

**2022年春季学期  
计算学部《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 陈本福 |
| 学号 | 1180300526 |
| 班号 | 1903401 |
| 电子邮件 | [marlon\_chan19@outlook.com](https://www.outlook.com/?refd=account.microsoft.com&fref=home.banner.viewinbox) |
| 手机号码 | 13304515838 |

**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 2](#_Toc72249753)

[3.1.1 Get the code and prepare Git repository 2](#_Toc72249754)

[3.1.2 Problem 1: Test Graph <String> 2](#_Toc72249755)

[3.1.3 Problem 2: Implement Graph <String> 3](#_Toc72249756)

[3.1.3.1 Implement ConcreteEdgesGraph 3](#_Toc72249757)

[3.1.3.2 Implement ConcreteVerticesGraph 6](#_Toc72249758)

[3.1.4 Problem 3: Implement generic Graph<L> 10](#_Toc72249759)

[3.1.4.1 Make the implementations generic 10](#_Toc72249760)

[3.1.4.2 Implement Graph.empty() 10](#_Toc72249761)

[3.1.5 Problem 4: Poetic walks 11](#_Toc72249762)

[3.1.5.1 Test GraphPoet 11](#_Toc72249763)

[3.1.5.2 Implement GraphPoet 11](#_Toc72249764)

[3.1.5.3 Graph poetry slam 12](#_Toc72249765)

[3.1.6 使用Eclemma检查测试的代码覆盖度 13](#_Toc72249766)

[3.1.7 Before you’re done 14](#_Toc72249767)

[3.2 Re-implement the Social Network in Lab1 15](#_Toc72249768)

[3.2.1 FriendshipGraph类 15](#_Toc72249769)

[3.2.2 Person类 17](#_Toc72249770)

[3.2.3 客户端main() 17](#_Toc72249771)

[3.2.4 测试用例 18](#_Toc72249772)

[3.2.5 提交至Git仓库 18](#_Toc72249773)

[4 实验进度记录 19](#_Toc72249774)

[5 实验过程中遇到的困难与解决途径 20](#_Toc72249775)

[6 实验过程中收获的经验、教训、感想 20](#_Toc72249776)

[6.1 实验过程中收获的经验和教训 20](#_Toc72249777)

[6.2 针对以下方面的感受 20](#_Toc72249778)

# 实验目标概述

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。

1. 针对给定的应用问题，从问题描述中识别所需的 ADT；
2. 设计 ADT 规约（pre-condition、post-condition）并评估规约的质量；
3. 根据 ADT 的规约设计测试用例；
4. ADT 的泛型化；
5. 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）
6. 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）；
7. 测试 ADT 的实现并评估测试的覆盖度；
8. 使用 ADT 及其实现，为应用问题开发程序；
9. 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。

在这里给出你的GitHub Lab2仓库的URL地址（Lab2-学号）。

https://github.com/ComputerScienceHIT/HIT-Lab2-1180300526

# 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

在这里简要概述你对该任务的理解。

本次实验给出了一个图接口，要求我们建立一个边和一个点类分别继承自这个图接口，实现抽象数据型，并且完成poem。

该部分培养训练我们学会ADT的泛型化，并且学会设计ADT规约，根据规约设计ADT的多种实现，学会使用OOP实现ADT，并且也需要我们对ADT的表示泄露做出防范。

### 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。

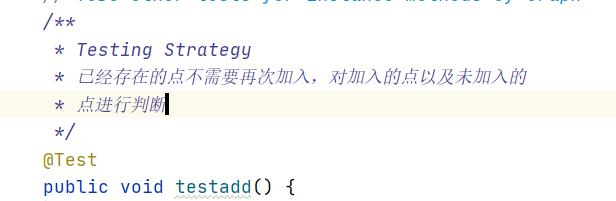
实验一已述

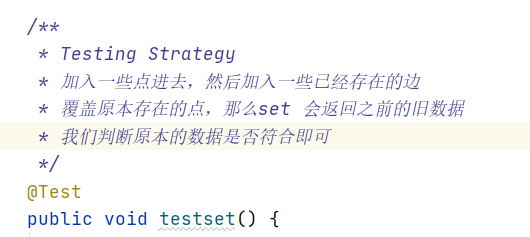
### Problem 1: Test Graph <String>

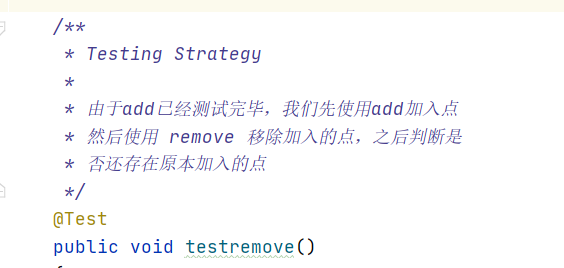
以下各部分，请按照MIT页面上相应部分的要求，逐项列出你的设计和实现思路/过程/结果。

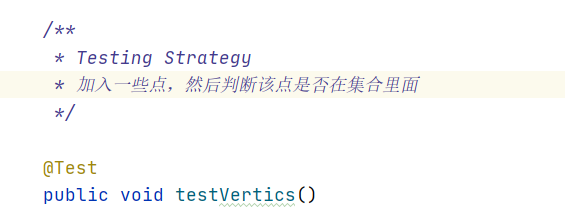
思路：针对Graph<String> 里面的方法进行逐一测试

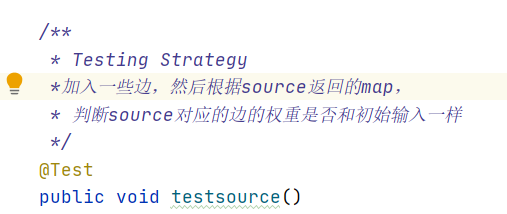
过程与结果如下：

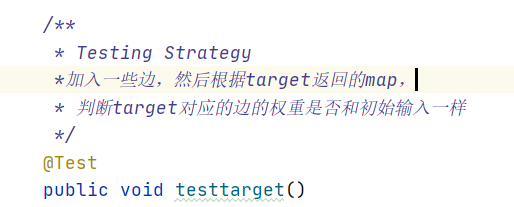












### Problem 2: Implement Graph <String>

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

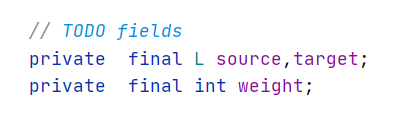
**思路：**

设计这一部分得先设计每一条边的组成，然后组合起来形成一幅图。

设计每一条边得先设计存储方式，怎么实现一些功能，比如获得source节点，获得target节点，然后在ConcreteEdgesGraph 主函数里面调用每一条边，并且结合起来。

**过程**：

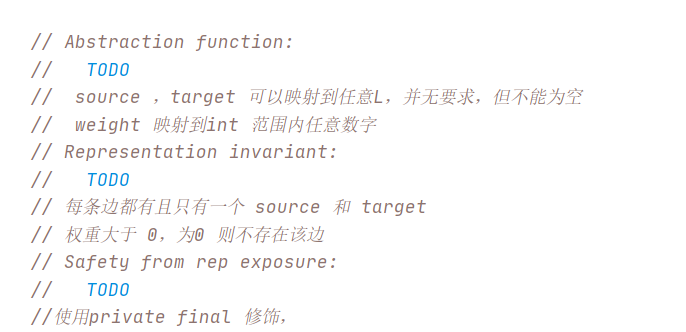
1. **实现EDGE类：**
2. 初始化边：加权有向边应该要存在应该source和target节点，以及权重，设计如下：



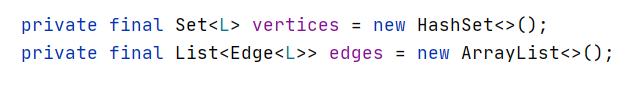
1. 函数设计：

|  |  |
| --- | --- |
| 函数名称 | 函数作用 |
| checkRep() | 检查不变性，节点不空，权重大于0 |
| Edge(L source,L target , int weight ) | 构造函数 |
| getTarget() | 获得边的target |
| getSource() | 获得边的source |
| getWeight() | 获得边的权重 |
| toString() | 打印边的信息 |

1. **AF 和 RI 以及防止表示泄露**



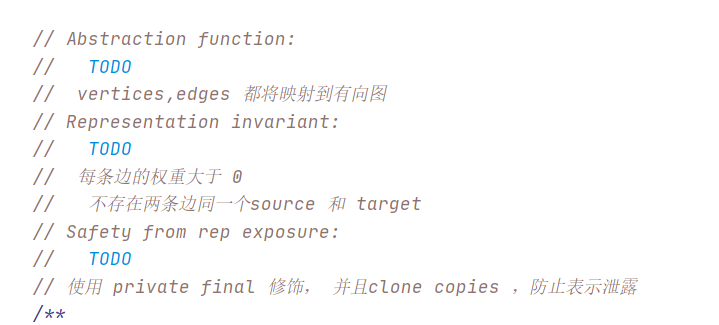
1. **实现ConcreteEdgesGraph类**
2. 初始化，存储顶点信息，以及边信息



1. 函数重写设计

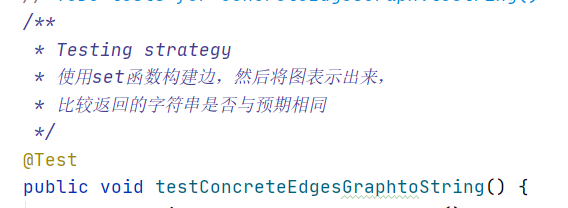
|  |  |
| --- | --- |
| 函数名称 | 函数用途 |
| ConcreteEdgesGraph | 构造函数 |
| checkRep | 检查函数 |
| add | 增加顶点 |
| Set | 增加边以及边顶点，以source为起点，target为终点，weight为权重构造边，如果存在边，那么更新边信息 |
| remove | 移除顶点，并且还需要移除与其相连的顶点的边。 |
| vertices | 返回所有顶点（需要clone保护） |
| Sources（L target） | 返回一个map存储以target为指向的source节点以及权重 |
| Target（L source） | 返回一个map存储以source为起始的target节点以及权重 |
| toString | 使用字符串返回图构成信息 |

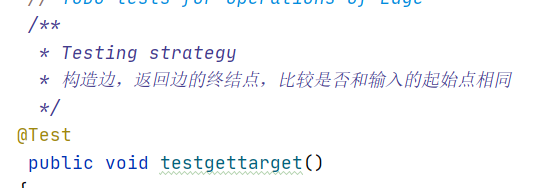
1. AF和RI 以及防止表示泄露

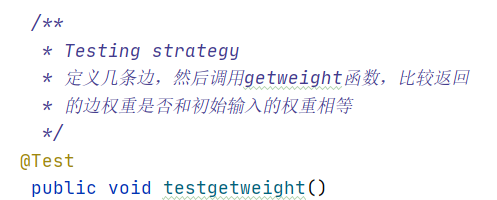


1. **测试**

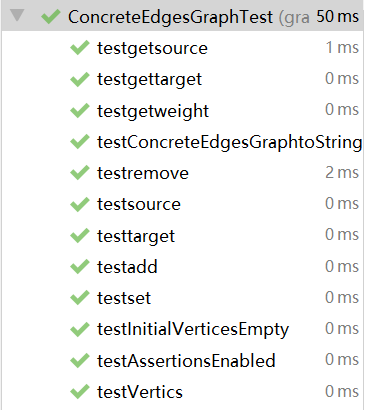
**一些测试策略**



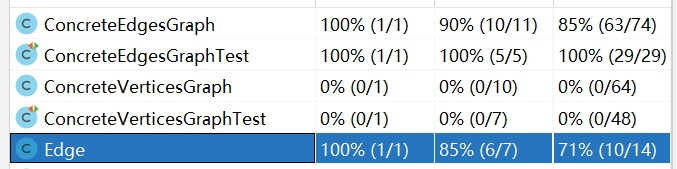




**所有测试结果**



**测试覆盖率：**

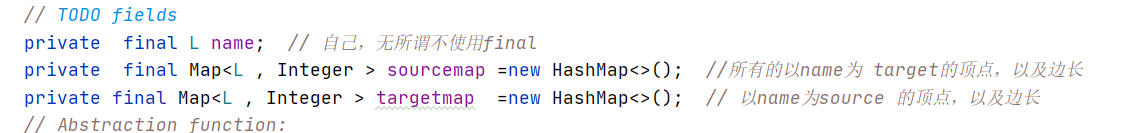


#### Implement ConcreteVerticesGraph

思路：实现该类需要先将节点信息实现，也就是先实现Vertics类，然后通过该类引用会更加方便

过程：

1. 实现 Vertics 类
2. 初始化变量信息：



1. 函数构造

|  |  |
| --- | --- |
| 函数名称 | 函数用途 |
| Vertex(L vertex) | 构造函数 |
| getName() | 获得顶点的name |
| checkRep() | 检查Rep |
| tosource(L source) | 返回某个source到name的边的权重，如果不存在则返回0. |
| totarget(L target) | 返回name到某个target的边的权重，如果不存在则返回0. |
| addsource(L source , int weight) | 给name节点添加一个source节点，以及边权重，存储在sourcemap里面，如果已经存在，并且weight大于0则更新边权重，等于0则remove该点 |
| addtarget(L target , int weight) | 给name节点添加一个target节点，以及边权重，存储在targetmap里面，如果已经存在，并且weight大于0则更新边权重，等于0则remove该点 |
| removeSource(L source ) | 移除name节点的source节点 |
| removeTarget(L target ) | 移除name节点的target节点 |
| toString() | 返回每个顶点的所有连接点，已经两者之间的权重 |

1. AF，RI以及防止表示泄露：



1. **ConcreteVerticesGraph 类**
2. 初始化变量：



1. 函数重写设计

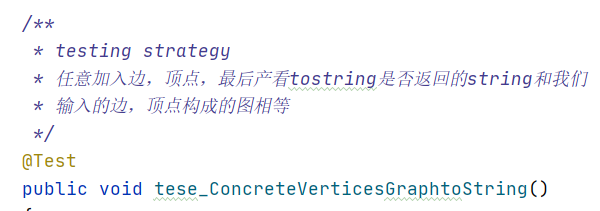
|  |  |
| --- | --- |
| 函数名称 | 函数作用 |
| ConcreteEdgesGraph() | 构造函数 |
| checkRep() | 检查函数，保持不变性 |
| add(L vertex) | 增加节点 |
| set(L source, L target, int weight) | 增加一条以source为起点，以target为终点，权重为weight的边，如果已经存在该边，那么更新权重为weight，weight==0的时候去除该边 |
| remove(L vertex) | 去除一个顶点，并且也要去除以该顶点为source，target的顶点 |
| vertices() | 返回所有顶点 |
| sources(L target) | 返回所有以target为终点的起点集合 |
| targets(L source) | 返回所有以source为起点的终点集合 |
| toString() | 返回所有图信息 |

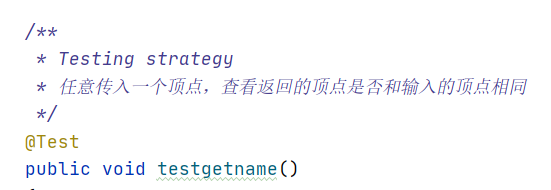
1. AF，RI和防止表示泄露

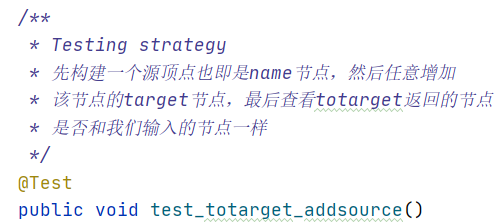


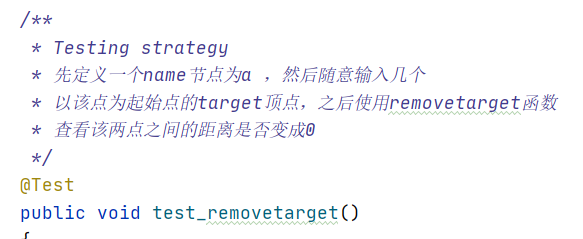
1. 测试；

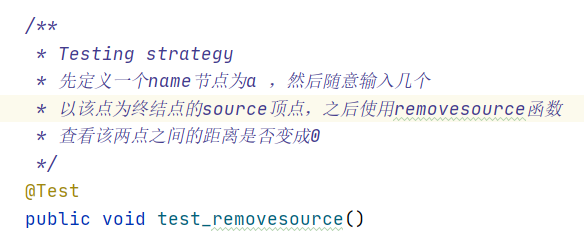
测试策略：



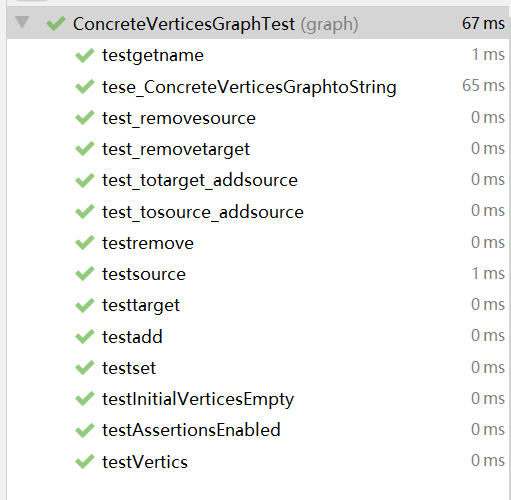




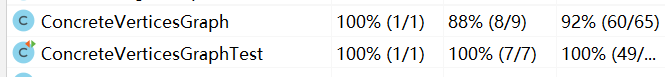




**测试结果：**



代码覆盖度测试：





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

#### Make the implementations generic

把string改为L即可实现泛型。可以根据IDEA错误提示。

#### Implement Graph.empty()

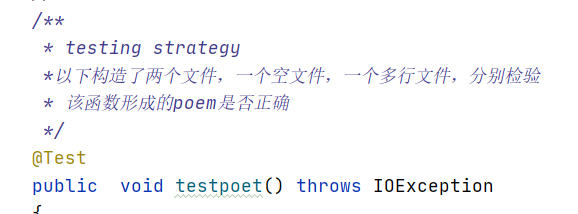
**构造一个空的图：**

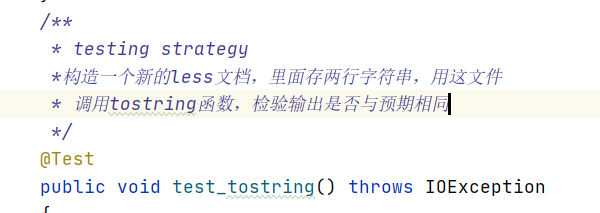


### Problem 4: Poetic walks

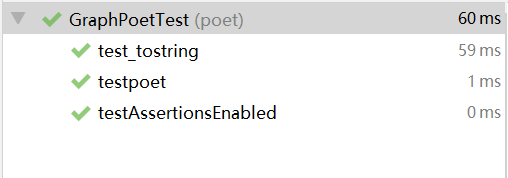
#### Test GraphPoet

**测试策略：**





**测试结果：**

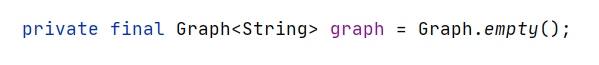


#### Implement GraphPoet

**思路：先调用以上实现的图来给字符串形成图，然后调用图生成新的字符串**

**过程：**

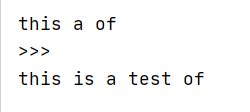
1. 初始化：



1. 函数设计：

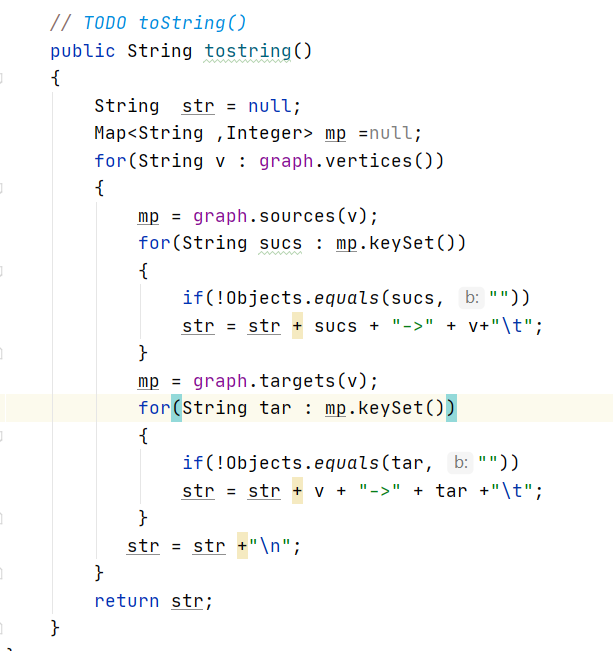
|  |  |
| --- | --- |
| 函数名称 | 函数作用 |
| GraphPoet(File corpus) | 使用该函数读取文件corpus并且构成一幅图，每个节点都是一个word，每两个word连在一起重复一遍，权重加一 |
| poem(String input) | 输入类型为字符串，通过上面函数构成的图来构成一句话，不过只能两个单词只有一个间隔，如果存在有两种情况的word那么选择其中权重较小的word节点 |

**结果：完成测试运行**

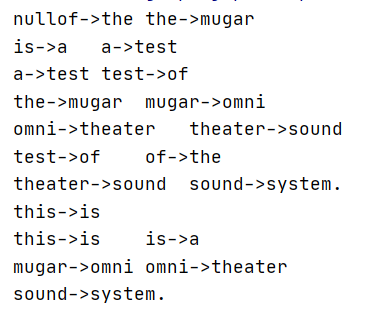


#### Graph poetry slam

在提供代码的基础上增加一个toString的输出



Main函数调用结果：



### 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。

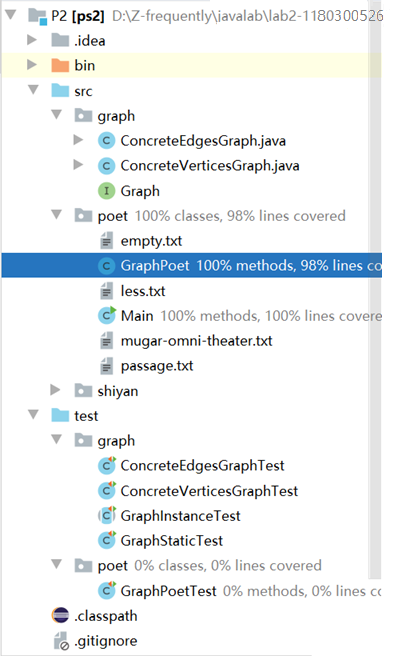
命令行：

Git add Lab2-1190201413

Git commit -m “P1”

Git push

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

在这里简要概述你对该任务的理解。

这次实验要求我们使用Poetic Walks中已经实现的Graph<L>及其两种图，边图和点图，（在这使用的是点图），实现lab1种实现的功能，并且尽可能复用lab2-P1中已经实现的方法，然后运行提供的main()和执行Lab1中的Junit测试用例，使之正常运行。

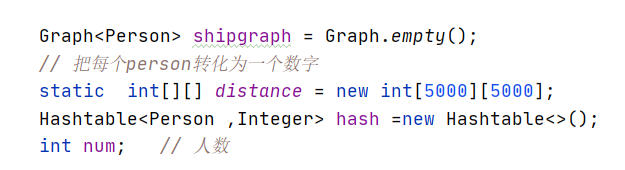
### FriendshipGraph类

给出你的设计和实现思路/过程/结果。

思路：这个类实现的功能有：增加节点、增加边，这两个函数都是直接调用P1已经实现了的功能，直接复用，然后是求距离，使用bfs搜索求距离

过程：

1. 数据初始化：

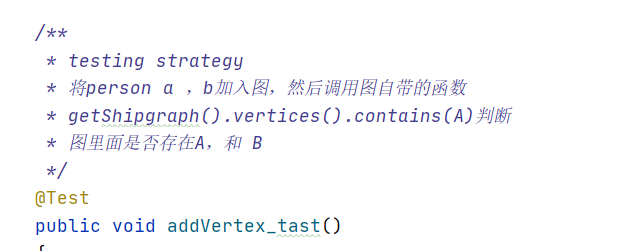


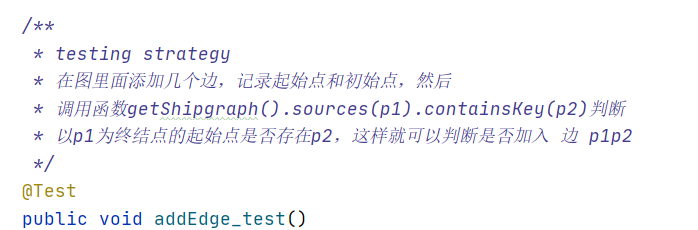
1. 函数设计：

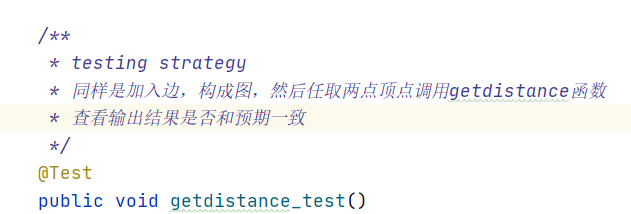
|  |  |
| --- | --- |
| 函数名称 | 函数作用 |
| getShipgraph() | 获得在该类里面建立的图 |
| addVertex(Person p1) | 增加节点 |
| addEdge(Person p1,Person p2) | 增加边，里面复用了P1中实现的set函数，weight恒为1 |
| getDistance(Person p1 , Person p2 ) | 求P1到p2 的距离 |

1. 测试

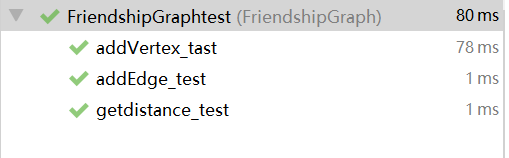
测试策略：







测试结果：



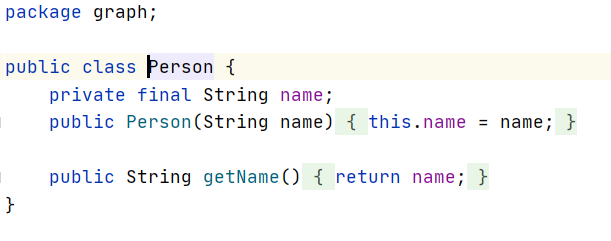
**覆盖度测试：**



### Person类

给出你的设计和实现思路/过程/结果。

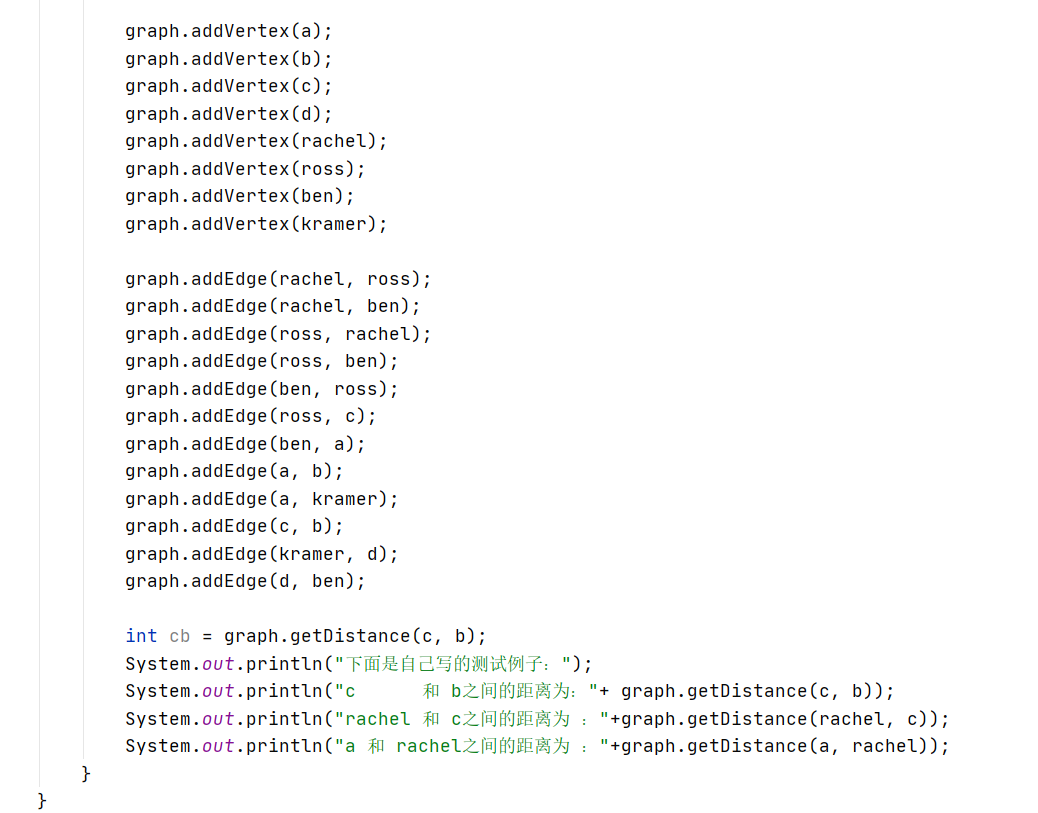
Person类实现较为简单：



### 客户端main()

给出你的设计和实现思路/过程/结果。

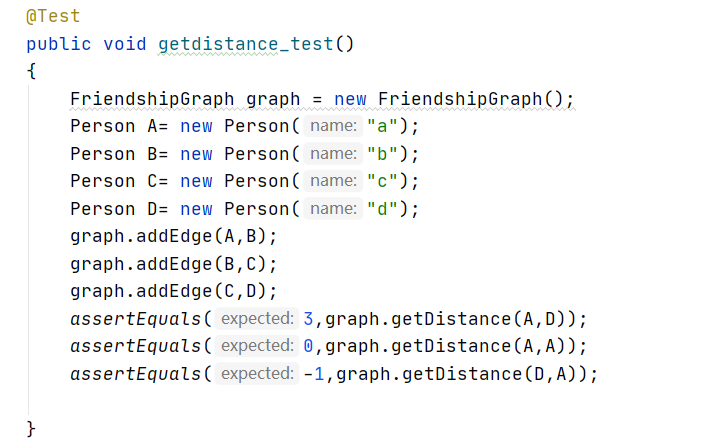
Main（）就是加入点，加入边，功能已经在FriendshipGraph类实现



### 测试用例

给出你的设计和实现思路/过程/结果。

测试通过：



### 提交至Git仓库

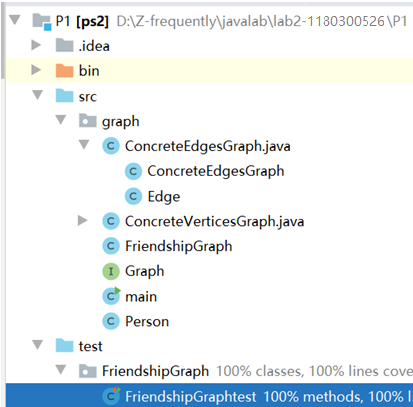
如何通过Git提交当前版本到GitHub上你的Lab3仓库。

Git add Lab2-11803000526

Git commit -m “P2”

Git push

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 |
| 5.26 | 8.00-8.30 | 完成实验手册阅读 | 未完成 |
| 5.26 | 8.30-9.00 | 完成实验手册阅读 | 未完成 |
| 5.26 | 9.00-11.00 | 完成实验手册阅读 | 完成任务 |
| 5.26 | 16.00-16.40 | 预览实验代码 | 按时完成 |
| 5.26 | 18.00-20.00 | 建立代码框架 | 按时完成 |
| 5.28 | 20.00-22.00 | 完成Problem2.1，边图类构建 | 按时完成 |
| 5.28 | 8.30-9.00 | 完成problem2.2，点图类构建 | 未完成 |
| 5.29 | 9.00-10.00 | 完成problem2.0 | 按时完成 |
| 5.29 | 12.00-12.30 | 阅读下一阶段要求 | 按时完成 |
| 5.29 | 12.30-16.30 | class | 按时完成 |
| 6.1 | 16.30-22.00 | 学习怎么写RI，AF | 按时完成 |
| 6.2 | 8.30-11.30 | 完成P1内容 | 按时完成 |
| 6.4 | 12.30-13.00 | 读P2任务 | 按时完成 |
| 6.5 | 13.00-14.00 | 完成P2 | 按时完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 刚开始不懂RI，AF，表示泄露，都并不会写，也不会写文档 | 查看往年的录屏资料，边学边写 |
| 抽象类的表示泄露预防以及不可变类型知识点不熟悉 | 网上查资料，请教学长 |
| 对接口，继承不熟悉 | 问学长，看mooc |

# 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，学会了设计 ADT 规约并评估规约的质量，学会了ADT泛型化，学会并熟练编写测试用例，更加理解可变类型和不可变类型，学会查看代码覆盖度。学会编写和设计AF，Ri以及如何防止表示泄露。

## 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编程代码比较多，但是可以多人一起编写，提高效率。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

泛型适应性更强，但是限制也更多，所以对于我来说，我更喜欢编写非泛型，因为不需要考虑到其他类型能不能使用。

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？

能够更有针对性得设计测试用例。可以适应。

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？

代码的可移植性搞，不需要重写代码如果换一个数据类型。

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？

没有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？

防止表示泄露是非常重要的一件事，如果不CheckRep万一发生表示泄露那么很难找到源头在哪，我们又得重新一句一句去找表示泄露处。以后编程肯定也要这样做

1. 关于本实验的工作量、难度、deadline。

P1 工作量较大，难度也较大，P2较小，大部分都是调用之前写过的代码，时间比较充足。

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

我希望老师可以多讲一些例子，例子太少很难理解到底什么意思，况且该课程国内找不到相对于的网课，所以如果没听明白那么就真的找不到补救方法。