如何检查和提高代码覆盖度及如何使用ADT进行简单的OOP编程。

## 针对以下方面的感受

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？

答：面向ADT的编程更底层、更抽象、更通用化，不如直接面向应用场景编程具体，但ADT是面向应用场景编程的基础。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

答：总体上差异不是很大，但泛型更加灵活，有时也要考虑更多情况。

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？

答：优势是能增强编程的目的性，更好地保证ADT在编写过程中的正确性，减少错误和漏洞积累，便于及时修正和调整；个人不是很适应这种测试方式，会在今后的学习和实践中继续适应。

1. P1设计的ADT在多个应用场景下使用，这种复用带来什么好处？

答：减少重复开发的工作量，提高开发效率。

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？

答：意义是提高程序的正确性、健壮性和安全性，保证程序质量，避免编程的实际工作脱离目标，且便于用户使用；我愿意在以后编程中坚持这样做，因为这对于程序开发、维护和多人协作具有重要作用。

1. 关于本实验的工作量、难度、deadline。

答：工作量适中，难度适中，deadline较晚也保证了比较充足的时间。

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

答：收获了很多Java编程的相关知识，认识到编写程序与实际开发的巨大差异，学习到了很多实际开发软件的重要知识、方法和技能。建议增强课件及课堂讲解内容的逻辑性、层次性，便于同学们进行预习和复习。