

**2022年春季学期  
计算学部《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 李世轩 |
| 学号 | 120L022109 |
| 班号 | 2003007 |
| 电子邮件 | 1146887979@qq.com |
| 手机号码 | 15234117960 |

**目录**

[1 实验目标概述 1](#_Toc103807050)

[2 实验环境配置 1](#_Toc103807051)

[3 实验过程 2](#_Toc103807052)

[3.1 Poetic Walks 2](#_Toc103807053)

[3.1.1 Get the code and prepare Git repository 2](#_Toc103807054)

[3.1.2 Problem 1: Test Graph <String> 3](#_Toc103807055)

[3.1.3 Problem 2: Implement Graph <String> 4](#_Toc103807056)

[3.1.3.1 Implement ConcreteEdgesGraph 4](#_Toc103807057)

[3.1.3.2 Implement ConcreteVerticesGraph 7](#_Toc103807058)

[3.1.4 Problem 3: Implement generic Graph<L> 10](#_Toc103807059)

[3.1.4.1 Make the implementations generic 10](#_Toc103807060)

[3.1.4.2 Implement Graph.empty() 12](#_Toc103807061)

[3.1.5 Problem 4: Poetic walks 13](#_Toc103807062)

[3.1.5.1 Test GraphPoet 13](#_Toc103807063)

[3.1.5.2 Implement GraphPoet 14](#_Toc103807064)

[3.1.5.3 Graph poetry slam 15](#_Toc103807065)

[3.1.6 使用Eclemma检查测试的代码覆盖度 15](#_Toc103807066)

[3.1.7 Before you’re done 16](#_Toc103807067)

[3.2 Re-implement the Social Network in Lab1 17](#_Toc103807068)

[3.2.1 FriendshipGraph类 17](#_Toc103807069)

[3.2.2 Person类 18](#_Toc103807070)

[3.2.3 客户端main() 18](#_Toc103807071)

[3.2.4 测试用例 19](#_Toc103807072)

[3.2.5 提交至Git仓库 20](#_Toc103807073)

[4 实验进度记录 21](#_Toc103807074)

[5 实验过程中遇到的困难与解决途径 22](#_Toc103807075)

[6 实验过程中收获的经验、教训、感想 22](#_Toc103807076)

[6.1 实验过程中收获的经验和教训（必答） 22](#_Toc103807077)

[6.2 针对以下方面的感受（必答） 22](#_Toc103807078)

# 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

简要陈述你配置本次实验所需环境的过程，必要时可以给出屏幕截图。

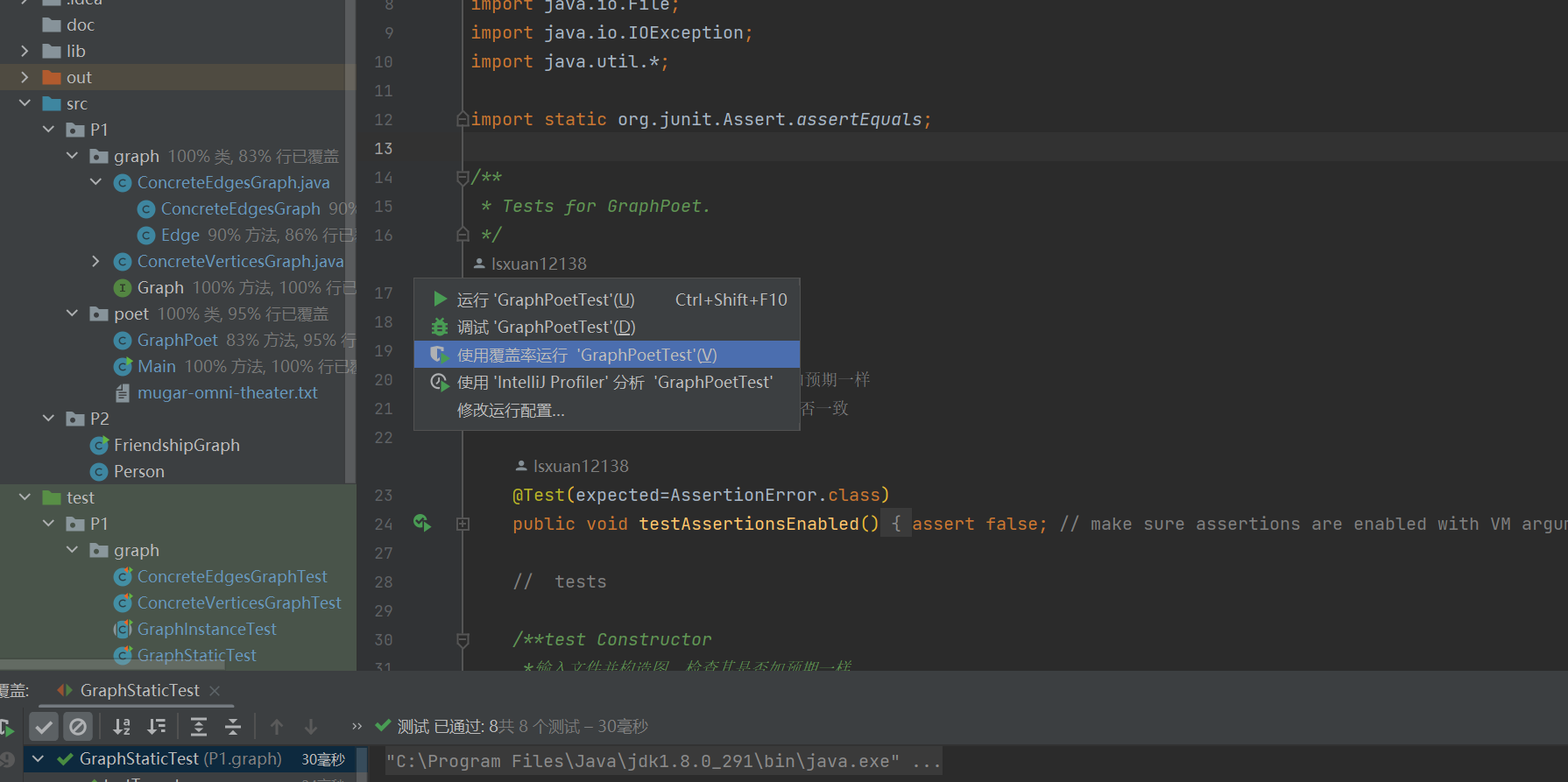
特别是要记录配置过程中遇到的问题和困难，以及如何解决的。

大部分配置在前一次实验中以及配置过，这里不再赘述。

这里简要说明idea中进行代码覆盖度测试的方法

在代码左侧的绿色三角，也就是代表运行的地方，右键即可打开菜单，如图

点击使用覆盖率运行，即可对该方法覆盖到的类，方法，代码行等进行查看



在这里给出你的GitHub Lab2仓库的URL地址（Lab2-学号）。

https://github.com/ComputerScienceHIT/HIT-Lab2-120L022109.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

在这个任务总需要处理以下几个任务

-为接口Graph<L>编写通用的测试用例

-实现接口的实现类ConcreteEdgesGraph<String>，并编写相应的测试方法

-实现接口的实现类ConcreteVerticesGraph<String>，并编写相应的测试方法

-将ConcreteEdgesGraph<String> 和ConcreteVerticesGraph<String>修改为接受泛型L的类

-编写诗意漫步的测试方法

-实现诗意漫步中的方法

### Get the code and prepare Git repository

如何从GitHub获取该任务的代码、在本地创建git仓库、使用git管理本地开发。

如何从GitHub获取该任务的代码

在Git Bash中输入

git clone "仓库地址"

即可从GitHub

在本地创建git仓库

进入某一目录后，使用

git init

即可在本地创建git仓库

使用git管理本地开发。

添加到缓存区

git add <filename>

git add \*

提交

git commit -m “代码提交信息”

### Problem 1: Test Graph <String>

分别为Graph中定义的接口设计测试方法

testAdd方法

-测试添加一个普通的节点的情况

-测试添加一个重复节点的情况

testSet方法

-测试添加正常的权值非零的边

-测试添加节点不存在的边

-测试改变权值

-测试删去存在的边

-测试删去不存在的边

testRemove方法

-删去一个存在的节点

-删去一个不存在的节点

testVertices方法

-测试一个构建好的图

-删除节点后再次测试

-增加节点后再次测试

testSources方法

-测试一个正常情况

-测试删去边的情况

-测试增加边的情况

testTarget方法

-测试一个正常情况

-测试删去边的情况

-测试增加边的情况

### Problem 2: Implement Graph <String>

#### Implement ConcreteEdgesGraph

首先实现Edge类

根据要求Edge是一个不可变的类

使用private final来修饰其中所有关键字，且其中所有类型均是不可变的类型，如图3-1所示。

因为需要修改权值的操作，我们编写一个setWeight方法，但是在其中返回一个新的修改权值后的副本，如图3-2所示。

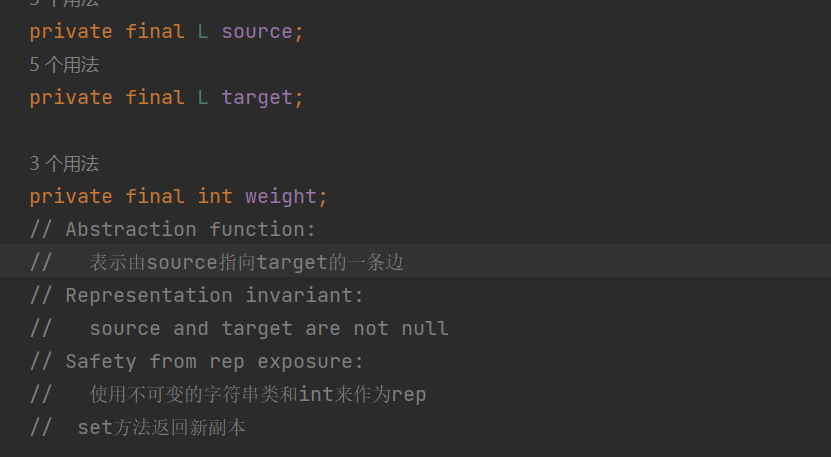


图3-1

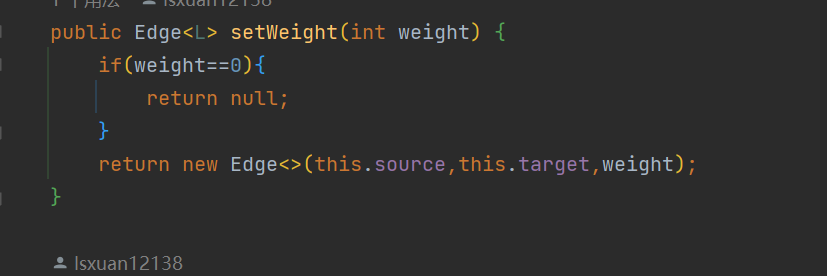


图3-2

接下来实现ConcreteEdgesGraph

其中的rep已经规定好，我们不再修改，图3-3中是相对应的RI和AF

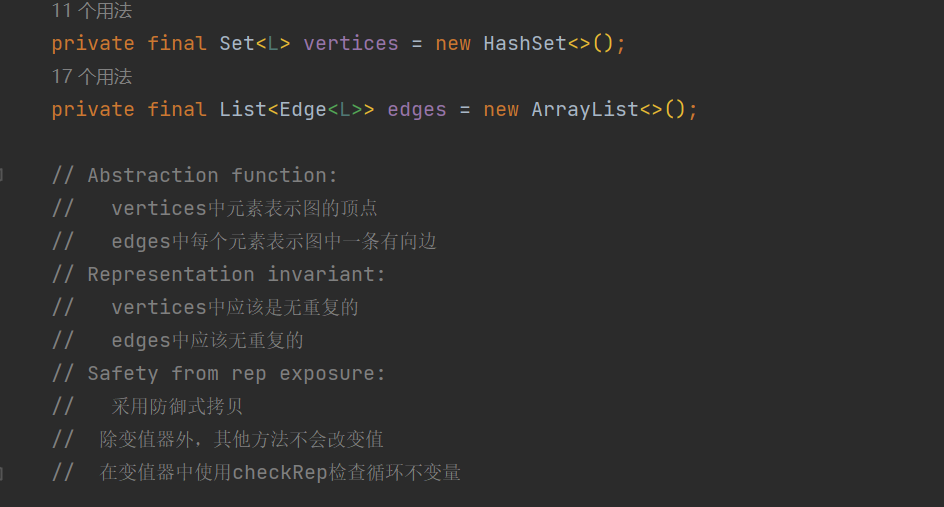


图3-3

add方法public boolean add(L vertex)

若在图的节点集vertices中包含vertex，则直接返回false

若不包含，则向vertices中增加vertex

set方法public int set(L source, L target, int weight)

首先对异常情况对象处理

若source 或者target为null则抛出异常

若权值weight小于0，则爆出异常

然后进行处理，分为两种情况

若图中已经包含边则对边的权值进行修改，使用setweight方法，得到新的边edge并删去原来的边

若此时weight是0，那么edge为null，此时，不进行操作，返回

若weight>0，那么把新得到的edge加入的边集edges中，此时，若source 或者target不再顶点集中，则添加进去

若图中没有相同的边，

若weight=0，不做修改，直接返回

若weight>0，向edges中添加新的边此时，若source 或者target不再顶点集中，则添加进去

remove方法public boolean remove(L vertex)

首先对异常情况判断

若vertex为空则抛出异常

若顶点集中不包含vertex，直接返回false

然后进行处理

将顶点集中的vertex直接移除

然后遍历边集，若其中某一条边的source或者target与vertex相等，将其移除

返回true

vertices方法public Set<L> vertices()

直接拷贝顶点集并返回

sources方法public Map<L, Integer> sources(L target)

首先对异常情况判断

若target为空则抛出异常

然后进行处理

创建一个新的map用来存放结果

遍历边集，若边的终点与target相同，将其放入map

返回map

targets方法public Map<L, Integer> targets(L source)

首先对异常情况判断

若source为空则抛出异常

然后进行处理

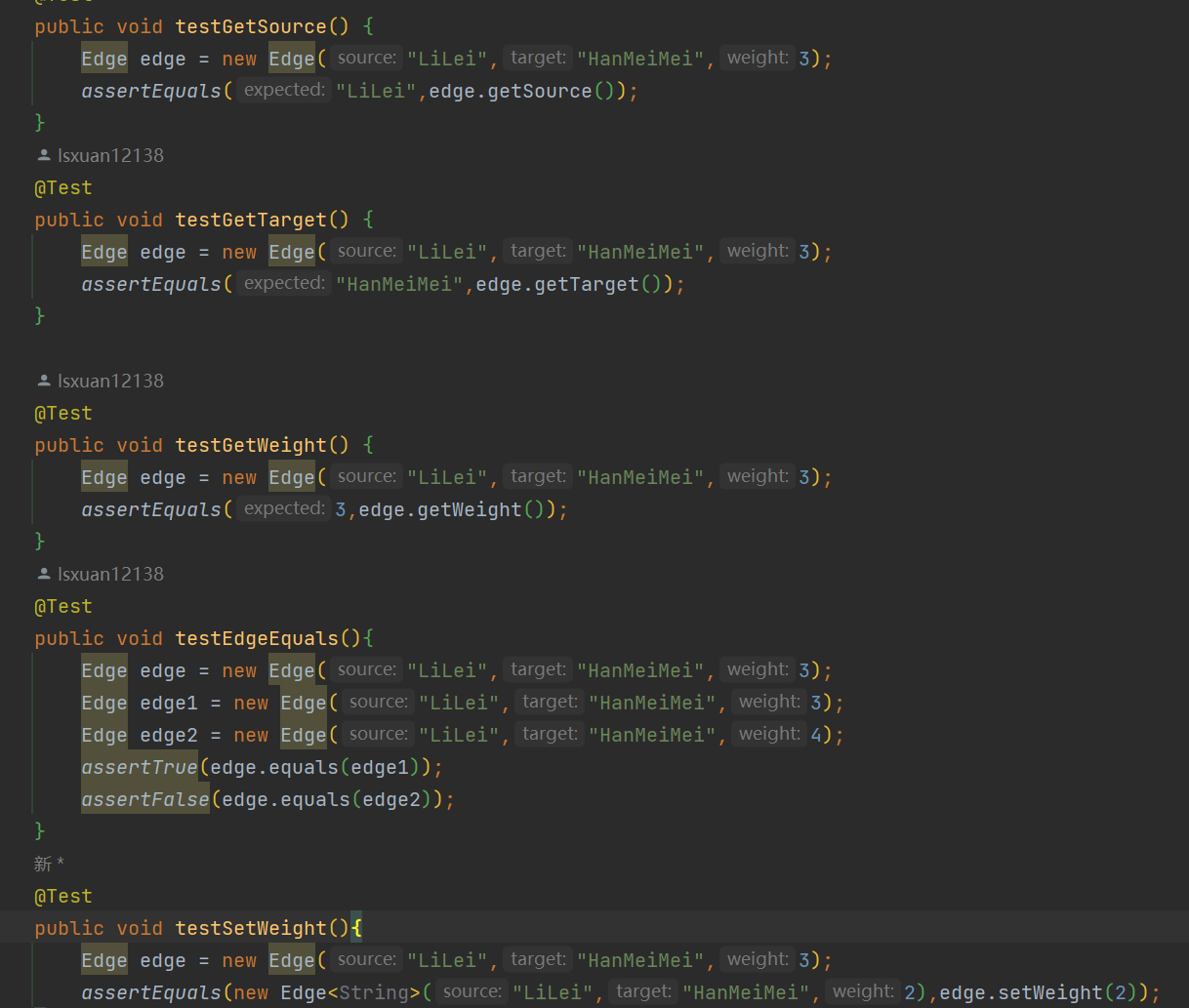
创建一个新的map用来存放结果

遍历边集，若边的起点与source相同，将其放入map

返回map

测试ConcreteEdgesGraph

首先测试Edge的各个方法



然后对ConcreteEdgesGraph.toString()进行测试

-测试空图

-测试有节点但没有边的图

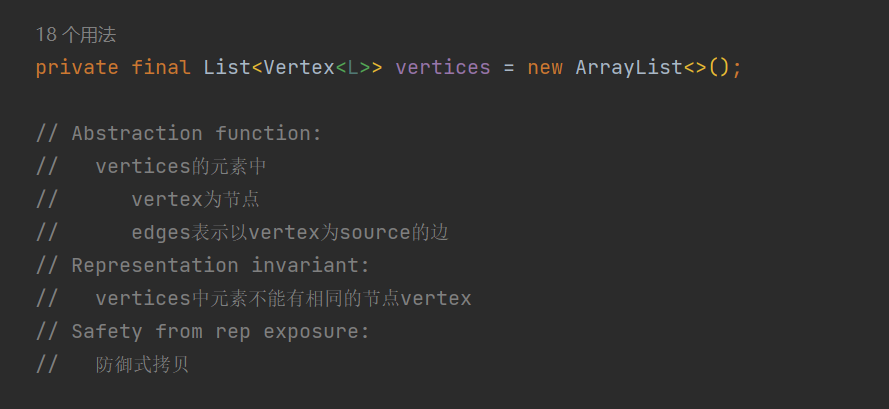
-测试有边有点的图



#### Implement ConcreteVerticesGraph

实现Vertex类

首先先看ConcreteVerticesGraph中的rep



因为ConcreteVerticesGraph中只有一个Vertex的list

那么显然Vertex中应该同时有顶点与边的信息

我们使用String vertex来表示节点名称，用Map<String,Integer> edges来表示以vertex为起点的所有边的终点和对应的权值



这里直接使用常规的setter与getter不做特殊的修改

然后实现ConcreteVerticesGraph

add方法public boolean add(L vertex)

遍历图的点集vertices若其中包含vertex，则直接返回false

若不包含，则向vertices中增加vertex

set方法public int set(L source, L target, int weight)

首先对异常情况对象处理

若source 或者target为null则抛出异常

若权值weight小于0，则爆出异常

然后进行处理，分为两种情况

若顶点集中有source

遍历source的每一条边

若找到对应边，

若weight>0，则修改边，返回

若weight=0，则删除边，返回

若找不到对应边，查看点集中是否存target

若有

weight=0，则不处理

若weight>0，则向source的map中添加对应边

返回改变之前的权值

若没有

添加节点target，并添加对应边，返回0

若顶点集中没有source

若weight为0，则不处理，直接返回0

若weight>0，则在vertices中添加节点及对应的边

遍历vertices寻找target，若其中没有则添加节点target，返回0

remove方法public boolean remove(L vertex)

首先对异常情况判断

若vertex为空则抛出异常

遍历顶点集中

若vertices中无vertex，直接返回false

若有将顶点集中的vertex直接移除

遍历vertices，若其中某一个顶点的edges中包含key：target，将其移除

返回true

vertices方法public Set<L> vertices()

new一个新的set

遍历vertices将每一个节点的vertex放入set中

返回set

sources方法public Map<L, Integer> sources(L target)

首先对异常情况判断

若target为空则抛出异常

然后进行处理

创建一个新的map用来存放结果

遍历边集，若边的终点与target相同，将其放入map

返回map

targets方法public Map<L, Integer> targets(L source)

首先对异常情况判断

若source为空则抛出异常

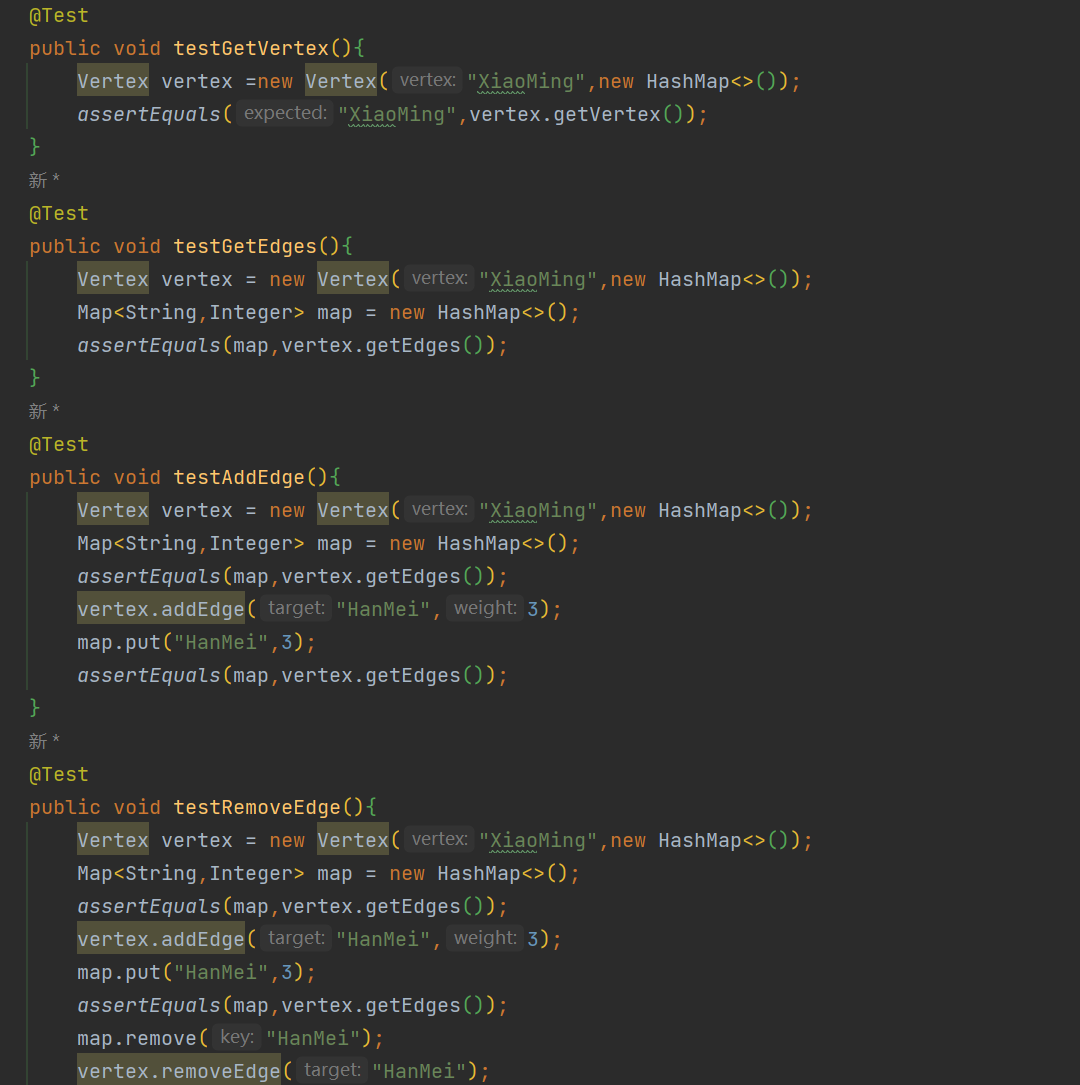
然后进行处理

创建一个新的map用来存放结果

遍历边集vertices，若边的起点与source相同，返回该vertex的edges的副本

若未找到，返回空的map

测试ConcreteVerticesGraph



测试ConcreteVerticesGraph.toString()



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

#### Make the implementations generic

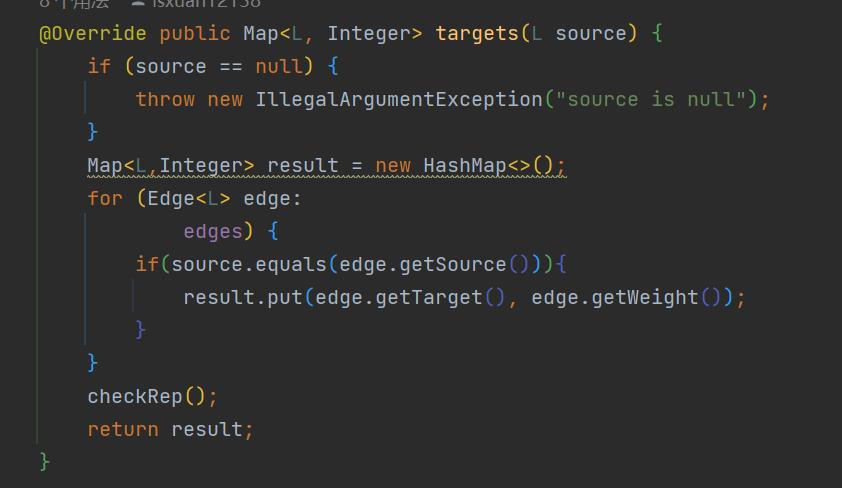
这一步只要将ConcreteEdgesGraph和ConcreteVerticesGraph在类名后增加中<L>，并把其中的String都换成泛型L，也就是

将Edge换成Edge<L>

将List<Edge>换成List<Edge<L>>

等等

这里简单展示几个







#### Implement Graph.empty()

**在这个任务中需要完成两个部分**

**-完成**Graph.empty()

-使用其他不可变类型完成GraphStaticTest

实现Graph.empty()

只要任选一个实现类，new对象并返回即可

这里我选择了ConcreteEdgesGraph



完成GraphStaticTest

这里的基本测试策略与GraphInstanceTest相同，故不再重复

只需要将其中的String换成其他的不可变类型即可，这里选择了以下几个类

Integer，Long，Character，Byte，BigDecimal，BigInteger

这里举一个例子



### Problem 4: Poetic walks

在该任务中需要完成两个子任务

-完成编写对GraphPoet中方法的测试

-根据给出的要求实现GraphPoet中的方法

#### Test GraphPoet

这个任务需要完成编写对GraphPoet中方法的测试

测试GraphPoet中的构造方法

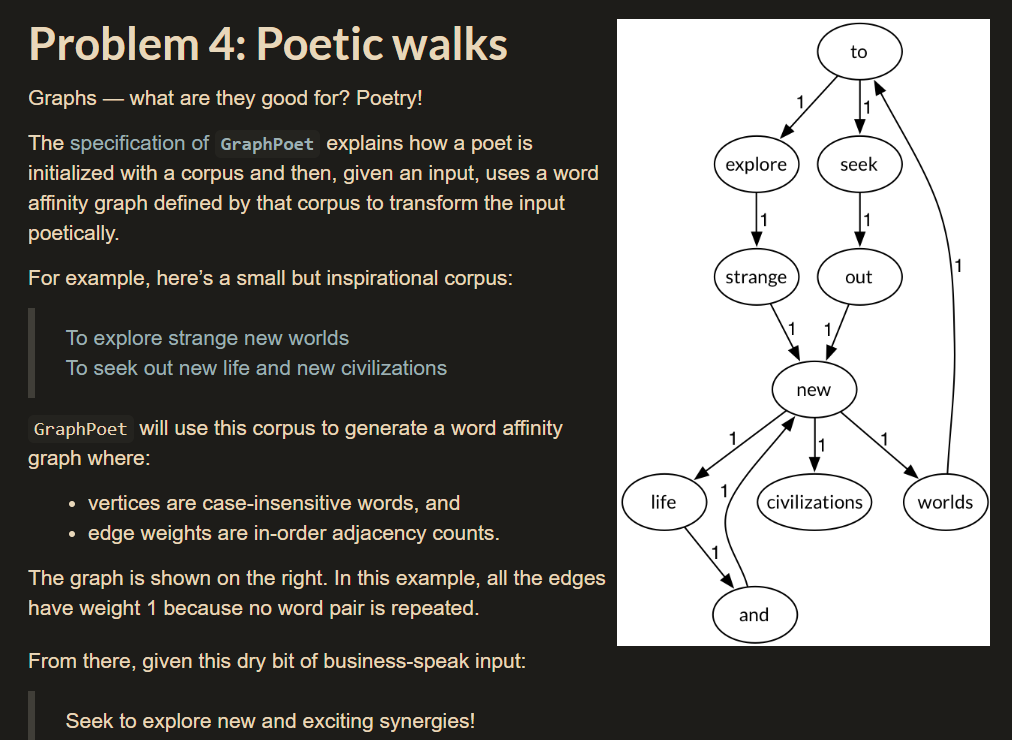
public void testConstructor()

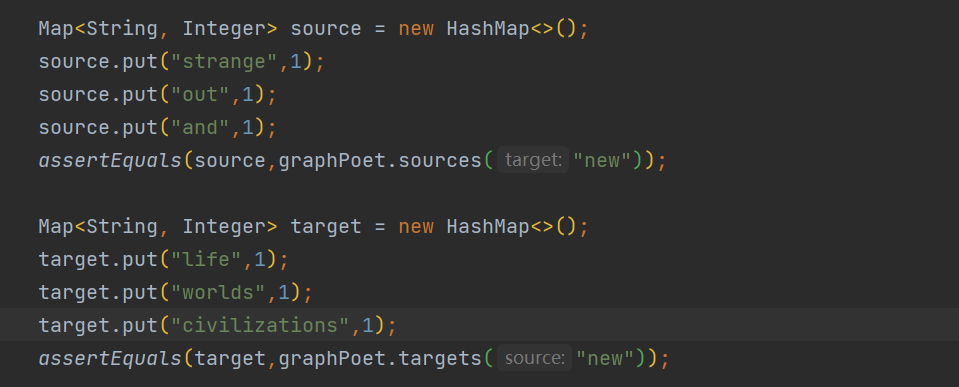
输入文件并构造图，检查其是否如预期一样

这里举一个例子



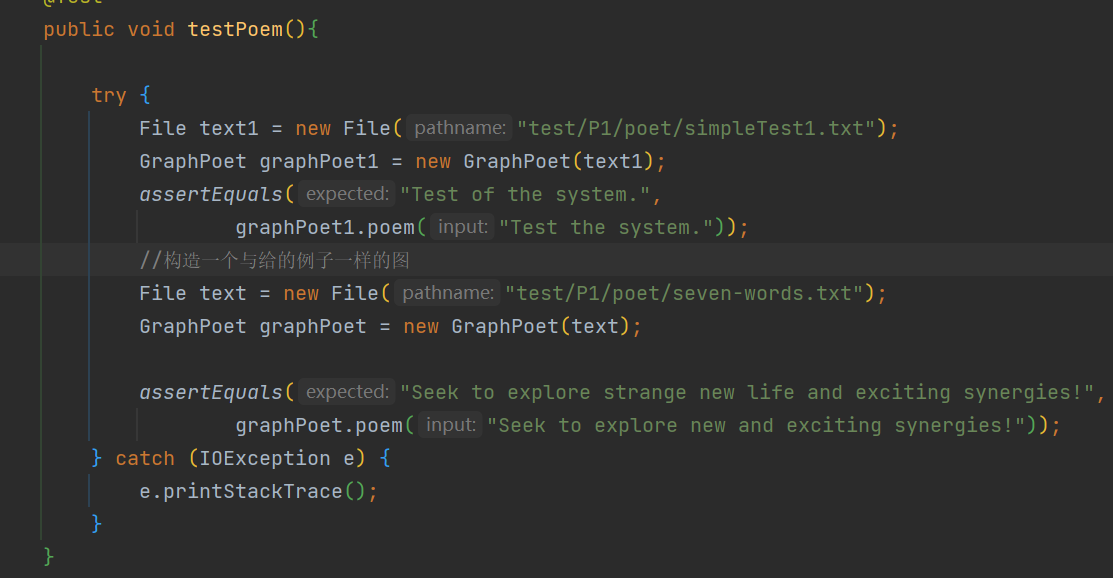
另外，构造一个与给出例子一样的图来进行测试





测试GraphPoet中的poem方法

输入input，检查输出的poet是否一致



#### Implement GraphPoet

构造方法

public GraphPoet(File corpus) throws IOException

使用Scanner作为文件输入流

读入文件corpus，读入每一行并处理

将读入的一行以空格分隔，并放入数组

遍历数组

若我们处理的是第一行，那么直接进行处理

先设置

set(line[i].toLowerCase(), line[i + 1].toLowerCase(),1);

若返回的权值不为0，则重新设置set(line[i].toLowerCase(), line[i + 1].toLowerCase(), beforeWeight + 1);

若处理的不是第一行

对上一行的最后一个单词，和这一行最后一个词，进行相同处理

再进行同上处理

poem方法

public String poem(String input)

创建一个StringBuilder存放结果

对输入用空格进行分割放入数组words中

遍历数组

先将数组中的元素words[i]放入结果中

分别获取words[i]在图中的targets和words[i+1]的sources

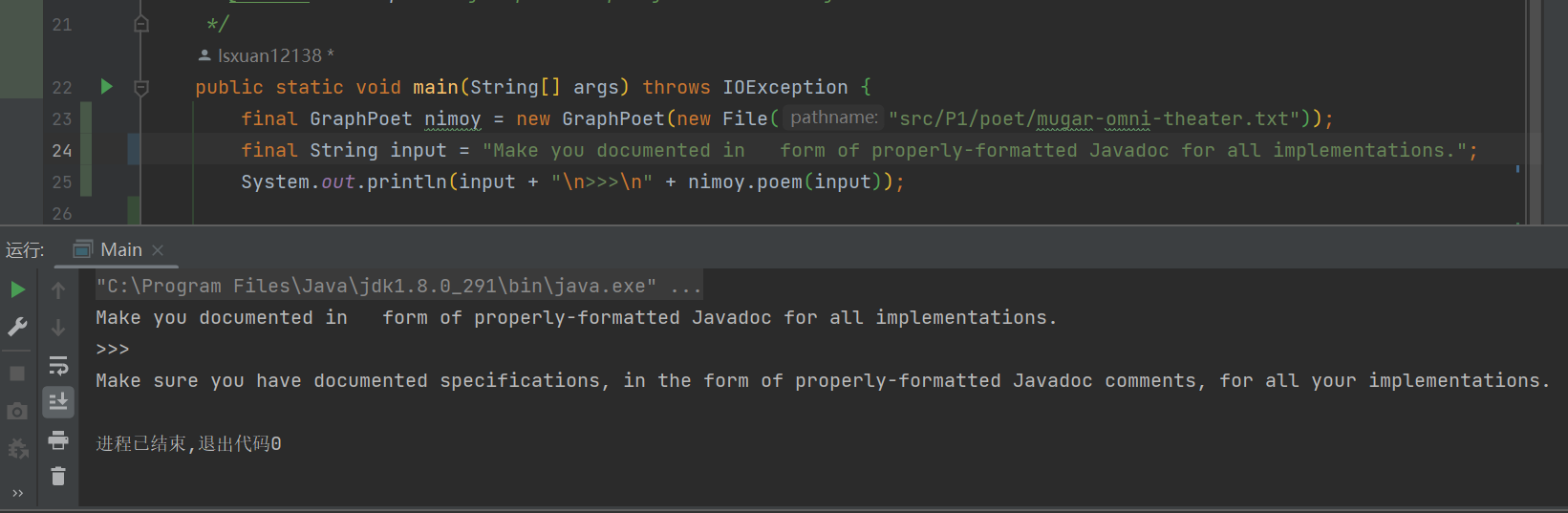
遍历targets，获取targets和sources共有的且路径权值最大的元素

将得到的元素放入结果中

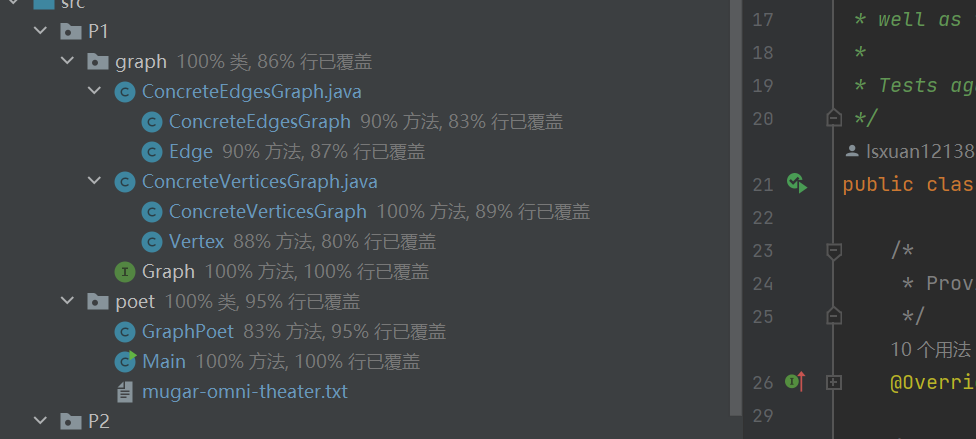
将数组中最后一个元素放入结果

返回结果

#### Graph poetry slam



### 使用Eclemma检查测试的代码覆盖度



### 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。

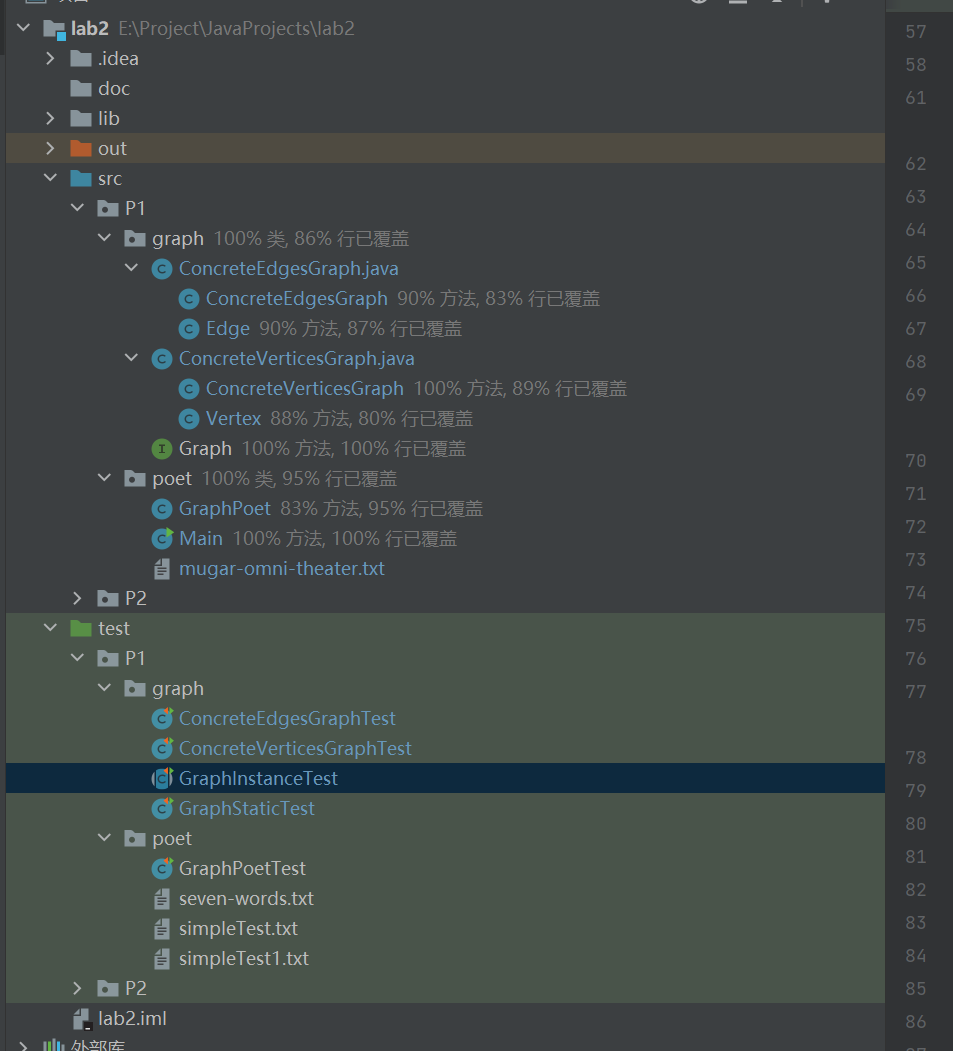
如何通过Git提交当前版本到GitHub上你的Lab2仓库。

使用指令

git push origin master

其中的master可以换为任何其他分支

在这里给出你的项目的目录结构树状示意图。

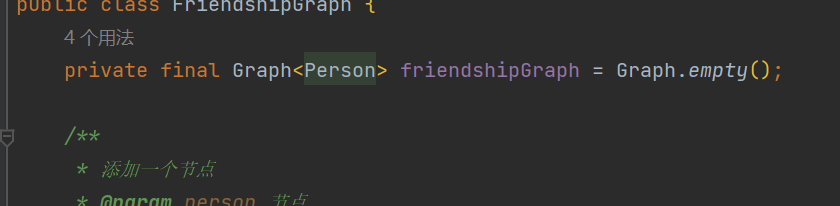


## Re-implement the Social Network in Lab1

该任务要求使用P1中实现的接口Graph<L>来重新实现在lab1中的Social Network

### FriendshipGraph类

使用Graph<Person>作为FriendshipGraph的rep



addVertex方法public boolean addVertex(Person person)

若图的点集中已经包含person，则输出信息返回false

若不包含，则加入person，返回true

addEdge方法public boolean addEdge(Person source,Person target)

先对异常情况处理

若source或者target为null，抛出异常

若source.equals(target)，输出不能添加自己为好友，返回false

若friendshipGraph.set(source,target,1)的返回值不为0，则输出已经是好友了，返回false

若返回值为0，返回true

getDistance 方法public int getDistance(Person source,Person target)

先对异常情况处理

若source或者target为null，抛出异常

若source与target相同，返回0

接下来计算不同的两点间的最短路径长度

进行广度优先遍历，这里做出如下处理

设置标记，表示读完一层

若标记为true读完一层，添加空节点，作为记录，将标记设为false

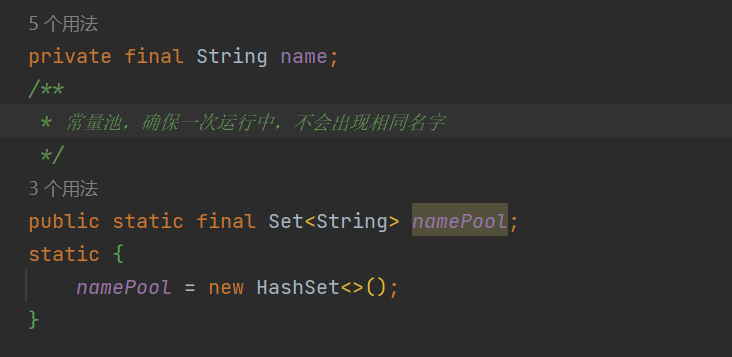
若读到空节点，表明读完一层，设置标记为true

若找到target节点，返回

若在广度优先遍历后未返回

说明两个节点未连接，返回-1

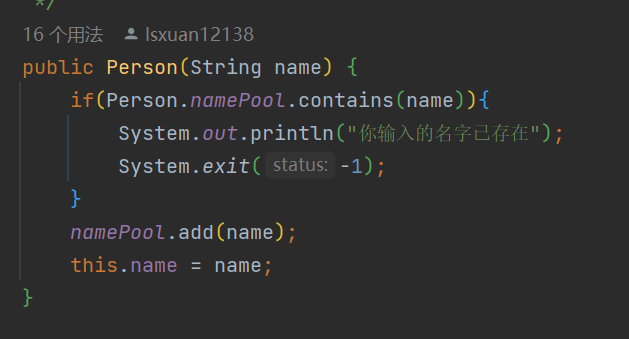
### Person类



使用String name作为Person的rep

设置一个静态的属性Set<String> namePool作为常量池，确保一次运行中，不会出现相同名字

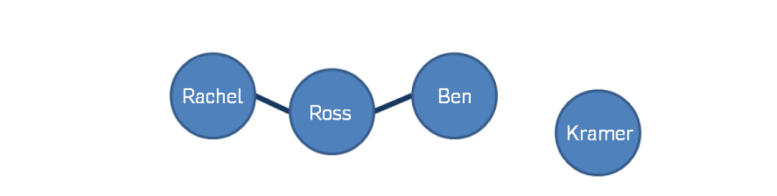
在构造方法中进行判断



实现相应的getter方法

### 客户端main()

使用使用指导书中的代码。构造这样的图



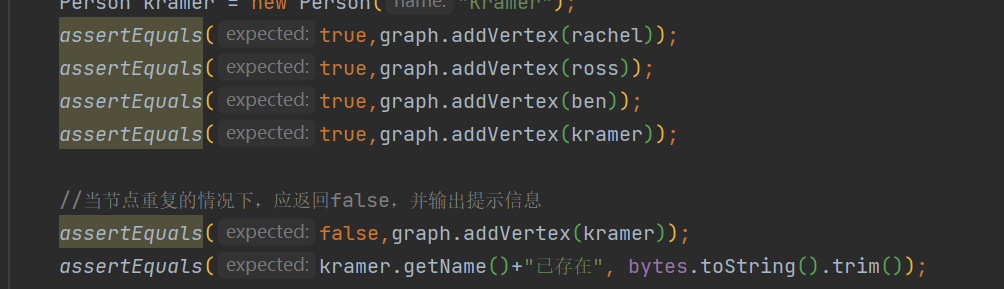


### 测试用例

3.2.4.1 测试testAddVertex

测试正常添加节点的返回值是否正确

测试当节点重复添加时的反应

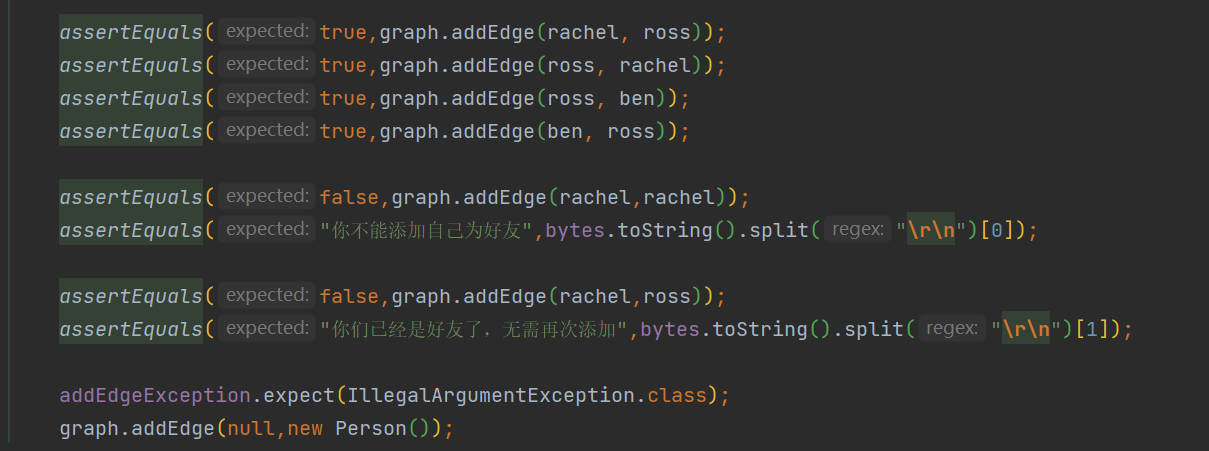


3.2.4.2 测试testAddEdge

测试正常添加边的返回值是否正确

测试当p1，p2相同时，程序的反应

测试重复添加边时的反应



3.2.4.3 测试testGetDistance

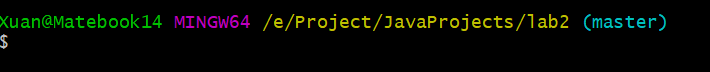
测试当两个点间只有一条有向边的反应

测试两个点间有正常的两条边的结果是否正确

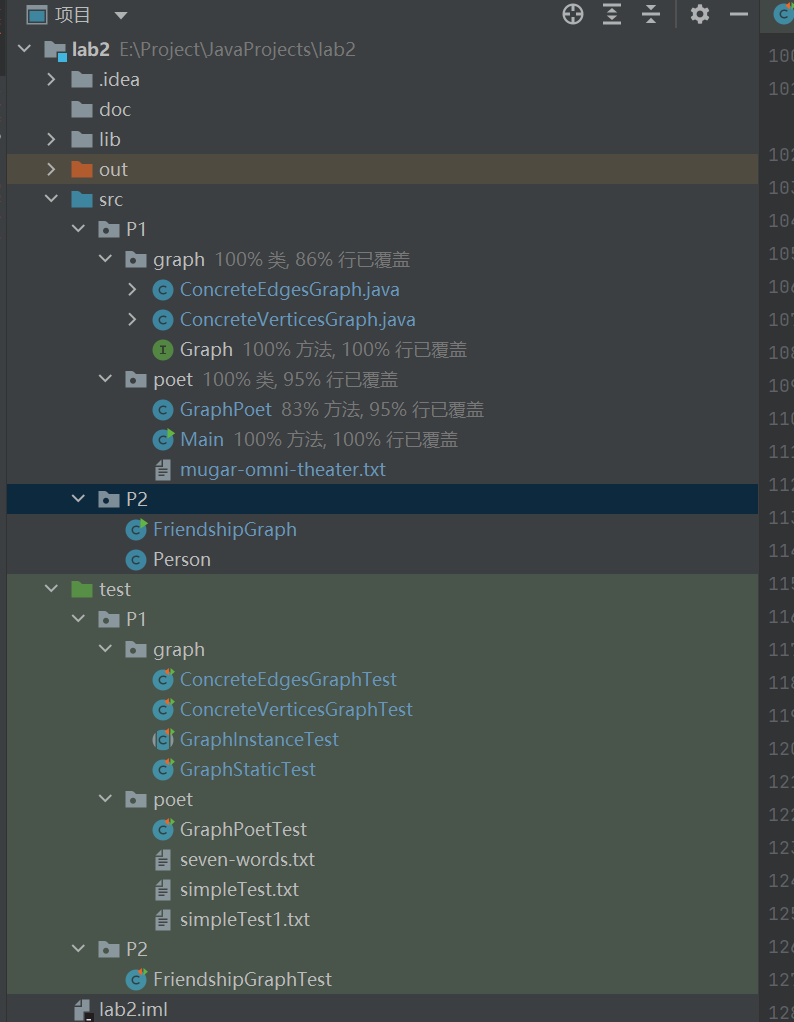


### 提交至Git仓库

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。



# 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 |
| 2022-5-11 | 16:00-17:30 | 完成P1 problem1 | 完成 |
| 2022-5-14 | 18:00-20:30 | 完成P1 problem2.1 | 完成 |
| 2022-5-14 | 20:00-22:30 | 完成P1 problem2 | 完成 |
| 2022-5-15 | 17:30-18:30 | 修改P1 problem1&2 | 完成 |
| 2022-5-15 | 18:30-19:00 | 完成P1 problem3.1 | 完成 |
| 2022-5-15 | 19:00-20:00 | 完成P1 problem3 | 完成 |
| 2022-5-15 | 20:20-21:15 | 完成P1 problem4.1 | 完成 |
| 2022-5-15 | 21:20-22:25 | 完成P1 problem4 | 完成 |
| 2022-5-16 | 15:45-16:15 | 修改P1 | 完成 |
| 2022-5-16 | 16:20-17:05 | 完成P2 | 完成 |
| 2022-5-16 | 17:05-17:30 | 修改P1 problem4.2 | 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 |
| 对于Vertex类中rep的设计 | 仔细观察ConcreteVerticesGraph的rep后，自行解决  因为ConcreteVerticesGraph中只有一个Vertex的list  那么显然Vertex中应该同时有顶点与边的信息  我们使用String vertex来表示节点名称，用Map<String,Integer> edges来表示以vertex为起点的所有边的终点和对应的权值 |
| GraphPoet的构造方法中，因为以行为单元来处理，所以上一行的最后一个单词和下一行第一个单词的连接，一开被忽略 | 重新设计方法  若我们处理的是第一行，那么直接进行处理  先设置  set(line[i].toLowerCase(), line[i + 1].toLowerCase(),1);  若返回的权值不为0，则重新设置set(line[i].toLowerCase(), line[i + 1].toLowerCase(), beforeWeight + 1);  若处理的不是第一行  对上一行的最后一个单词，和这一行最后一个词，进行相同处理 |
|  |  |

# 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（必答）

初步学会设计 ADT 规约（pre-condition、post-condition）

了解了ADT 的泛型化；

学会使用idea评估测试的覆盖度；

体会到ADT的复用带来的好处；

在测试代码中，写 testing strategy 并据此设计测试用例。

## 针对以下方面的感受（必答）

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，对于ADT的设计者而言，是一个相当困难的过程，我们要设计一整套对应的操作接口，并进行实现，在这个过程中，还有考虑到各种各样的异常情况

对应ADT的使用者而言，这种方式会非常的便捷，我们可以选用第三方开发的类库来为自己的程序做支撑，而不必对所有的操作都自己编写

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

差别不大

使用泛型在编程过程中只能使用一些通用的方法，除此之外，其他的功能通常都需要自己实现

不使用泛型，针对一些类型，也许会有一些更加便捷的操作

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？

使用这种方式，我们对ADT的测试会更加全面且天马行空，不会因为实现ADT时的某些想法造成先入为主

目前还算适应

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？

这种复用可以大大节省程序员开发需要ADT支撑的功能时的时间，不必将大量的时间花费在太细节的具体实现上

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？

这些工作可以在编程过程中为我们提供指导和约束，在设计好后，我们的所有工作都必须在这个前提下进行，可以减少思考的可能性，也减少出错的可能

1. 关于本实验的工作量、难度、deadline。

工作量比较大

难度适中

Deadline比较宽松

1. 《软件构造》课程进展到目前，你对该课程有何收获和建议？

学习到现在，对Java或者说对面向对象编程有了更深的认识

学到了一些很通用也非常优秀的编程思想