

**2021年春季学期  
计算学部《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 劳鑫航 |
| 学号 | 1190201502 |
| 班号 | 1903009 |
| 电子邮件 | 1025052998@qq.com |
| 手机号码 | 18804638803 |

**目录**

[1 实验目标概述 1](#_Toc72247573)

[2 实验环境配置 1](#_Toc72247574)

[3 实验过程 1](#_Toc72247575)

[3.1 Poetic Walks 1](#_Toc72247576)

[3.1.1 Get the code and prepare Git repository 1](#_Toc72247577)

[3.1.2 Problem 1: Test Graph <String> 1](#_Toc72247578)

[3.1.3 Problem 2: Implement Graph <String> 1](#_Toc72247579)

[3.1.3.1 Implement ConcreteEdgesGraph 2](#_Toc72247580)

[3.1.3.2 Implement ConcreteVerticesGraph 2](#_Toc72247581)

[3.1.4 Problem 3: Implement generic Graph<L> 2](#_Toc72247582)

[3.1.4.1 Make the implementations generic 2](#_Toc72247583)

[3.1.4.2 Implement Graph.empty() 2](#_Toc72247584)

[3.1.5 Problem 4: Poetic walks 2](#_Toc72247585)

[3.1.5.1 Test GraphPoet 2](#_Toc72247586)

[3.1.5.2 Implement GraphPoet 2](#_Toc72247587)

[3.1.5.3 Graph poetry slam 2](#_Toc72247588)

[3.1.6 Before you’re done 2](#_Toc72247589)

[3.2 Re-implement the Social Network in Lab1 2](#_Toc72247590)

[3.2.1 FriendshipGraph类 2](#_Toc72247591)

[3.2.2 Person类 3](#_Toc72247592)

[3.2.3 客户端main() 3](#_Toc72247593)

[3.2.4 测试用例 3](#_Toc72247594)

[3.2.5 提交至Git仓库 3](#_Toc72247595)

[4 实验进度记录 3](#_Toc72247596)

[5 实验过程中遇到的困难与解决途径 3](#_Toc72247597)

[6 实验过程中收获的经验、教训、感想 4](#_Toc72247598)

[6.1 实验过程中收获的经验和教训 4](#_Toc72247599)

[6.2 针对以下方面的感受 4](#_Toc72247600)

# 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

# 实验环境配置

用的IDEA编译器，自带覆盖率测量，无需配置如EclEmma类的插件。

在这里给出你的GitHub Lab2仓库的URL地址（Lab2-学号）。

# 实验过程

请仔细对照实验手册，针对两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

本次实验给出了一个图接口，要求我们建立一个边图类、一个点图类分别继承自这个图接口，并且在里面实现一系列方法，并且实现抽象数据型，并用这个图的抽象数据型完成poet的工作。主要目的是练习ADT的规约设计和ADT的实现。

### 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。

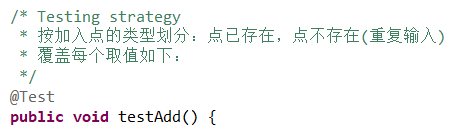
从网址：<https://github.com/rainywang/Spring2021_HITCS_SC_Lab2/tree/master/P1>

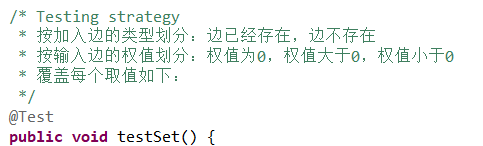
处得到代码。

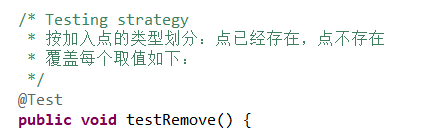
### Problem 1: Test Graph <String>

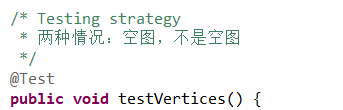
思路：这是针对Graph<String>设计相应的测试策略，主要针对里面的每个方法进行等价类划分的测试。

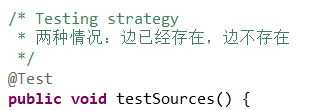
过程：如下图所示，为所有方法的测试策略：

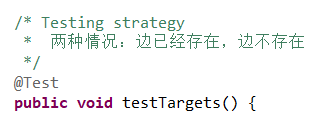




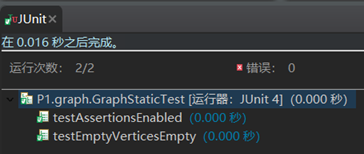








结果：在下面的junit总体中看。

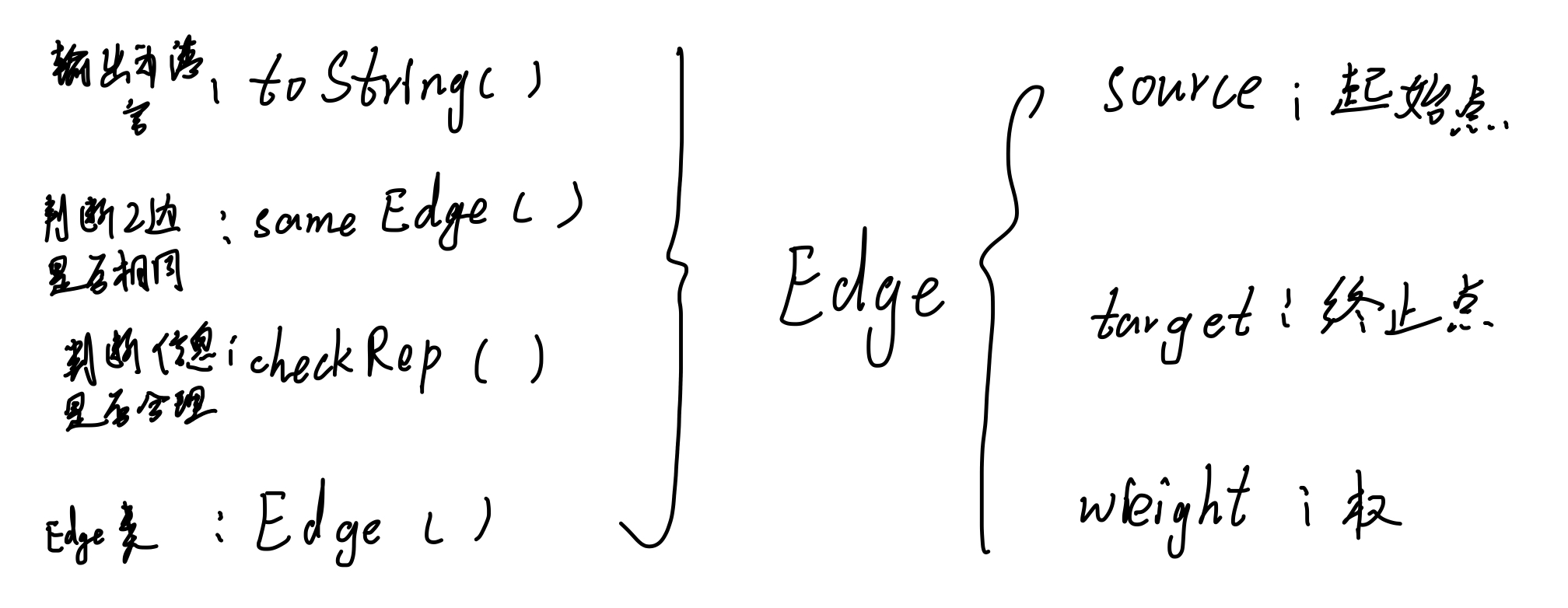


### Problem 2: Implement Graph <String>

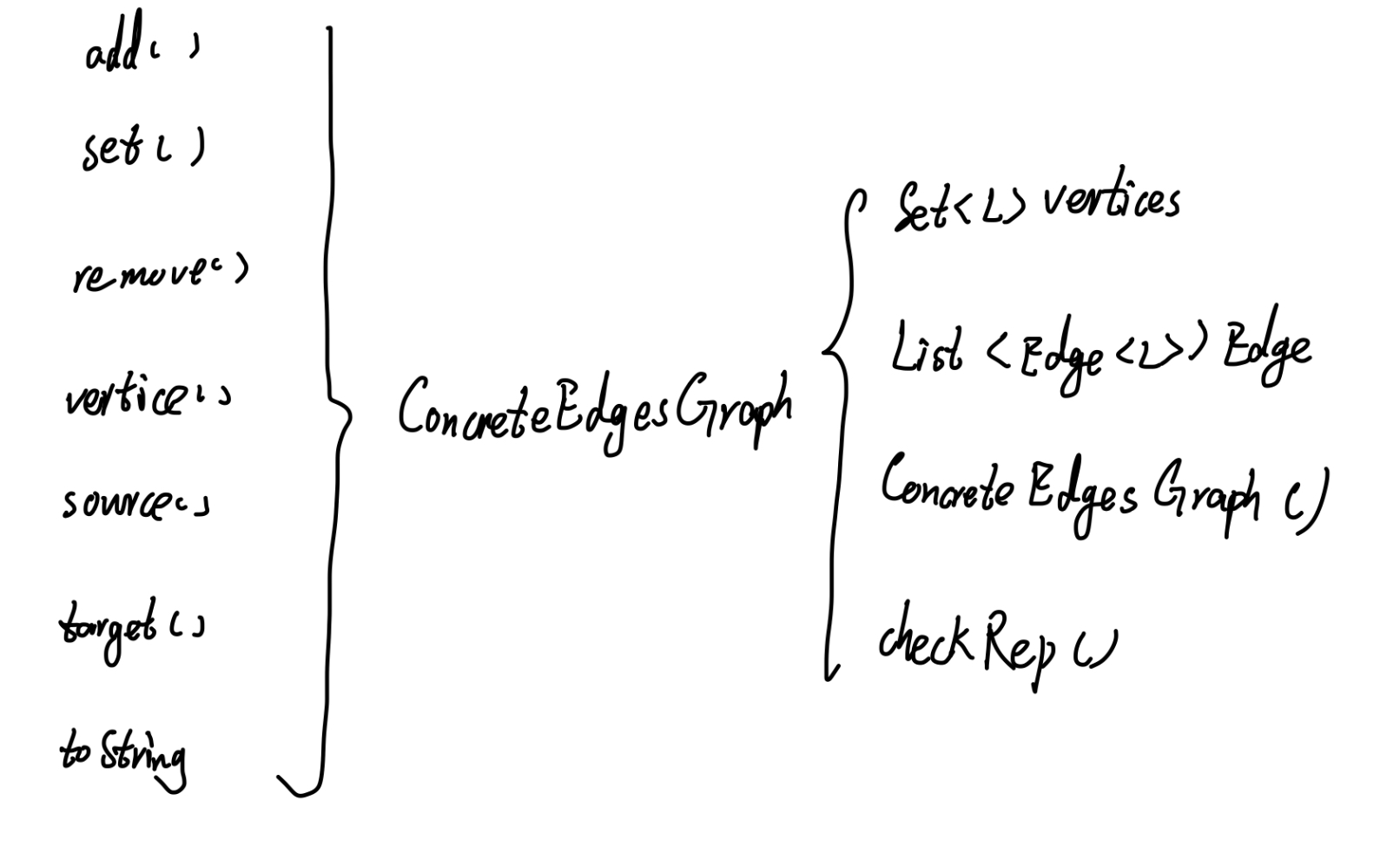
以下各部分，请按照MIT页面上相应部分的要求，逐项列出你的设计和实现思路/过程/结果。

#### Implement ConcreteEdgesGraph

**Edge:**Edge的功能主要为存储边的信息。此外，增加了判断两条边是否相等的方法。

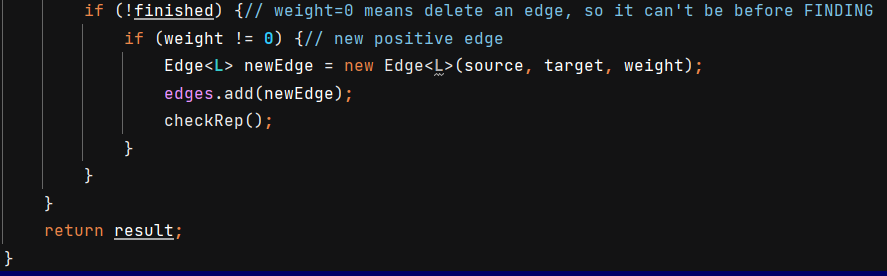


****ConcreteEdgesGraph：****该类以Edge为基础，用集合来存储点和边，每增加一条边就会改变集合，而点的删除同理。

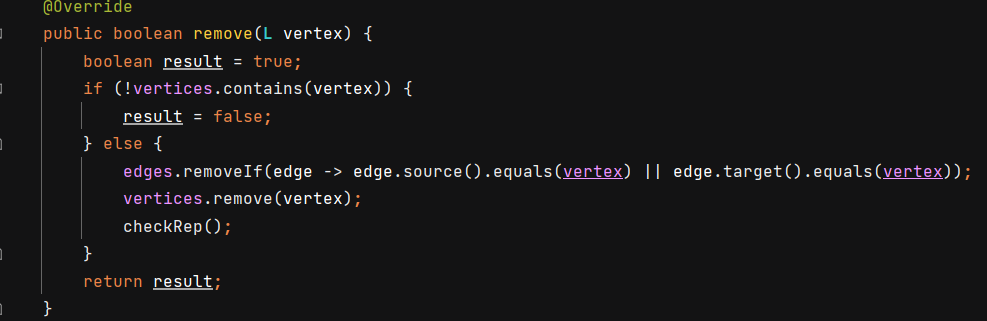


Set():



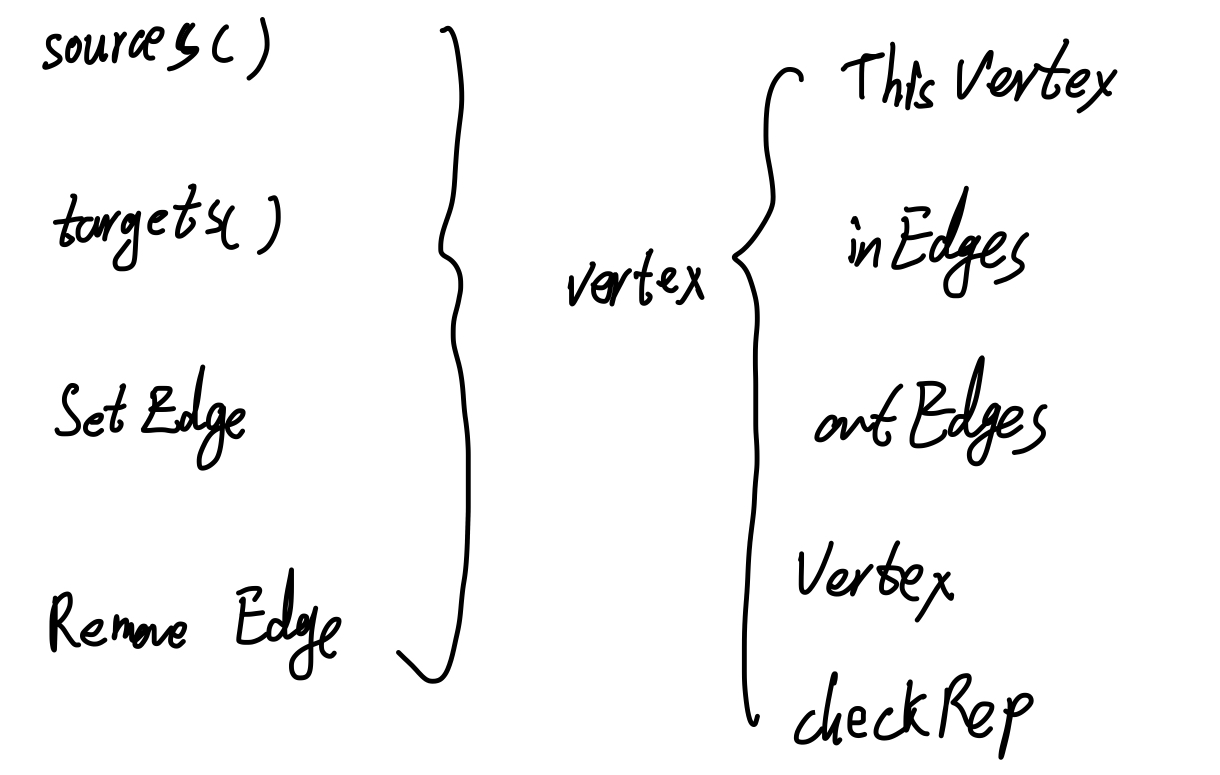


Remove():



#### Implement ConcreteVerticesGraph

****Vertex:****Vertex是点的抽象类，包含3个信息：点名称、入边、出边。Vertex需要能访问这3个信息，以及增加或删除进边、出边。



SetInEdges():

public int setInEdge(L source, int weight) {

if (weight <= 0)

return 0;

Iterator<L> it =inEdges.keySet().iterator();

while (it.hasNext()) {

L key = it.next();

if (key.equals(source)) {

int lastEdgeWeight = inEdges.get(key);

it.remove();

inEdges.put(source, weight);

return lastEdgeWeight;

}

}

inEdges.put(source, weight);

checkRep();

return 0;

}

**ConcreteVerticeGraph():**以点为基础的图，每个点通过唯一的标识进行区分，set和remove都依赖与Vertex类中的添加和删除操作，sources和targets也调用了Vertex类的方法。

Set():

@Override

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

if (weight < 0)

throw new RuntimeException("Negative weight");

if (source.equals(target))

return 0;

Vertex<L> from = null, to = null;

for (Vertex<L> vertex : vertices) {

if (vertex.ThisVertex().equals(source))

from = vertex;

if (vertex.ThisVertex().equals(target))

to = vertex;

}

if (from == null || to == null)

throw new NullPointerException("Inexistent vertex");

int lastEdgeWeight;

if (weight > 0) {

lastEdgeWeight = from.setOutEdge(target, weight);

lastEdgeWeight = to.setInEdge(source, weight);

} else {

lastEdgeWeight = from.removeOutEdge(target);

lastEdgeWeight = to.removeInEdge(source);

}

checkRep();

return lastEdgeWeight;

}

Remove():

@Override

public boolean remove(L vertex) {

for (Vertex<L> THIS : vertices) {

if (THIS.ThisVertex().equals(vertex)) {

for (Vertex<L> v : vertices) {

if (THIS.sources().containsKey(v)) {

// THIS.removeInEdge(v);

v.removeOutEdge(THIS.ThisVertex());

}

if (THIS.targets().containsKey(v)) {

// THIS.removeOutEdge(v);

v.removeInEdge(THIS.ThisVertex());

}

}

vertices.remove(THIS);

checkRep();

return true;

}

}

checkRep();

return false;

}

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

#### Make the implementations generic

在程序中选择“重构”或选择“String”并选择更改所有匹配项（要注意toString），即可实现泛化类型

#### Implement Graph.empty()

使Graph.empty()能返回一个新的空:

public static Graph<String> empty() {

return new ConcreteEdgesGraph();

}

### Problem 4: Poetic walks

#### Test GraphPoet

1. 在基于预设的测试用例基础上，增加等价类划分的多种情况。
2. 等价类划分：两个单词之间不存在连接词，两个单词之间只有一个连接词，两个单词之间有多个连接词。
3. 此外还要注意句末的句号，测试当一个句子最后一个词是“桥”的一端。

#### Implement GraphPoet

1. ****表示不变量和检查不变量:**应用中**所有的点都不为空。
2. ****构造函数：****用文件输入单词，String.split()分割为数组，通过String.toLowerCase()小写化。 接下来构建图，相邻的单词加边。首先要在加边前通过Graph.add()加点，加边时要判断是否存在：由于Graph.set()能返回之前加的边的值，以此来判断是否存在，存在则在之前的值加一（之前的边的值保存为lastEdgeWeight）。

int lastEdgeWeight = graph.set(words[i - 1].toLowerCase(), words[i].toLowerCase(), 1);if (lastEdgeWeight != 0) graph.set(words[i - 1].toLowerCase(), words[i].toLowerCase(), lastEdgeWeight + 1);

1. ****Poem(String input)****

当相邻两个单词任意一个不在之前创建的图里，则将后者单词加入即可（再加个空格）当存在时，由于Bridge长度只能为2，所以：分别求两个单词的sources和targets，将该Map转换为Set求交集；若交集为空，则无桥，若交集不空，则在交集中找最短的桥（可以在Map的value中查询weight）。

交集：

targets = graph.targets(words[i - 1].toLowerCase());

sources = graph.sources(words[i].toLowerCase());

intersection = sources.keySet();

intersection.retainAll(targets.keySet());

最大值：

int maxBridge = Integer.MIN\_VALUE;

String bridge = "";

for (String key : intersection) {

if (sources.get(key) + targets.get(key) > maxBridge) {

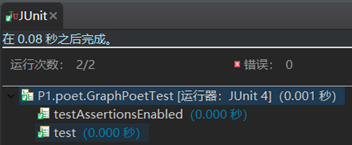
maxBridge = sources.get(key) + targets.get(key);

bridge = key;

}

}

#### Graph poetry slam

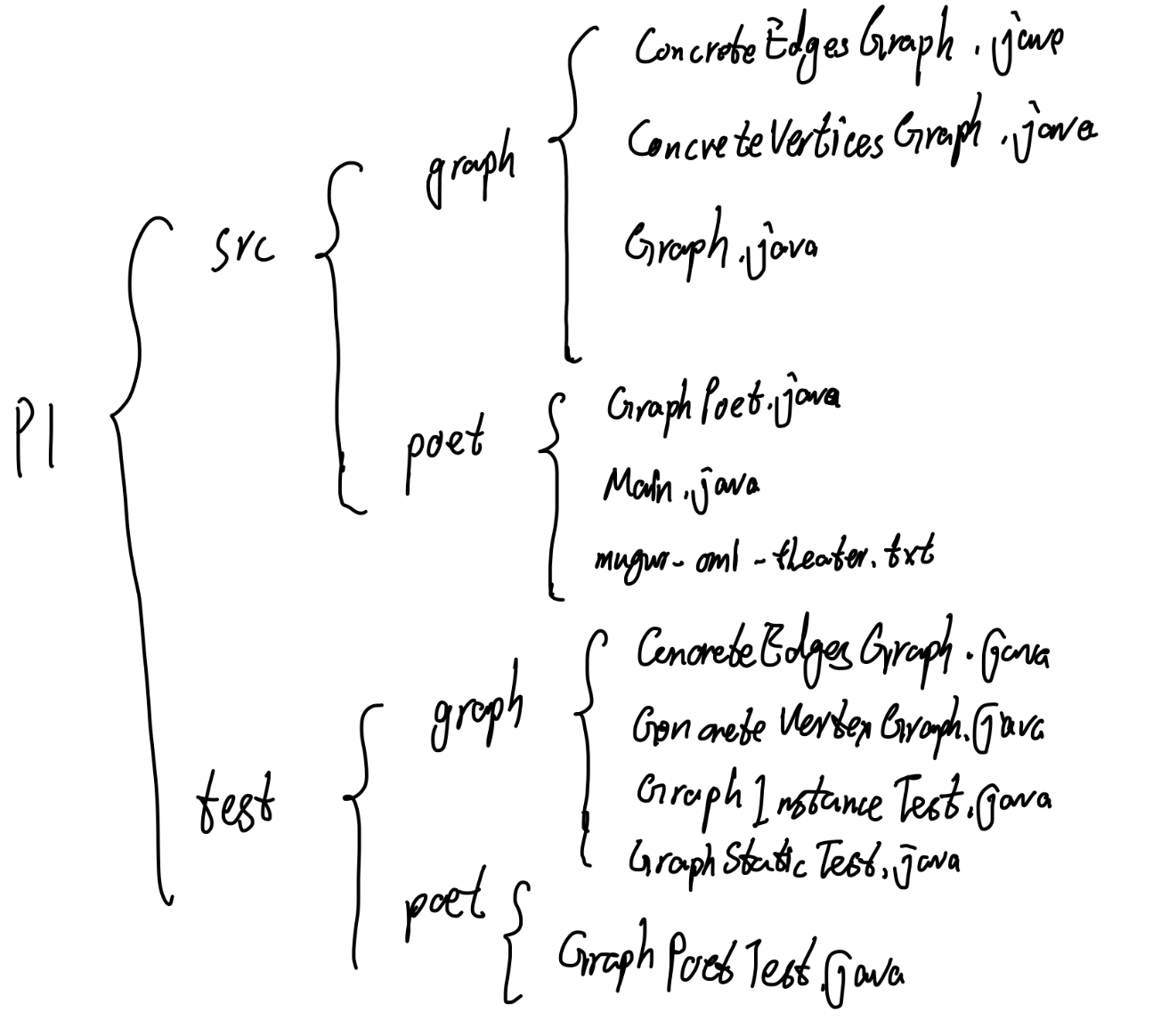


### Before you’re done

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。

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。



## Re-implement the Social Network in Lab1

我们在3.1中写的ADT，把第一次实验中的FriendshipGraph重新实现一遍，图中的节点仍然是Person类型，所以泛型L一律为Person. 而对于已经写好的FriendshipGraph中的方法，要用3.1中的Graph ADT中的方法来实现它们。

### FriendshipGraph类

****Graph < Person > graph：****直接调用Graph的静态方法.empty()生成一个空的图。

****boolean addVertex()：****直接调用graph.add()添加点。

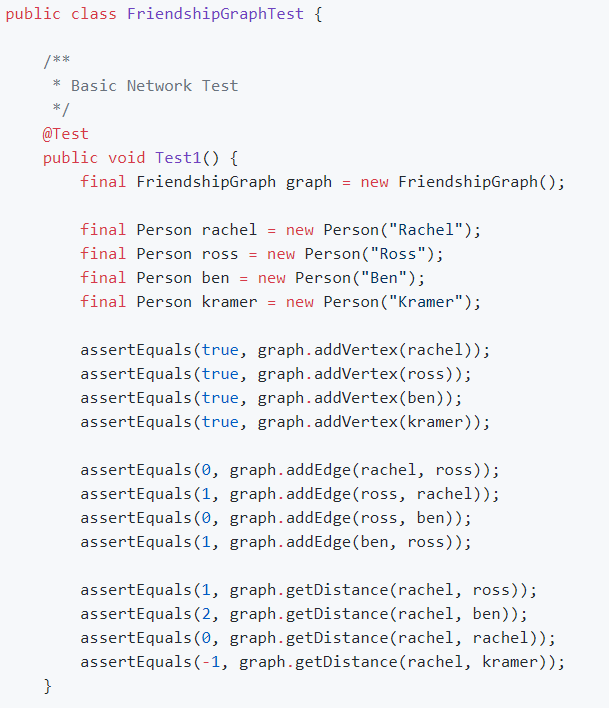
****int addEdge()：****调用graph.set()两次，添加双向边，默认权值为1，并记录可能存在的旧边的权值。

****int getDistance()：****首先判断起止点是否相等。再新建Map<Person, Integer> dis表示从起始点开始到该Person的距离，以及Map<Person, Boolean> vis表示该Person是否访问过。将两个Map初始化后，把起点标记为已经访问（所有涉及这两个Map的操作均需要remove后再put，后文不再阐述）。然后开始BFS搜索，找到终点为止。

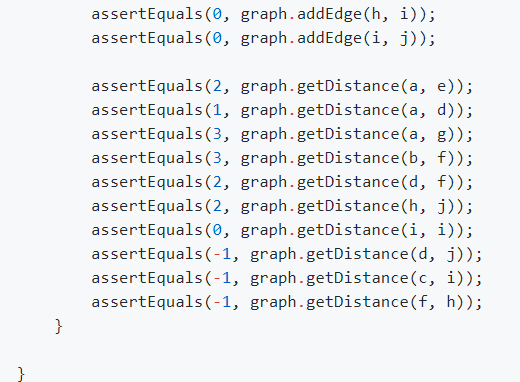
### Person类

* 该类的目标是将每一个人对应到一个Person对象，并存储名字的信息。为了防止泄露，我将String Name设置为私有且不可变的。在构造函数中将Name初始化。

### 客户端main()

。





### 测试用例

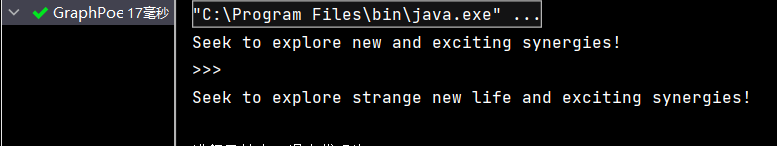
ConcreteEdgesGraphTest：



ConcreteVerticesGraphTest：



GraphPoetTest:



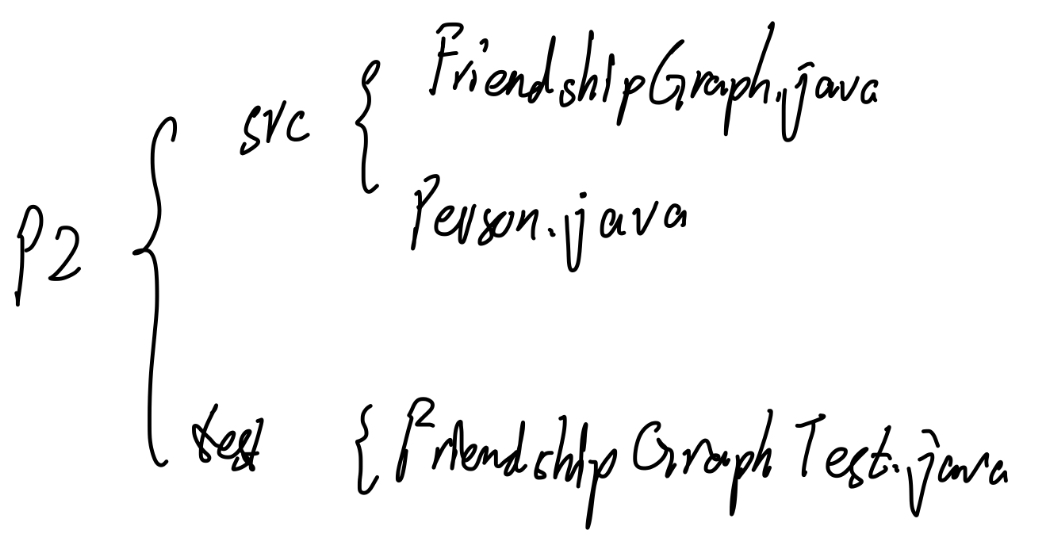
FriendshipText:



### 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。

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 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 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

## 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？
2. 使用泛型和不使用泛型的编程，对你来说有何差异？
3. 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？
4. 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？
5. 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？
6. 关于本实验的工作量、难度、deadline。
7. 《软件构造》课程进展到目前，你对该课程有何体会和建议？