

**2021年春季学期**

**计算学部《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 李旻翀 |
| 学号 | 1190200208 |
| 班号 | 1936602 |
| 电子邮件 | 1190200208@stu.hit.edu.cn |
| 手机号码 | 18608109998 |

**目录**

[1 实验目标概述 3](#_Toc74467257)

[2 实验环境配置 3](#_Toc74467258)

[3 实验过程 3](#_Toc74467259)

[3.1 Poetic Walks 4](#_Toc74467260)

[3.1.1 Get the code and prepare Git repository 4](#_Toc74467261)

[3.1.2 Problem 1: Test Graph <String> 4](#_Toc74467262)

[3.1.3 Problem 2: Implement Graph <String> 5](#_Toc74467263)

[3.1.4 Problem 3: Implement generic Graph<L> 12](#_Toc74467264)

[3.1.5 Problem 4: Poetic walks 13](#_Toc74467265)

[3.1.6 使用Eclemma检查测试的代码覆盖度 15](#_Toc74467266)

[3.1.7 Before you’re done 16](#_Toc74467267)

[3.2 Re-implement the Social Network in Lab1 17](#_Toc74467268)

[3.2.1 FriendshipGraph类 17](#_Toc74467269)

[3.2.2 Person类 20](#_Toc74467270)

[3.2.3 客户端main() 20](#_Toc74467271)

[3.2.4 测试用例 21](#_Toc74467272)

[3.2.5 提交至Git仓库 21](#_Toc74467273)

[4 实验进度记录 22](#_Toc74467274)

[5 实验过程中遇到的困难与解决途径 23](#_Toc74467275)

[6 实验过程中收获的经验、教训、感想 23](#_Toc74467276)

[6.1 实验过程中收获的经验和教训 23](#_Toc74467277)

[6.2 针对以下方面的感受 23](#_Toc74467278)

# 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

在Lab 1中已经完成编程环境的配置；

Lab 2中要求安装代码覆盖度插件EclEmma，但IEDA中自带了完善的代码覆盖度检查功能，故并未进行EclEmma的安装。

在这里给出你的GitHub Lab2仓库的URL地址（Lab2-学号）。

<https://github.com/ComputerScienceHIT/HIT-Lab2-1190200208>

# 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

本环节给出了一个Graph接口，要求我们秉承测试驱动开发的思想，分别建立继承graph接口的边图类、点图类，并且在类中具体实现操作graph的一系列方法。另外，我们需要在类中具体实现Edge或Vertex数据型，在完成之后，用我们设计的图的抽象数据型完成poetic walks的工作。这一环节的主要目的是动手实践ADT的规约设计并具体设计、实现ADT。

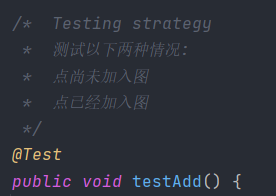
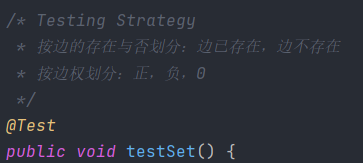
### Get the code and prepare Git repository

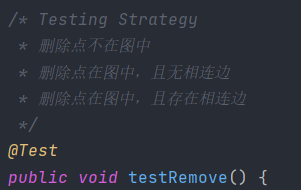
从<https://github.com/rainywang/Spring2021_HITCS_SC_Lab2/tree/master/P1>获取实验代码，使用git pull与本地进行连接，完成后查看具体工程文件，修改包路径以便运行。

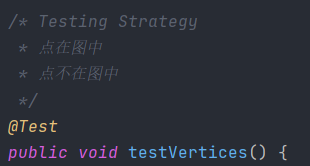
### Problem 1: Test Graph <String>

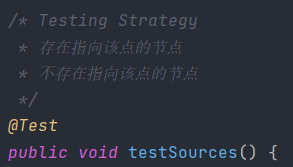
这一部分是按照测试驱动开发的思想，针对Graph<L>的不同方法与成员，设计出相应的等价类划分测试。

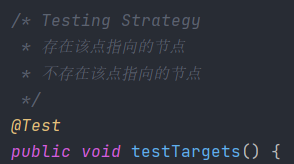
不同方法、成员的测试策略如下：









### 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。

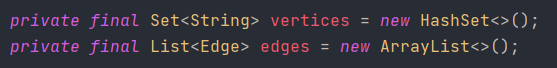
#### Implement ConcreteEdgesGraph

为ConcreteEdgesGraph重写抽象类。

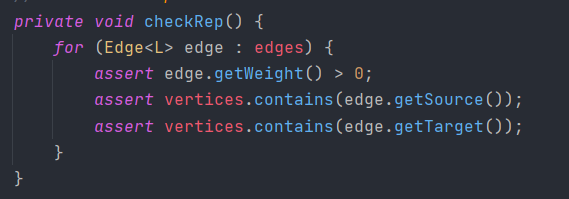
对于ConcreteEdgesGraph，其AF、RI与防止Rep泄露的安全措施如下：



ConcreteEdgesGraph中有以下两个rep：



为了防止表示泄露，在调用各个方法返回之前，使用checkRep方法检查是否有表示泄露的情况出现，具体实现以RI为原则：



以下简要阐述ConcreteEdgesGraph类重写的各个方法：

1. add方法。

为图的顶点集vertices添加新的顶点。若vertices中已有此顶点，返回false，否则返回true。

1. set方法。

根据输入参数source与target对边集edges进行检索，若检索到对应的边，则将其边权修改为参数weight的值，并返回被替换的值。若未检索到对应边，则返回-1。

在设计此方法时，有两个地方需要注意：

1. 在遍历时进行删除，需要显式使用迭代器，不然会产生严重错误。
2. 因为题中要求Edge是不可变的，所以在修改边权时，若已存在由source到target的边，则需要先删掉原来的边，再添加一条起点终点不变，边权修改过的新边，而不能直接在原来边的基础上对权值进行修改。
3. remove方法

从点集vertices中删掉参数对应的点及点所连接的边。若不存在对应点，则返回false，否则返回true。

1. vertices方法

返回顶点集合。

在这一部分中，为防止rep泄露，需要新建一个Hashmap保存返回的顶点。

1. sources方法

寻找target对应的原点，并将原点与对应的边权以键值对的形式保存在Map中返回。

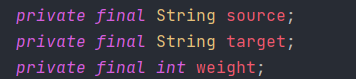
1. targets方法

寻找source对应的终点，并将终点与对应的边权以键值对的形式保存在Map中返回。

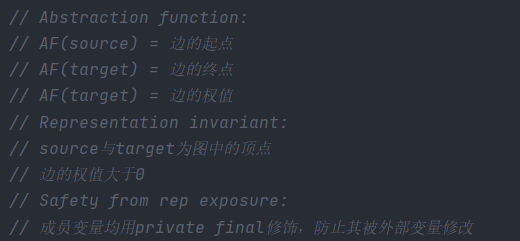
1. toString方法

调用Edge类中的tostring方法，将图的信息以字符串形式输出。

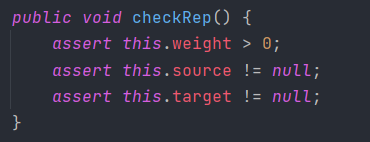
在设计Edge类时，为其设计了三个私有成员变量，分别代表起点，终点，边权，由于其用private final关键字修饰，所以可以防止表示泄露的问题：



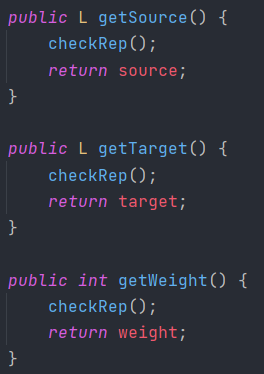
Edge类的AF、RI与防止Rep泄露的安全措施如下：



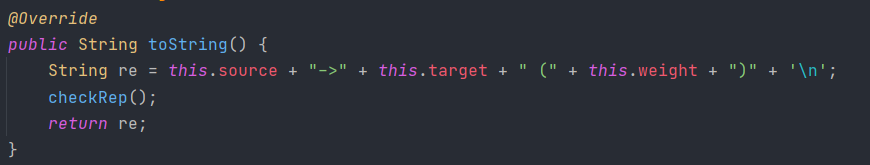
根据Edge的RI，设计其checkRep方法如下：



因为Edge是不可变的，所以为其设计了3个Getter，分别用于获取起点，终点以及边权。并未设置对其成员变量进行修改的setter。



Edge的toString方法用于输出该边的起点，终点以及边权。输出格式为： a->b (5)，代表由a指向b，权值为5的边。

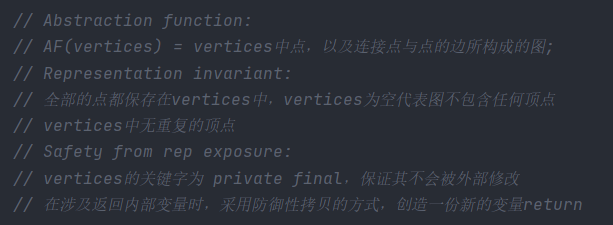


在对Edge进行比较时，由于在实现各个方法时并未对不同的Edge进行比较判断其是否相等，所以没有对Equals方法与Hashcode方法进行重写。

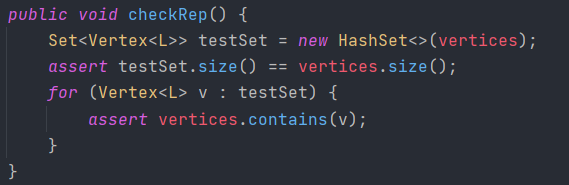
#### Implement ConcreteVerticesGraph

类似于ConcreteEdgesGraph，ConcreteVerticesGraph也是对Graph接口的具体实现。在Implement ConcreteVerticesGraph的步骤中，我们需要从已有的成员变量出发，重写Graph中的抽象方法，并自行设计ConcreteVerticesGraph中所用到的Vertex类的成员及方法。

对于ConcreteVerticesGraph，其AF、RI与防止Rep泄露的安全措施如下：

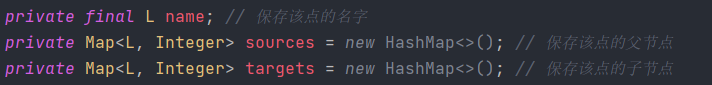


根据ConcreteVerticesGraph的RI，设计其checkRep方法如下，主要完成了对vertices是否包含了图中所有顶点的检查。

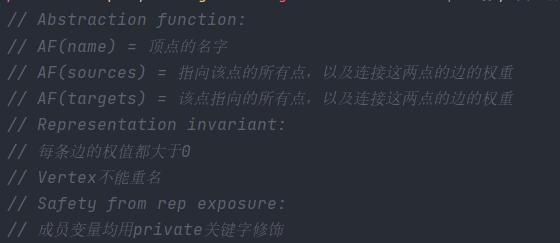


具体每个方法的重写思路与过程在此不再赘述。

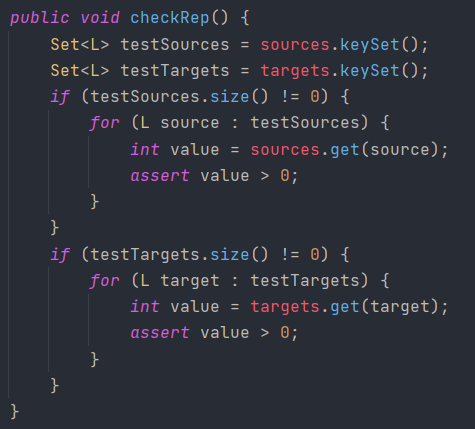
对于Vertex类，由于ConcreteVerticesGraph中的成员变量较少，所以Vertex类需要承担更多的功能，设计其成员变量如下：



对于Vertex类，其AF、RI与防止Rep泄露的安全措施如下：

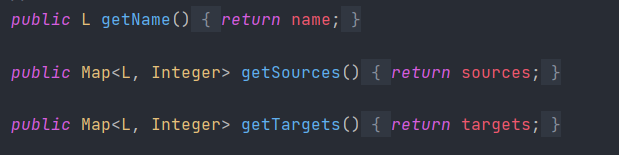


根据Vertex的RI，设计其checkRep方法，在checkRep方法中，主要检查了该点以及该点的子节点、父节点的权值是否始终为正。具体实现如下：

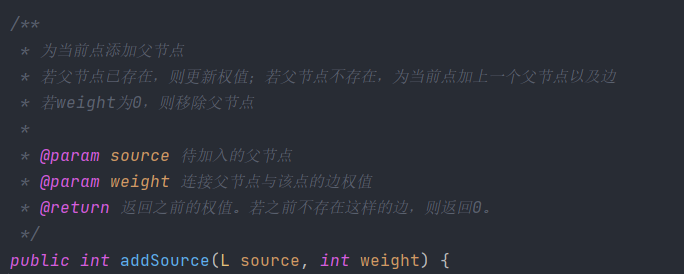


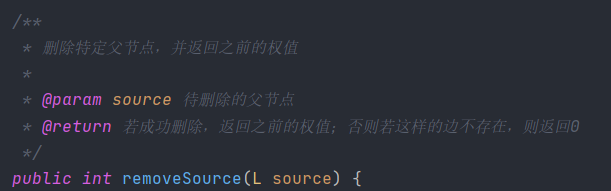
由于Vertex是可变的类，所以在为其设计三个getter之外，要设计能够修改其成员变量的public方法。

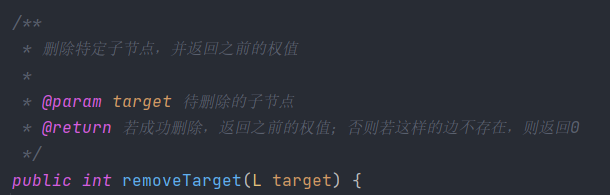
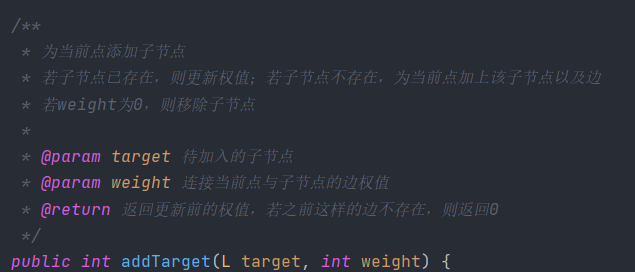
首先，Vertex类的getter如下：



除此之外，设计了四个方法：addSources、addTargets、removeSource、removeTarget，以便对Vertex类的私有成员sources与targets进行修改。这四个方法的spec如下：







对于Vertex类，设计其toString方法如下，以便以字符串形式返回该点的具体信息。



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

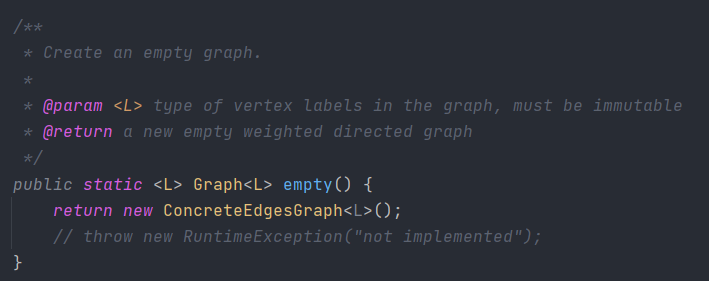
#### Make the implementations generic

在这一步中，需要对之前所编写的程序进行重构，使得我们的类不依赖于具体的String类，而能适用于任意类型L。

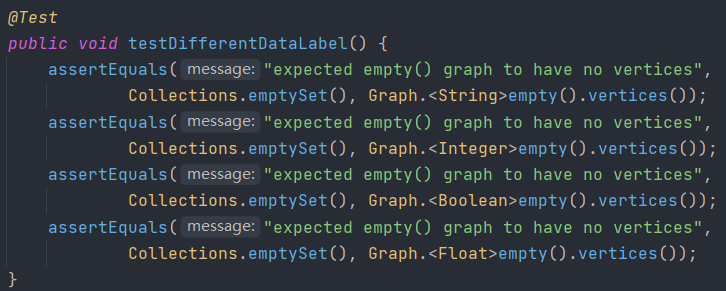
具体的修改方式是将原来代码中的用于表示泛型的String类型替换为泛型的标识符L，并将Edge修改为Edge<L>，Vertex修改为Vertex<L>。根据IDEA的提示进行一步步的修改即可。

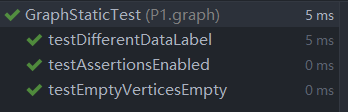
#### Implement Graph.empty()

在Graph.java中修改Graph.empty()的具体实现即可，如图所示：



修改完成后，我们在GraphStaticTest中添加对其的测试用例，测试结果及覆盖度如下：







### Problem 4: Poetic walks

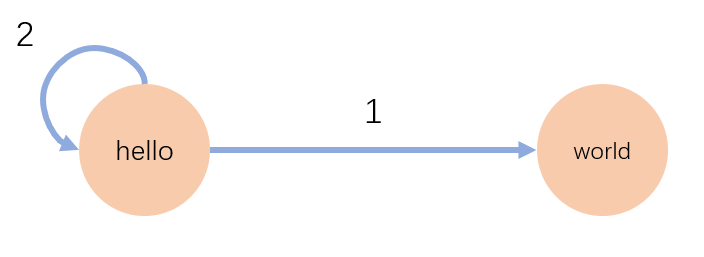
#### Test GraphPoet

首先需要理清Problem 4具体完成的任务。

在这一步中，需要依据语料生成一个单词图，单词图的每个顶点是语料中的一个单词，单词图的边代表前一个单词紧接着后一个单词，边的权重为前一个单词紧接着后一个单词的次数，并且不考虑大小写与标点符号。举个例子：

hello,hello,HeLlo,world!

其中，hello→hello出现两次，hello→world出现一次，则这个语料构成的图为：

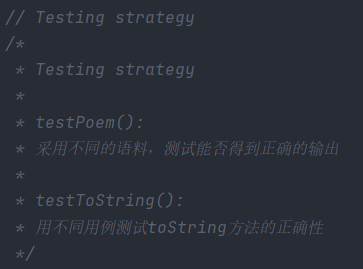


更加具体的示例可以参照MIT实验官网的页面。

在根据语料构造好图后，下面需要完成的任务是：输入一个句子，提取出句子两两相邻的单词对，在图中进行一次检索，若句子的前后两单词w1→w2在单词图中隔着一个顶点b，则将b加入句子中，得到w1→b→w2。具体的构建规则如下：

1. w1与w2间只能间隔一个顶点，这也就是说，如果在图中出现了w1→a→b→w2，甚至间隔更多的情况，则不会在w1→w2间加入单词。
2. 如果从w1到w2同时有两条路径w1→a→w2与w1→b→w2，则选择权重最高路径上的单词加入w1与w2之间。
3. 输出的句子中原单词的大小写保持不变，加入的单词全用小写。

在理清思路后，我们可以着手开始编写测试用例，具体的测试策略为编写Poem与toString方法的测试用例，对于Poem，采用不同的语料，测试能否得到正确的输出；对于toString，用不同用例测试toString方法的正确性。如图所示：

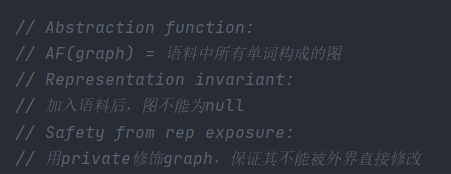


#### Implement GraphPoet

GraphPoet的成员变量如下，为一个初始化的空Graph，Graph的具体实现依赖于ConcreteEdgegGraph或ConcreteVerticesGraph：



其AF，RI，以及防止表示泄露的措施如下图所示：



在GraphPoet中，我们需要依次实现三个方法：

一．构造方法GraphPoet

在GraphPoet的构造方法中，我们需要读取文本文档，并由其中语料按照前文所阐述的方法生成一个单词图。具体实现方式为：输入文件路径并按行读入，将单词进行切分，进行删除标点符号，大写转小写等预处理，每次取相邻元素在图中添加新边。

二．生成句子方法Poem

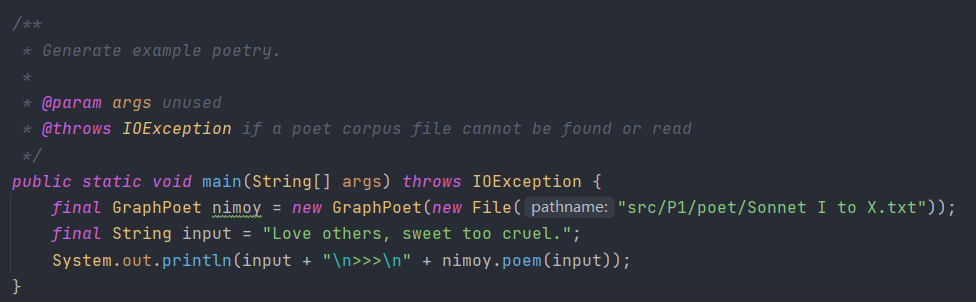
Poem方法输入一个String参数作为原始句子，输出根据单词图匹配过的，扩充了bridge words的新句子。具体实现思路为一次读入两个单词，检索前一个单词子节点的子节点，若其中包含后一个单词，则选择途径通路权重最高的一路，将该路上途径节点表示的单词加入原来两单词之间。

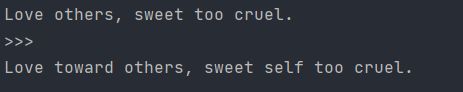
三．toString方法

简单重写toString方法，将整个图中所有点的指向转化为一条字符串输出。

#### Graph poetry slam

将MIT页面上的Sonnet I-X保存到Sonnet I to X.txt文档中，修改main函数，函数输入与输出如下：

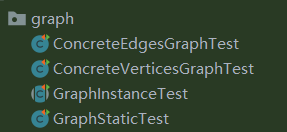


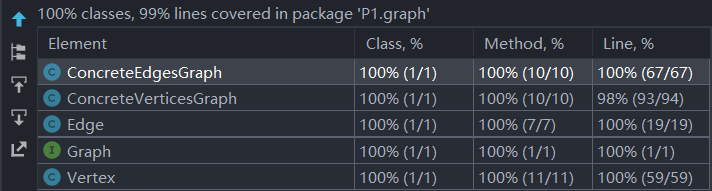


### 使用Eclemma检查测试的代码覆盖度

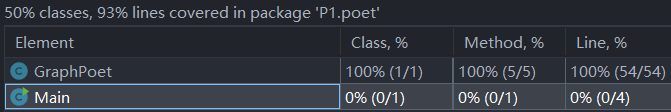
本人使用的IDE为IDEA，其中自带了代码覆盖度的测试模块，故直接用IDEA的内置模块对代码覆盖度进行测试。

运行Graph的Test用例，得到的总体代码覆盖度几乎均达到了100%：





运行GraphPoetTest，其对GraphPoetTest的覆盖度也达到了100%：



所以可以认为，我们的测试是充分的。

### Before you’re done

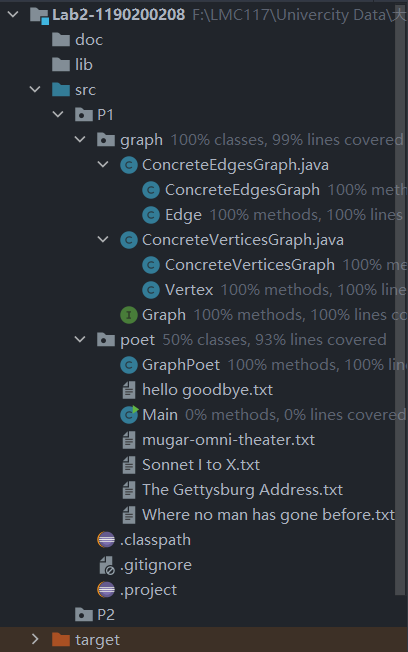
请按照[http://web.mit.edu/6.031/www/sp17/psets/ps2/#before\_youre\_done](http://web.mit.edu/6.031/www/sp17/psets/ps2/)的说明，检查你的程序。

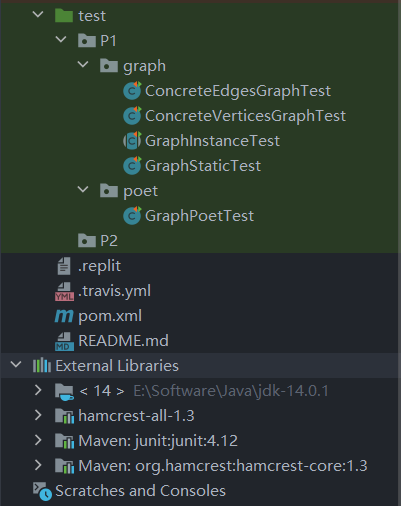
如何通过Git提交当前版本到GitHub上你的Lab2仓库。

在这里给出你的项目的目录结构树状示意图。

参考MIT的页面说明，检查确认自己编写的程序无误。并通过IDEA内置的git功能将完成版本上传至个人仓库。

项目目录结构树状示意图如下：





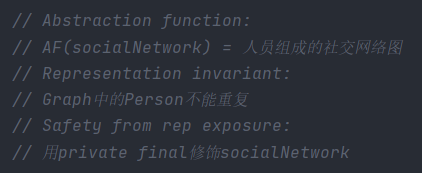
## Re-implement the Social Network in Lab1

这一环节要求我们基于在前面步骤中定义的Graph<L>及其两种实现，将泛型L替换为Person，按照Lab1中Social NetWorek的要求，实现Lab1 Problem 3中FriendshipGraph的各种功能，并且尽可能复用我们在前文构造的类中已经实现的方法。最后，运行提供的main()和执行Lab1中的Junit测试用例，确保其正常运行。在这一环节中，我选用ConcreteEdgesGraph作为基础实现功能。

### FriendshipGraph类

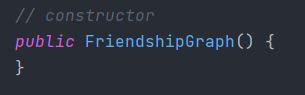
在这一环节中FriendshipGraph设计为ConcreteEdgesGraph<Person>，但因其与ConcreteEdgesGraph的继承关系，为便于访问，在FriendshipGraph的具体实现时均采用super关键字修饰。

FriendshipGraph类的AF，RI以及防止表示泄露的措施为：

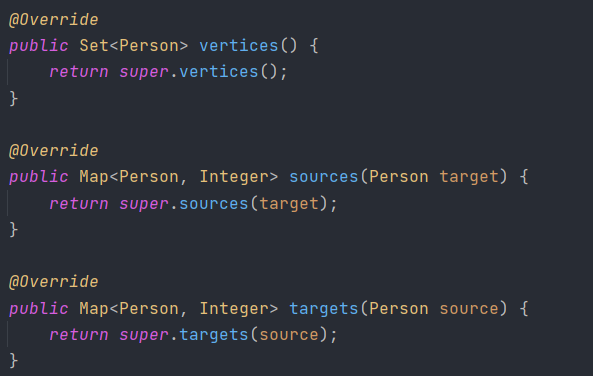


在FriendshipGraph类中，实现了如下方法：

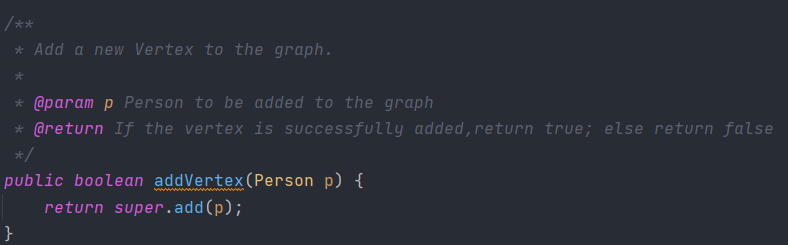
1. 构造方法



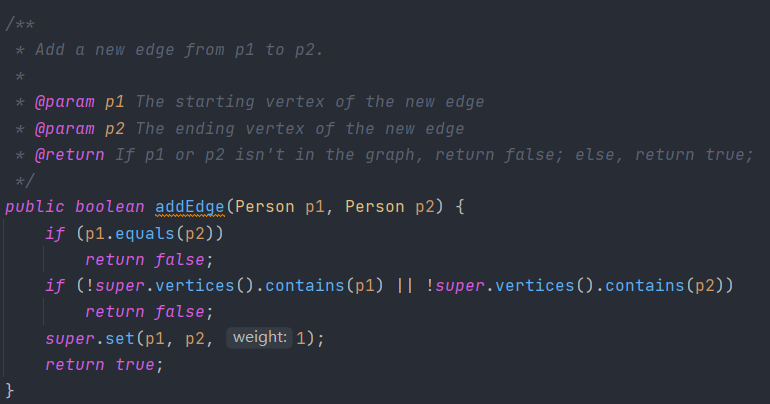
1. 重写了FriendshipGraph的vertices，sources，targets方法，使其能够返回super类（即ConcreteEdgesGraph）的对应返回值



1. addVertex方法。为图中添加顶点。调用add方法即可。



1. addEdge方法。为图中添加一条有向边。调用set方法即可。



1. getDistance方法。计算两顶点p1，p2间的最短距离。由于Lab1中该函数的实现依赖于关系矩阵，所以此方法我采用了递归的方式彻底重构，避开了涉及关系矩阵。具体而言，该方法的实现如下：

首先对异常情况进行判断，若图中不包含参数顶点或两参数顶点相同，则直接返回，之后开始递归判断。先令最短距离min为顶点数+1（保证若存在通路，则实际的最短距离一定比其小），取p1顶点的子顶点，调用函数本身递归地寻找子顶点到p2的最短距离，记为temp，若temp非-1且比min小，则令min=temp，最后根据min的值判断函数的返回值。



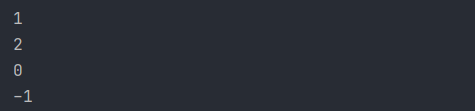
### Person类

Person类设置了一个成员变量String name表示其名字，并重写了equals与hashcode方法方便对不同的Person类进行比较。与Lab1不同的是，Lab1中的Person类还具有一个表示其在关系矩阵中位置的变量。由于Lab2实现时避开了关系矩阵，故删掉了该成员变量。



### 客户端main()

此处的客户端与Lab1中的客户端保存一致，最终的运行结果为：



结果正确。

### 测试用例

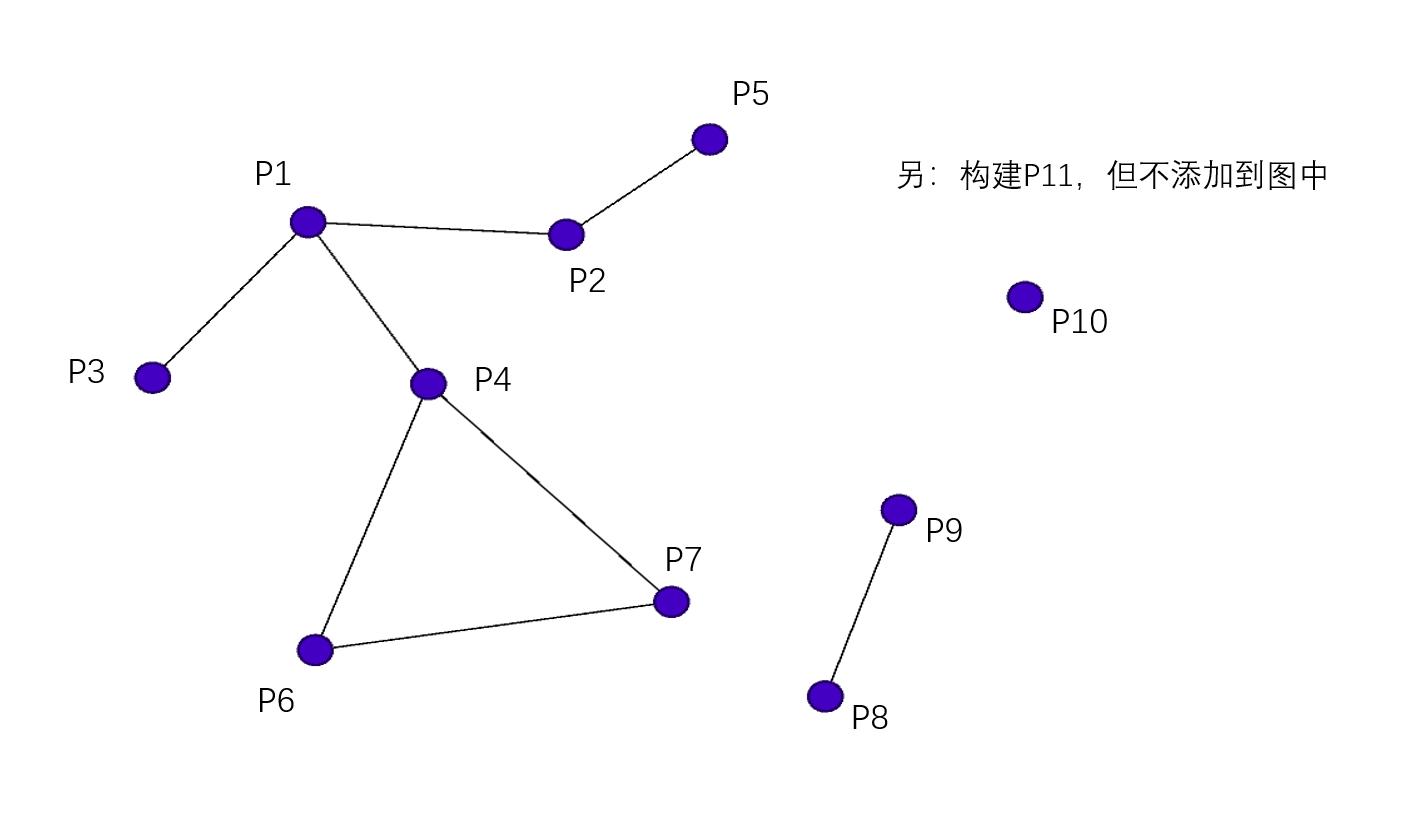
测试策略如下：

分别对addVertex，addEdge，getDistance方法进行测试。

对于addVertex方法：测试加入节点不存在，加入节点已存在两种情况；

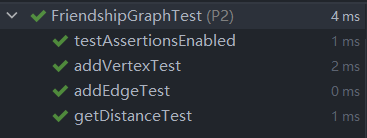
对于addEdge方法：测试正常加入，头尾节点不存在，加入自己到自己的边三种情况；

对于getDistance方法，构造如下图：



测试这几种情况：顶点到自己的距离，顶点间存在通路，顶点间不存在通路，顶点不在图中。

测试结果如下：



### 提交至Git仓库

通过IDEA内置的git功能直接上传至github的对应仓库

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 |
| 2021.5.25 | 14:00-16:00 | 对实验有一个整体把握，开始编写3.1 | 按计划完成 |
| 2021.5.28 | 10:15-11:00 | 完成GraphInstanceTest测试代码的编写 | 按计划完成 |
| 2021.5.30 | 14:00-19:00 | 完成ConcreteEdgesGraph基本函数的构建，完成ConcreteVerticesGraph的vertice类编写 | 按计划完成 |
| 2021.5.31 | 16:00-21:00 | 完成ConcreteVerticesGraph的方法重写，并重构了两个图的具体实现，使其不依赖于String | 按计划完成 |
| 2021.6.1 | 19:00-22:00 | 完善之前的测试代码 | 按计划完成 |
| 2021.6.2 | 17:00-22:00 | 完善测试代码，并为Edges与Vertex类编写单独测试用例，并测试两种图的toString( )方法 | 按计划完成 |
| 2021.6.3 | 17:00-22:00 | 完成GraphPoet测试用例的编写；  完成GraphPoet部分构造函数的编写 | 按计划完成 |
| 2021.6.5 | 8:30-11:30 | 完成GraphPoet余下部分的编写 | 按计划完成 |
| 2021.6.6 | 14:00-18:00 | 完成P2以及报告撰写 | 按计划完成 |
| 2021.6.7 | 8:30-9:30 | 检查、调试及完善 | 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 |
| 在刚上手完成实验时，对AF，RI，以及checkRep, 表示泄露等概念理解不深，感到无从下手 | 通过老师继续上课讲解相关内容，以及自己课后看PPT复习得以解决 |
| 设计测试用例时思路较为混乱 | 复习之前的PPT，通过划分等价类求笛卡尔积的方法，按规则编写测试用例 |
| 对继承的关系理解不深，在设计P2的函数时出现问题 | 查阅CSDN博客，了解super关键字的使用方法 |

# 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

通过此次实验，我体验了自行设计ADT的整个流程，对TDD与其他开发理论的了解进一步加深。Java编程能力也有所提高。

这次实验给我的教训是，在开始动手之前要思考好整体的架构如何一步步搭建。

在自行设计多种类来实现功能的情况下，自己设计的很多类之间有很多重复和矛盾的部分，函数与函数之间，类与类之间的关系自己感觉还有些混乱，也走了不少弯路，对Java的语法也还不算特别熟悉，这需要我多加练习。

## 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编程与面向应用场景编程有相当大的差异。面向ADT编程要求对编写的程序整体有充分的认识，要考虑接口、抽象类、类的整体设计，考虑哪些部分能够复用，哪些部分才需要具体实现。面向ADT编程做的好可以节省大量时间；而面向应用场景编程则有“就事论事”的意味，好处是针对一件事做，编程思路简单，问题是工作量大，复用性低。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

泛型可以适应更多的应用场景，相对不使用泛型复杂度略有提升，使用泛型个人感觉是一种很好的编程习惯。

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？

在给出ADT的规约后就开始编写测试用例的TDD编程策略能够不受既有代码的干扰，完全就函数的功能编写测试用例。其好处是显而易见的：模块化编程，提高正确率与效率。在Lab 2中接触到这种编程方式初上手有些不适应，还需要多加练习。

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？

可以减少工作量，提高效率与编程体验。

1. P3要求你从0开始设计ADT并使用它们完成一个具体应用，你是否已适应从具体应用场景到ADT的“抽象映射”？相比起P1给出了ADT非常明确的rep和方法、ADT之间的逻辑关系，P3要求你自主设计这些内容，你的感受如何？

Lab 2没要求做P3。

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？

可以明确编程思路，方便自己也方便其他代码的阅读者，同时能够时刻提醒自己注意规范，检查是否有表示泄露。个人觉得这样写有些繁琐，但确实很有必要。

1. 关于本实验的工作量、难度、deadline。

本实验的工作量在做完以后回头看，其实真正的工作并不算多，就难度而言也并不算特别大，但deadline相对较紧，这是因为在MIT的页面上题目要求以英文形式介绍，写的并不算特别清晰，要点分布分散，造成走了许多弯路，浪费了很多时间。

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

课程很好，通过课程与实验的推进，编程能力有了质的变化，开始从整体的角度来思考程序的编写。老师讲的很好，但讲课节奏较快，望以后增加学时。