

**2021年春季学期  
计算学部《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 林搏海 |
| 学号 | 1190202128 |
| 班号 | 1903002 |
| 电子邮件 | 1172798125@qq.com |
| 手机号码 | 13452053282 |

**目录**

[1 实验目标概述 1](#_Toc72249750)

[2 实验环境配置 1](#_Toc72249751)

[3 实验过程 1](#_Toc72249752)

[3.1 Poetic Walks 1](#_Toc72249753)

[3.1.1 Get the code and prepare Git repository 1](#_Toc72249754)

[3.1.2 Problem 1: Test Graph <String> 1](#_Toc72249755)

[3.1.3 Problem 2: Implement Graph <String> 1](#_Toc72249756)

[3.1.3.1 Implement ConcreteEdgesGraph 2](#_Toc72249757)

[3.1.3.2 Implement ConcreteVerticesGraph 2](#_Toc72249758)

[3.1.4 Problem 3: Implement generic Graph<L> 2](#_Toc72249759)

[3.1.4.1 Make the implementations generic 2](#_Toc72249760)

[3.1.4.2 Implement Graph.empty() 2](#_Toc72249761)

[3.1.5 Problem 4: Poetic walks 2](#_Toc72249762)

[3.1.5.1 Test GraphPoet 2](#_Toc72249763)

[3.1.5.2 Implement GraphPoet 2](#_Toc72249764)

[3.1.5.3 Graph poetry slam 2](#_Toc72249765)

[3.1.6 使用Eclemma检查测试的代码覆盖度 2](#_Toc72249766)

[3.1.7 Before you’re done 2](#_Toc72249767)

[3.2 Re-implement the Social Network in Lab1 2](#_Toc72249768)

[3.2.1 FriendshipGraph类 2](#_Toc72249769)

[3.2.2 Person类 3](#_Toc72249770)

[3.2.3 客户端main() 3](#_Toc72249771)

[3.2.4 测试用例 3](#_Toc72249772)

[3.2.5 提交至Git仓库 3](#_Toc72249773)

[4 实验进度记录 3](#_Toc72249774)

[5 实验过程中遇到的困难与解决途径 3](#_Toc72249775)

[6 实验过程中收获的经验、教训、感想 4](#_Toc72249776)

[6.1 实验过程中收获的经验和教训 4](#_Toc72249777)

[6.2 针对以下方面的感受 4](#_Toc72249778)

# 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）；

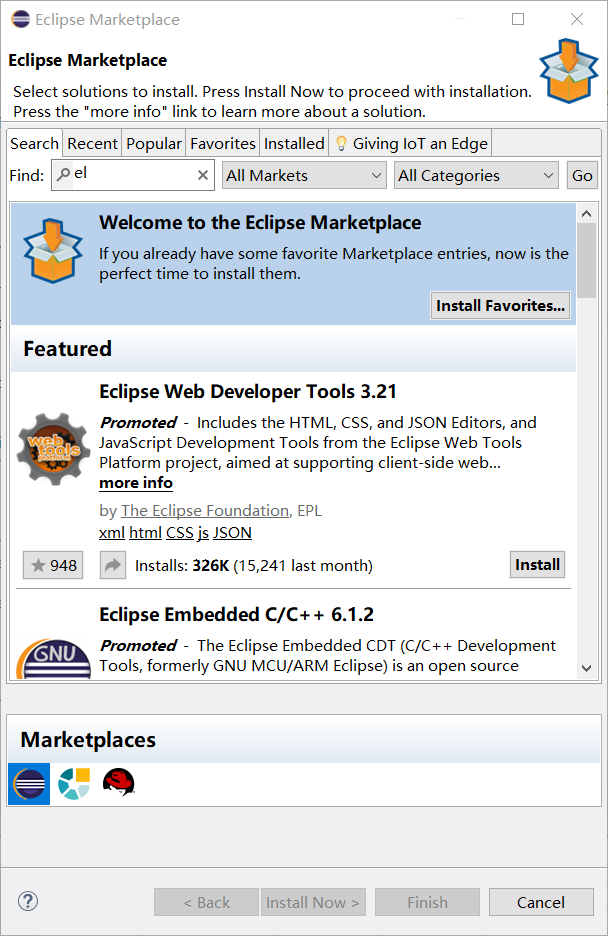
测试 ADT 的实现并评估测试的覆盖度；

使用 ADT 及其实现，为应用问题开发程序；

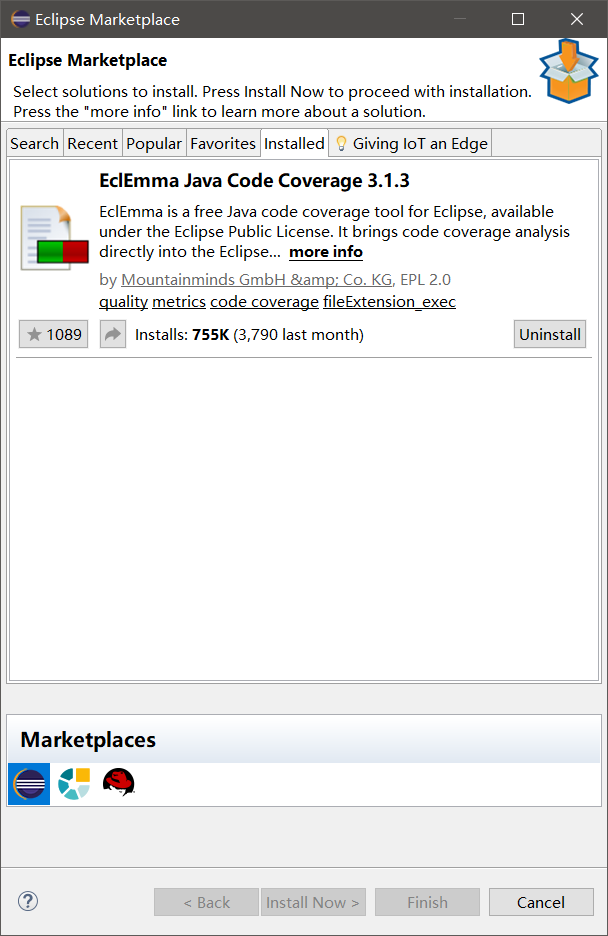
在测试代码中，能够写出 testing strategy 并据此设计测试用例。。

# 实验环境配置

我们可以在eclipse中安装MarketPlace，方便我们安装插件。



在其中搜索EclEmma，安装即可



地址如下：

<https://github.com/ComputerScienceHIT/HIT-Lab2-1190202128>

# 实验过程

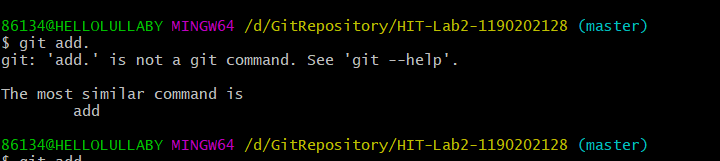
请仔细对照实验手册，针对三个问题中的每一项任务，在下面各节中记录你的实验过程、阐述你的设计思路和问题求解思路，可辅之以示意图或关键源代码加以说明（但千万不要把你的源代码全部粘贴过来！）。

## Poetic Walks

使用等价类划分的原则先写出对两种图的测试代码，在完成了对图的测试代码之后，我们就可以完成两种图的具体实现，然后将其改为L泛型。需要注意的是关于AF，RI的设计以及如何保证表示不变性的不泄露，以及defensive copies策略的应用。

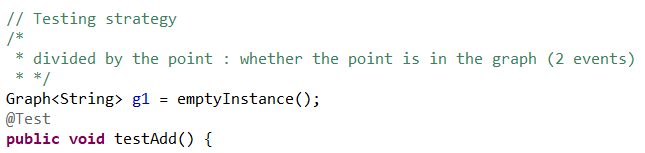
### Get the code and prepare Git repository

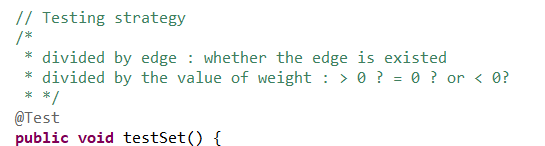
实验环境配置同第一次实验，只需要将代码仓库改名，再将仓库clone至本地，并且将分支名改为master即可（如下图）

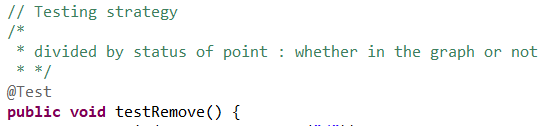


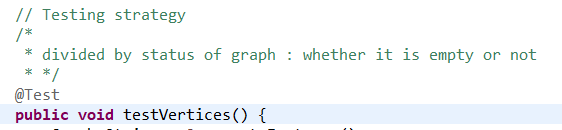
### Problem 1: Test Graph <String>

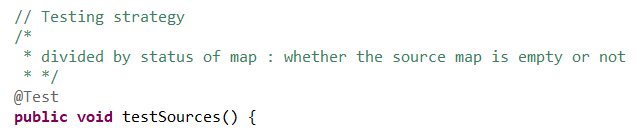
可以利用等价类划分的思想为Graph<String>写出测试用例，值得注意的是，我们的测试应该覆盖写到的每一个方法，并且尽量覆盖其中的每一个分支，具体测试策略如下：

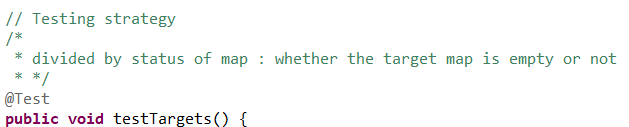












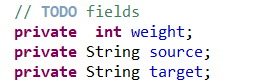
### Problem 2: Implement Graph <String>

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。

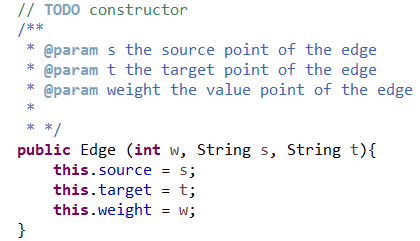
#### Implement ConcreteEdgesGraph

1. 实现Edge类

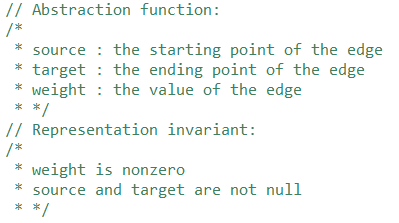
对于Edge类的数据成员，显然我们需要定义两个String分别保存起始点source和终点target，我们还需要一个int类型的数据来保存边的权值，这些数据我们不希望被外部直接访问修改，因此设置为private。



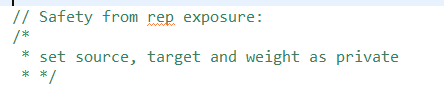
此外，我们需要提供类的构造方法如下：



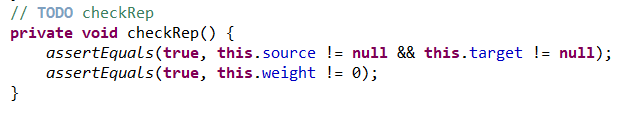
关于AF，RI的声明：



防止不变性泄漏的策略：



以及对表示不变性的检查：

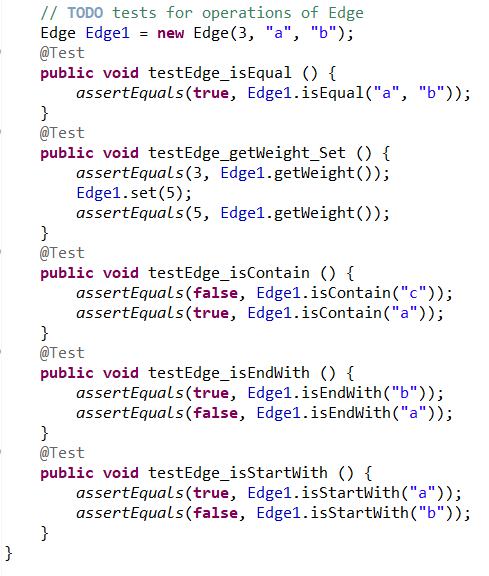


关于AF，RI

在此基础上，我们可以给出edge类中的其它方法的描述：

|  |  |  |
| --- | --- | --- |
| 函数原型 | 功能描述 | 是否调用checkRep |
| **Public boolean** isEqual(String source, String target) | 判断Edge中的起始点和终点是否与输入参数对应相等 | 否 |
| **public** **boolean** isContain(String vertex) | 判断Edge中的起始点或者是终点是否为vertex | 否 |
| **public** **boolean** isStartWith(String vertex) | 判断Edge中的起始点是否为vertex | 否 |
| **public** **boolean** isEndWith(String vertex) | 判断Edge中的终点是否为vertex | 否 |
| **public** **void** set(**int** weight) | 设置Edge中的权重为输入值 | 是 |
| **public** **int** getWeight（） | 返回Edge的权值 | 否 |
| **public** String getSource() | 返回edge的起始点 | 否 |
| **public** String getTarget() | 返回edge的终点 | 否 |
| **public** String toString() | 返回”source -> target weight : %d\n”形式的字符串 | 否 |

在完成了其具体方法实现之后，我们可以依据等价类划分的原则对于Edge类进行测试：

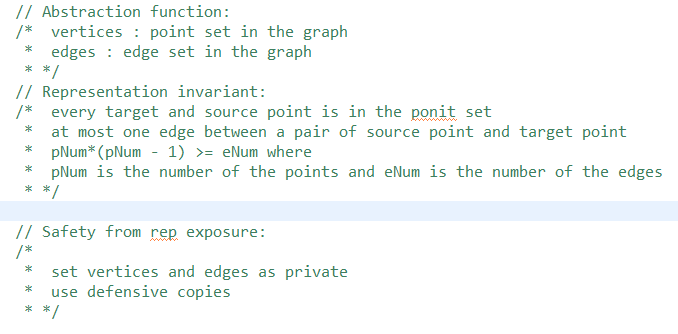


1. 实现ConcreteEdgesGraph

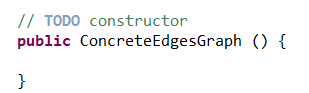
我们需要的数据结构包括一个记录所有点的点集和一个边的列表，同样，处于防止表示不变性泄漏的原因，我们将其都设置为private。



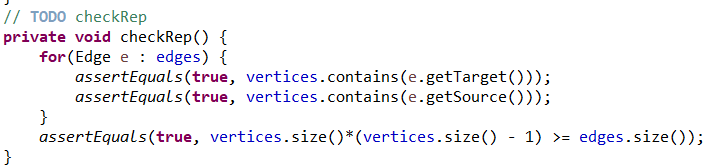
给出AF，RI等信息：



给出类的构造器：

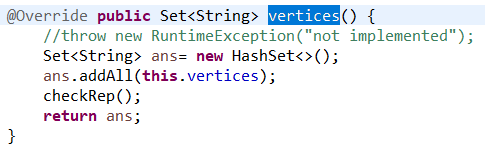


然后针对之前的防止泄漏策略写出checkRep方法：

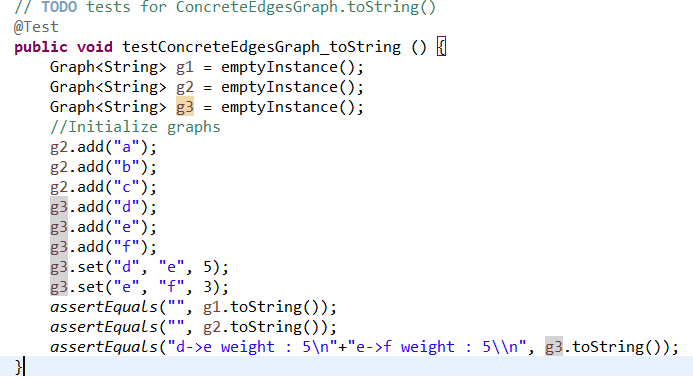


此外，不同于上一个实现的类Edge，在ConcreteEdgesGraph中所有实现的方法均为重写Graph中的方法，因此我们不在此处一一列出所有的实现，仅在此处对于哪些方法调用的checkRep方法做一个说明。事实上，经过分析发现，除了sources，targets和toString方法外，都需要调用checkRep方法，因为他们都改变了内部的数据。

另外，在针对vertices方法设计的时候，我们需要使用到defensive copy策略：

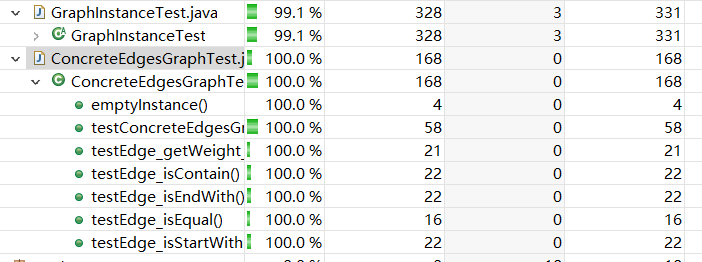


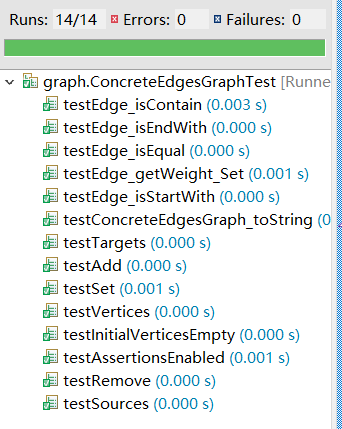
对于其中的toString方法，我们可以依据等价类划分的原则，分别测试一个空图，无边图和一个有边正常图：



代码覆盖率如下：

对于该图的整体测试结果如下：

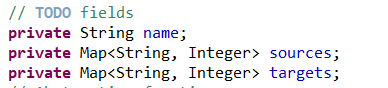




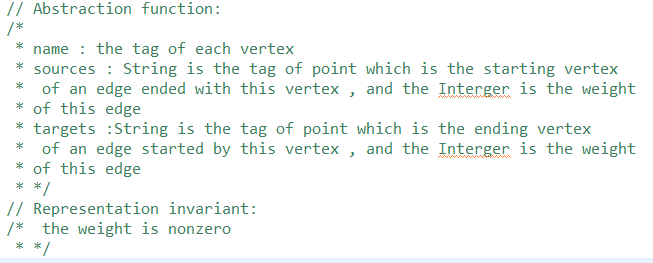
#### Implement ConcreteVerticesGraph

1. 实现Vertex类

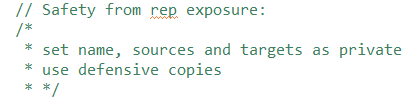
首先，我们需要明确此类的数据结构：



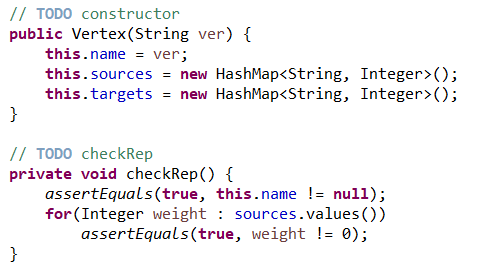
然后给出这个类的AF和RI：



给出防止表示不变性泄漏测策略：



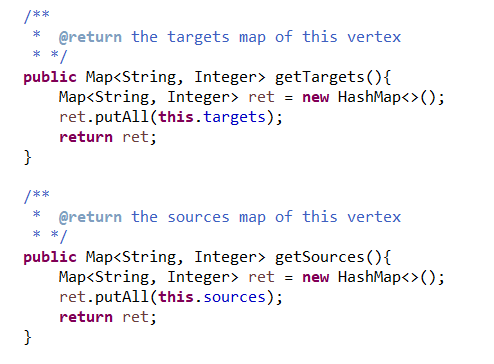
然后给出构造方法和检查表示不变性的方法checkRep：



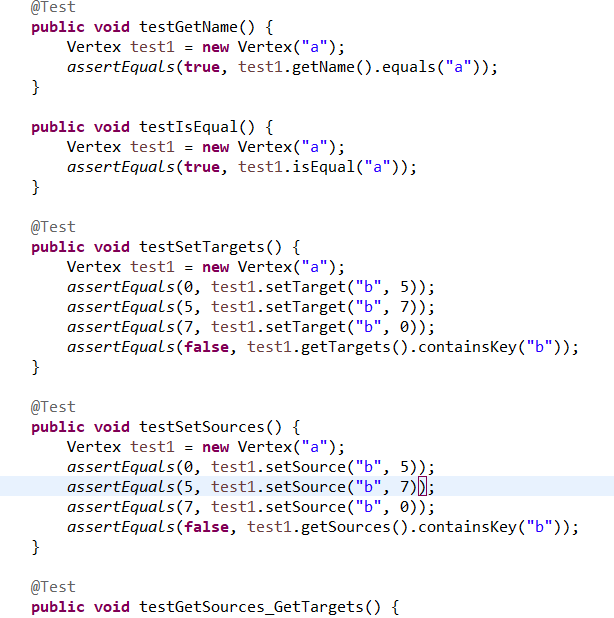
在此基础上，我们可以给出对于其它方法的实现的简单描述：

|  |  |  |
| --- | --- | --- |
| 函数原型 | 功能描述 | 是否调用checkRep |
| **public** String getName() | 返回点的名称 | 否 |
| **public** Map<String, Integer> getTargets() | 返回该点可以到达的终点和权值的map | 否 |
| **public** Map<String, Integer> getSources | 返回可以到达该点的起点和权值的map | 否 |
| **public** **int** setTarget(String target, **int** weight) | 如果weight不为0，则设置或更新该点的targets中的target关键值为的值为weight，如果target不存在则添加进targets。  如果weight为0，则删除targets中的target关键值，如果target不存在则不做修改。  返回target关键值之前的值，如果之前不存在target则返回0。 | 是 |
| **public** **int** setSource(String source, **int** weight) | 如果weight不为0，则设置或更新该点的sources中的source关键值为的值为weight，如果source不存在则添加进sources。  如果weight为0，则删除sources中的source关键值，如果source不存在则不做修改。  返回source关键值之前的值，如果之前不存在source则返回0。 | 是 |
| **public** **boolean** isEqual(String name) | 判断输入的name是否与当前vertex的name一致 | 否 |
| **public** String toString() | 将该点信息转换为字符串 | 否 |

值得注意的是，我们需要在getTargets和getSources方法中用到defensive copies策略：

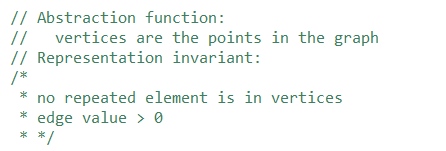


关于其测试策略，我们只需要覆盖其中的每一个方法和尽可能多的分支即可：

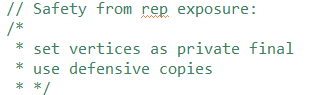


1. 实现ConcreteVerticesGraph

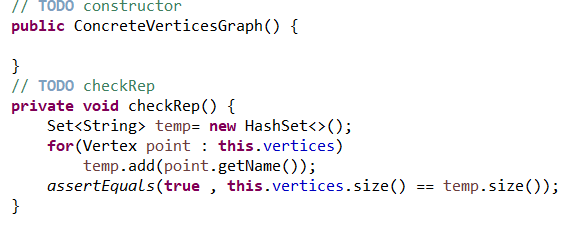
我们首先给出该类的AF和RI信息：



由此给出防止表示不变性泄漏的策略：



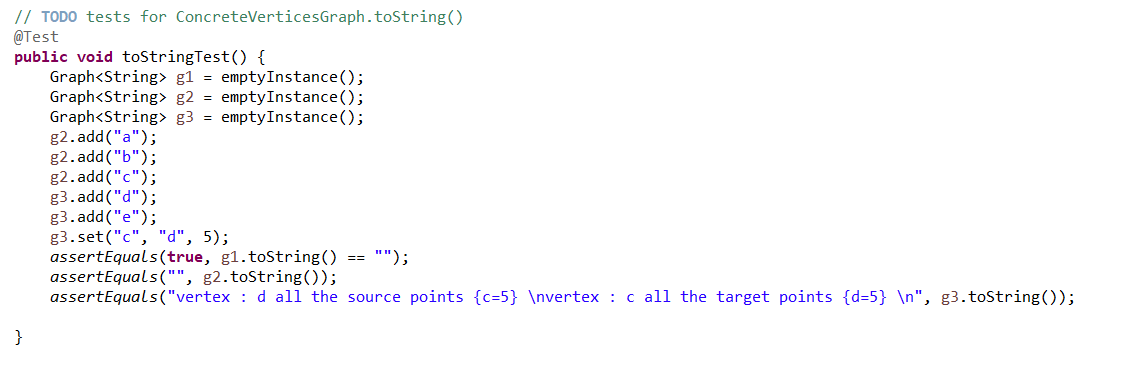
再给出类的构造器和checkRep方法：



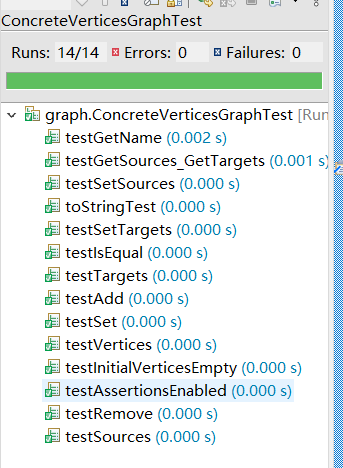
而除此之外的方法我们均是在重写Graph类中提供的方法，在此不一一列出，只在此处列出调用了checkRep方法的方法，经过研究我们发现，只有在add，set和remove方法中，ConcreteVerticesGraph的变量值才会被改变，因此我们在上述方法返回前调用checkRep。

关于defensive copies策略，我们发现在调用vertices，sources和targets方法时都需要使用防御性拷贝。

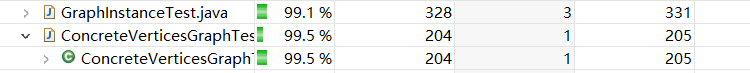
关于其中的toString方法的测试代码如下：



对该图的整体测试如下：



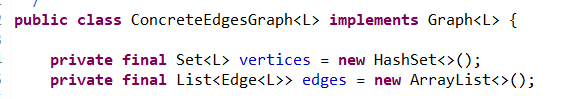
代码覆盖度情况如下：

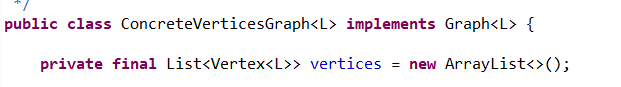


### Problem 3: Implement generic Graph<L>

#### Make the implementations generic

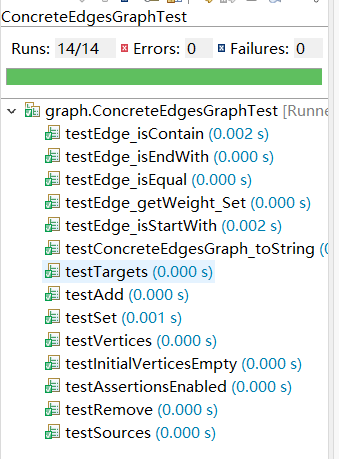
只需要将String替换为L即可，注意不应改变toString方法中的String。然后按照题目要求修改接口和具体实现的定义：

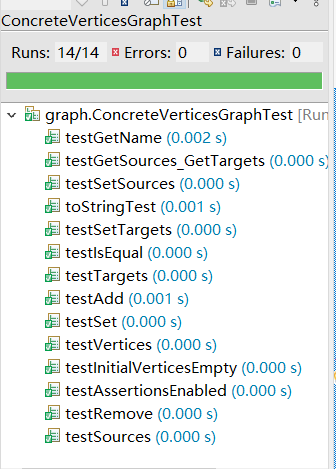




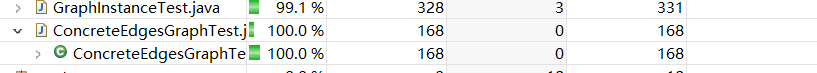
再将出现的Edge和Vertex替换为Edge<L>和Vertex<L>即可。

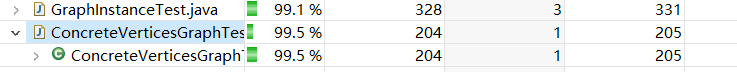
经过泛型转化之后继续运行Junit测试：





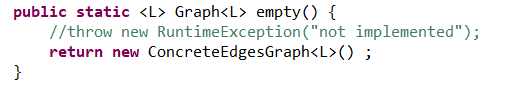
代码覆盖率：

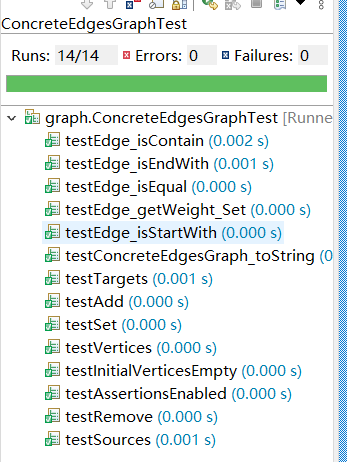




#### Implement Graph.empty()

我们可以选ConcreteEdgesGraph来实现，只需要修改代码如下即可：



再来查看此时的测试情况：  


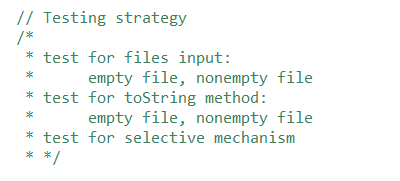
代码覆盖率：



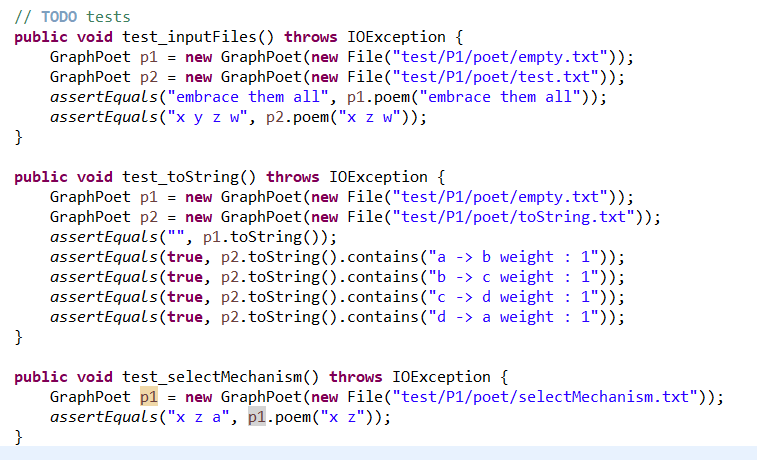
### Problem 4: Poetic walks

#### Test GraphPoet

我们只需要根据等价类划分的原则，合理划分等价类即可，测试策略如下：



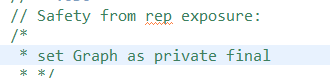
具体测试代码如下：



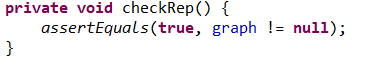
#### Implement GraphPoet

首先我们给出该类的AF和RI等信息：

给出避免数据泄露的方法：



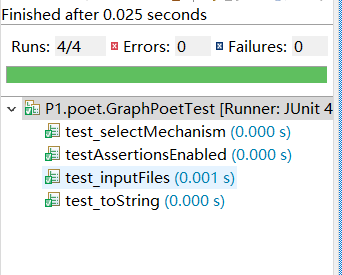
以及不变性检测的方法：



该类总共需要我们实现两个方法，在第一个方法GraphPoet中我们需要读取文件中的语料库并生成相应的图结构，我们可以采用BufferedReader逐行读取文件的内容，并且将读取到的内容按照空格符分离，并且添加进List中，然后再将List中的元素添加进图中。

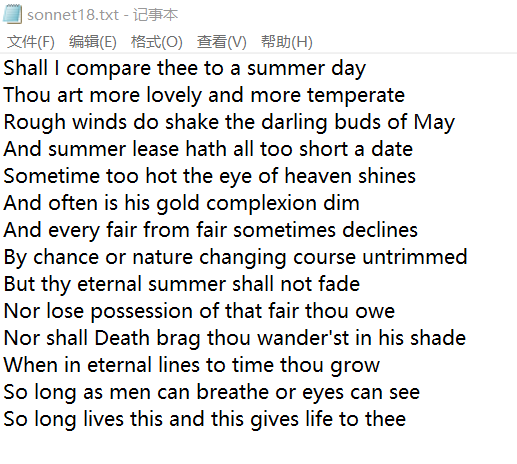
而对于第二个方法poem，我们则需要按照空格分割我们输入的诗句，然后进行桥的添加。我们还需要在有多座桥的时候选择权值最大的路径。而对于桥节点的寻找，我们只需要对当前节点的targets集合和下一个节点的sources集合取交集即可。

在完成程序后，我们运行检测模块如下：

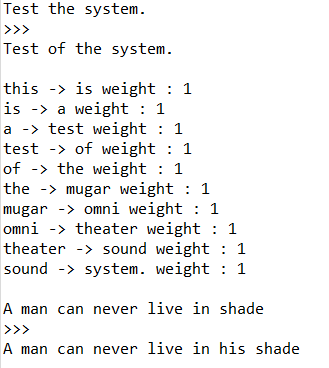


#### Graph poetry slam

我为原测试添加了toString的步骤，使其生成的树结构更直观，然后添加了一个语料库如下：



该语料库较大，就略去了toString环节，直接输出相应的诗句，整体测试结果如下：



### 使用Eclemma检查测试的代码覆盖度

Poet部分的代码覆盖率如下：

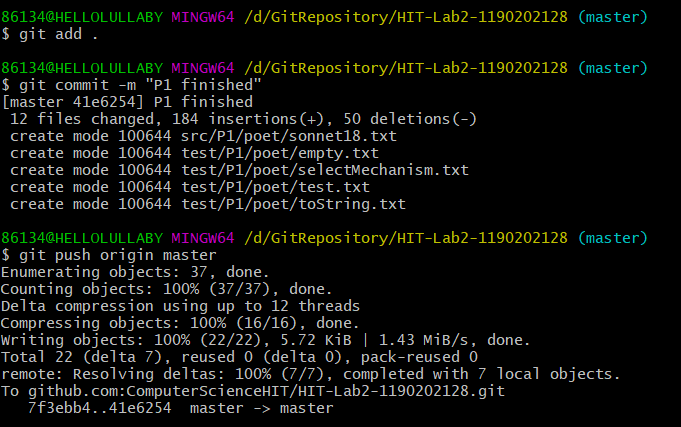
（其余部分的代码覆盖率附在相应的代码实现部分之后）



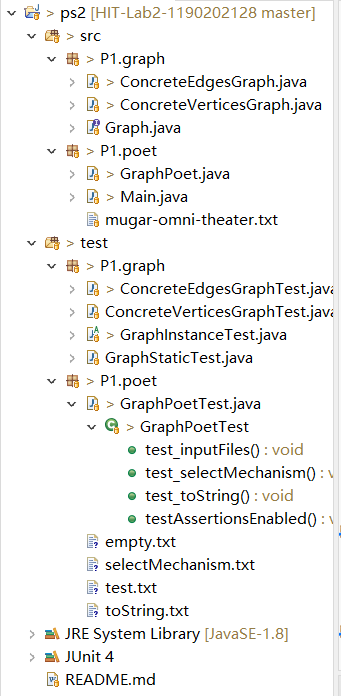
### Before you’re done

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。

提交当前的进度，只需如下操作即可：



项目的目录结构树状示意图：



## Re-implement the Social Network in Lab1

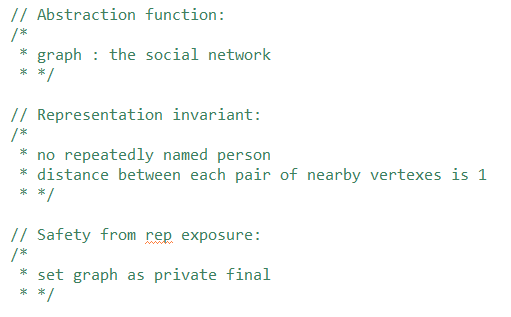
通过实现的graph接口和相应的类来重新实现FriendshipGraph即可，需要注意的是之前通过Floyd算法实现的getDistance无法复用，需要重新构建通过BFS算法实现的getDIstance方法。

### FriendshipGraph类

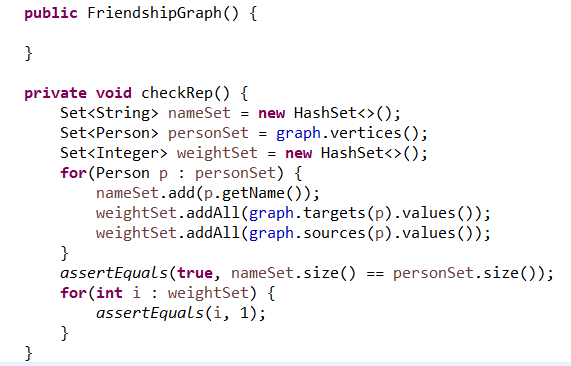
首先我们给出该类的成员变量：



其次我们可以给出该类的AF和RI，以及防止数据泄漏的策略：



然后据此给出对应的构造器和checkRep方法：



然后我们可以据此给出别的方法的概述：

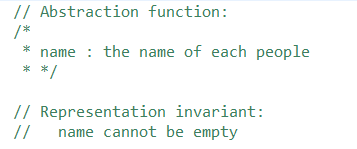
|  |  |  |
| --- | --- | --- |
| 函数原型 | 功能描述 | 是否调用checkRep |
| **public** **boolean** addVertex(Person person) | 将一个人person加入社交网络中，如果以存在此人则不加入其中且返回false，否则返回true且将其加入社交网络 | 是 |
| **public** **void** addEdge(Person A, Person B) | 设置社交网络中一条从A到B的有向边，如果A或B不存在则将其加入社交网络之后再设置有向边 | 是 |
| **public** **int** getDistance(Person A, Person B) | 返回A，B的社交距离，如果AB相同则返回0，如果AB之间没有路径则返回-1 | 否 |

### Person类

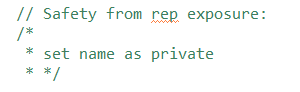
首先我们给出该类的成员变量：



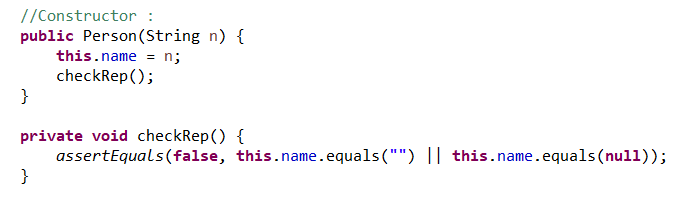
然后给出相应的AF和RI信息：



给出防止数据泄漏的策略：



给出相应的构造器和检测表示不变性的方法：

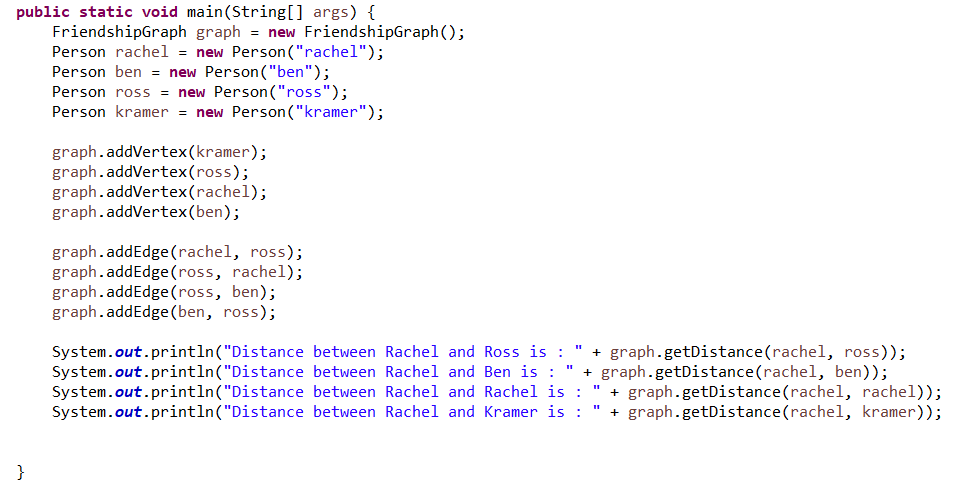


然后我们可以构造别的方法：

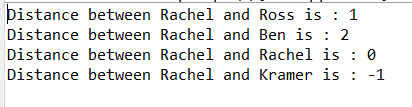
|  |  |  |
| --- | --- | --- |
| 函数原型： | 功能描述 | 是否调用checkRep |
| **public** String getName() | 返回该此人的姓名 | 否 |

### 客户端main()

在main方法中直接使用Lab1的代码：

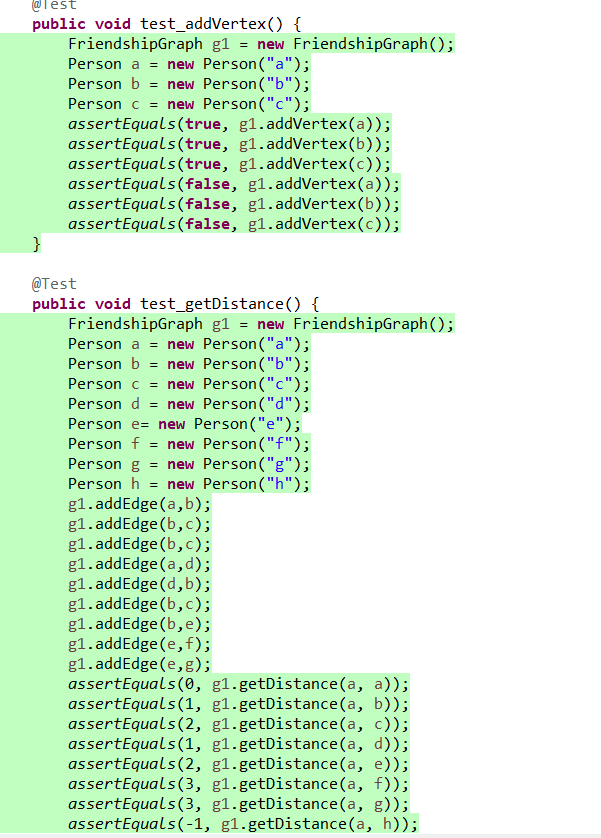


得到输出如下：

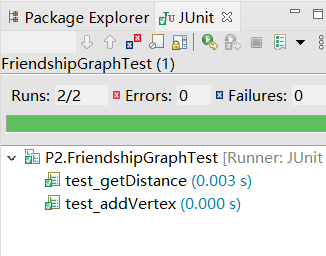


### 测试用例

按照等价类划分的原则对其进行测试，测试代码如下：



得到的结果如下：



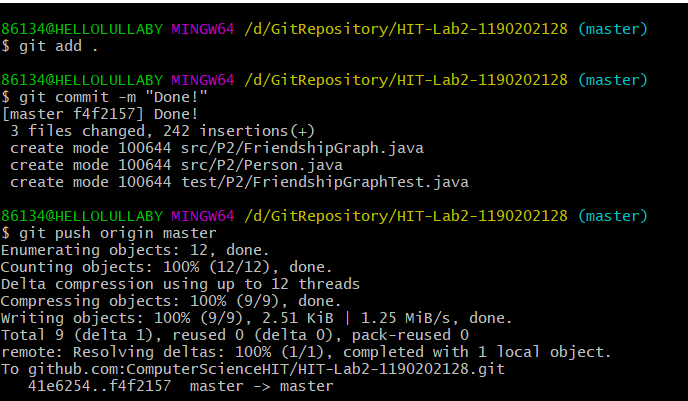
代码覆盖度如下：



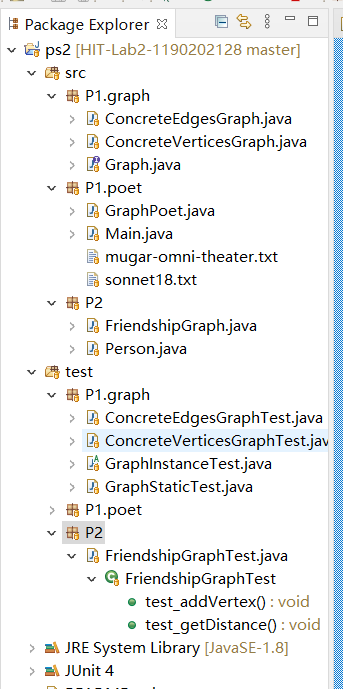
注：FriendshipGraph中代码未覆盖区域主要是由于未覆盖其中的main方法的代码

### 提交至Git仓库

如下操作即可：



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。

每次结束编程时，请向该表格中增加一行。不要事后胡乱填写。

不要嫌烦，该表格可帮助你汇总你在每个任务上付出的时间和精力，发现自己不擅长的任务，后续有意识的弥补。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 6.1 | 18:00-22:00 | P1的problem1-3 | 完成 |
| 6.3 | 17:00-22:00 | P1的problem4 | 完成 |
| 6.5 | 19:30-22:00 | P2 | 完成 |
| 6.7 | 19:30-21:40 | 完善报告 | 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 |
| Poem部分测试无法跑过 | 经过分析发现是因为在实现Graph的时候没有通过equals方法进行字符串比较而是采用了==，导致出错，而且这种错误也没有在test中显现，所以较难发现，将所有的字符串比较都换用equals方法后问题消失 |

# 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

了解了关于ADT的应用，函数规约，AF，RI等知识的使用，以及如何进行模块化编程，如何写出好的测试等等。

## 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？

面向ADT编程需要考虑更多，思维难度较大。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

使用泛型编程需要有更加好的抽象的能力，但是也方便了今后进行复用

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？

优势时能够更加不受干扰地写出测试用例，因为测试用例完全依赖于规约写出，在后期更容易测试出代码的bug

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？

好处在于可以减少程序员的工作，避免重复造轮子

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？

意义在于使得代码更易于被理解，更方便调用和进行测试，我愿意坚持。

1. 关于本实验的工作量、难度、deadline。

难度较大，好在时间较多，总体来说能够在ddl之前完成

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

希望对于PPT的注释有更多，加入更多的中文解释，希望对于国外的实验说明加入中文注释和翻译