

**2023年春季学期  
计算学部《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 房昕玉 |
| 学号 | 2021113290 |
| 班号 | 2137102 |
| 电子邮件 | 2354598747@qq.com |
| 手机号码 | 18629831988 |

**目录**

[1 实验目标概述 1](#_Toc19500)

[2 实验环境配置 1](#_Toc4039)

[3 实验过程 2](#_Toc25707)

[3.1 Poetic Walks 2](#_Toc11216)

[3.1.1 Get the code and prepare Git repository 2](#_Toc23199)

[3.1.2 Problem 1: Test Graph <String> 2](#_Toc103)

[3.1.3 Problem 2: Implement Graph <String> 4](#_Toc29923)

[3.1.3.1 Implement ConcreteEdgesGraph 4](#_Toc12454)

[3.1.3.2 5.Implement ConcreteVerticesGraph 8](#_Toc4519)

[3.1.4 Problem 3: Implement generic Graph<L> 12](#_Toc20981)

[3.1.4.1 Make the implementations generic 12](#_Toc18056)

[3.1.4.2 Implement Graph.empty() 13](#_Toc27146)

[3.1.5 Problem 4: Poetic walks 13](#_Toc22214)

[3.1.5.1 Test GraphPoet 14](#_Toc14843)

[3.1.5.2 Implement GraphPoet 14](#_Toc18062)

[3.1.5.3 Graph poetry slam 16](#_Toc9966)

[3.1.6 使用Eclemma检查测试的代码覆盖度 16](#_Toc14707)

[3.1.7 Before you’re done 16](#_Toc21795)

[3.2 Re-implement the Social Network in Lab1 18](#_Toc32201)

[3.2.1 FriendshipGraph类 18](#_Toc28777)

[3.2.2 Person类 19](#_Toc29632)

[3.2.3 客户端main() 19](#_Toc13641)

[3.2.4 测试用例 19](#_Toc13936)

[3.2.5 提交至Git仓库 20](#_Toc10340)

[4 实验进度记录 20](#_Toc24122)

[5 实验过程中遇到的困难与解决途径 21](#_Toc5588)

[6 实验过程中收获的经验、教训、感想 21](#_Toc12066)

[6.1 实验过程中收获的经验和教训（必答） 21](#_Toc151)

[6.2 针对以下方面的感受（必答） 21](#_Toc23313)

# 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：

l 针对给定的应用问题，从问题描述中识别所需的 ADT；

l 设计 ADT 规约（pre-condition、post-condition）并评估规约的质量；

l 根据 ADT 的规约设计测试用例；

l ADT 的泛型化；

l 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）

l 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）；

l 测试 ADT 的实现并评估测试的覆盖度；

l 使用 ADT 及其实现，为应用问题开发程序；

l 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

本次实验需要测试代码覆盖率：因为我使用的是idea，所以使用idea自带的测试代码覆盖率：

右键测试类，点击使用覆盖率运行



在这里给出你的GitHub Lab2仓库的URL地址（Lab2-学号）

<https://github.com/ComputerScienceHIT/HIT-Lab2-2021113290>

git@github.com:ComputerScienceHIT/HIT-Lab2-2021113290.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

为每个设计和实现的 ADT 撰写 mutability/ immutability 说明、AF、RI、safety from rep exposure。给出各 ADT 中每个方法的 spec。为每个 ADT 编写测试用例，并写明 testing strategy。

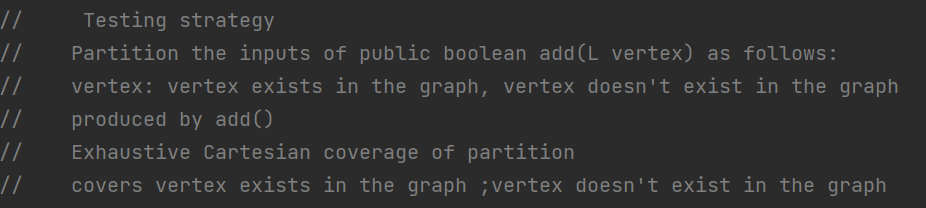
### Get the code and prepare Git repository

使用 idea 上面的 VCS把代码导入idea，在项目文件夹中使用git指令上传项目

### Problem 1: Test Graph <String>

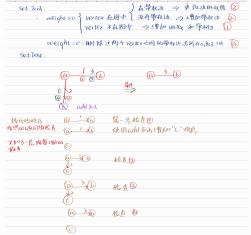
本问题要求为Graph接口中的方法编写testing strategy和test case，完成GraphInstanceTest的编写：我在做实验的时候首先研究了一下Graph接口，里面有对各个方法的spec，仔细阅读spec，编写各个方法的strategy。这个才是第一步，如果直接去编写测试类肯定会卡住，因为都不知道根据什么编写。

1. addTest:划分顶点为在图中的和不在图中的，一个顶点只能被添加一次，用vertices()方法来观察



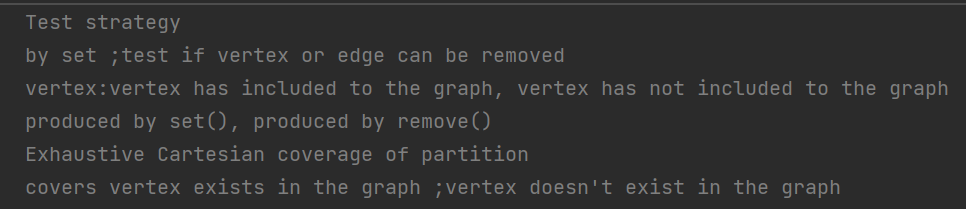
1. setTest:划分顶点为在图中的和不在图中的；顶点的权重大于0和=0；

测试覆盖了更改图，添加边，添加边和顶点，删除；由于set方法它所包含的功能比较多，于是我进行画图来辅助理解。

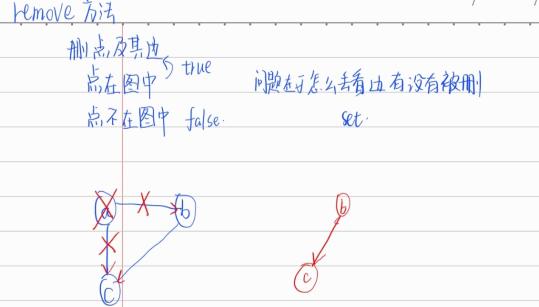


如图中所示，我用set的返回值和vertices()来观察。

1. removeTest:划分顶点在图中还是不在图中

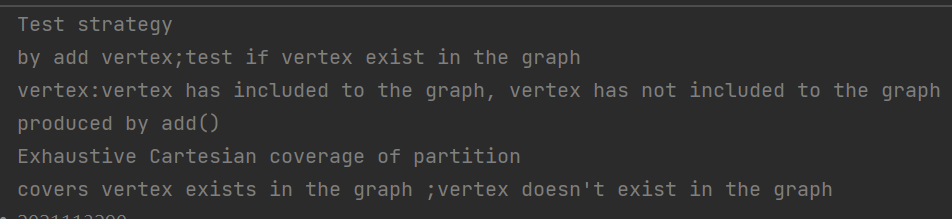


这个测试没有set复杂，但同样画图说明：



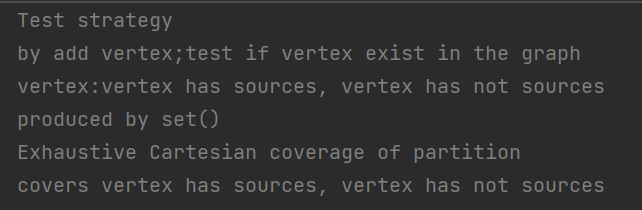
这里通过remove返回值和graph.sources().containsKey()来观察

1. verticesTest:



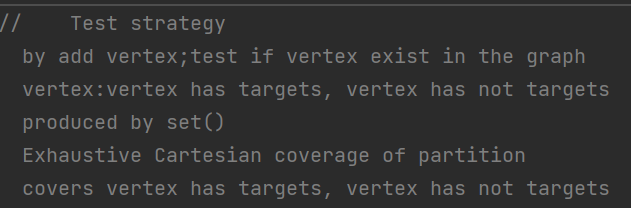
简单，通过vertices()观察

1. sourcesTest:划分为有源点的顶点和没有源点的顶点



这里通过graph.sources().containsKey()观察。

1. TargetsTest:划分为有终点的顶点和没有终点的顶点

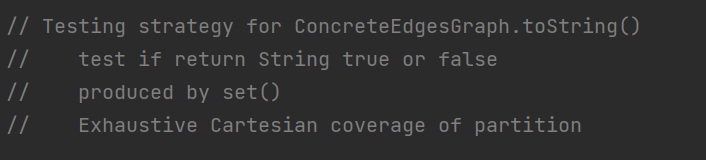


这里通过graph.targets().containsKey()观察。

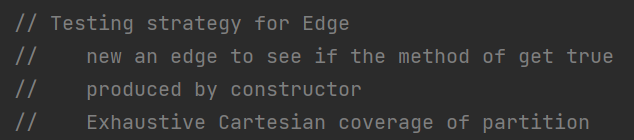
### Problem 2: Implement Graph <String>

#### Implement ConcreteEdgesGraph

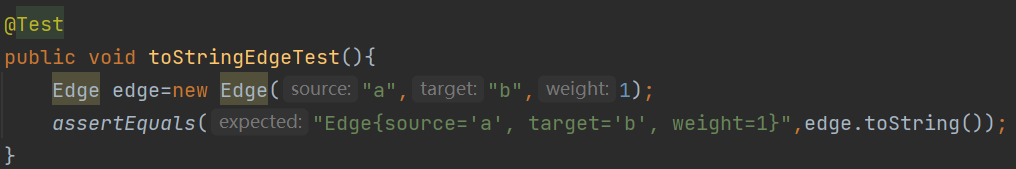
1. tests for ConcreteEdgesGraph.toString()：

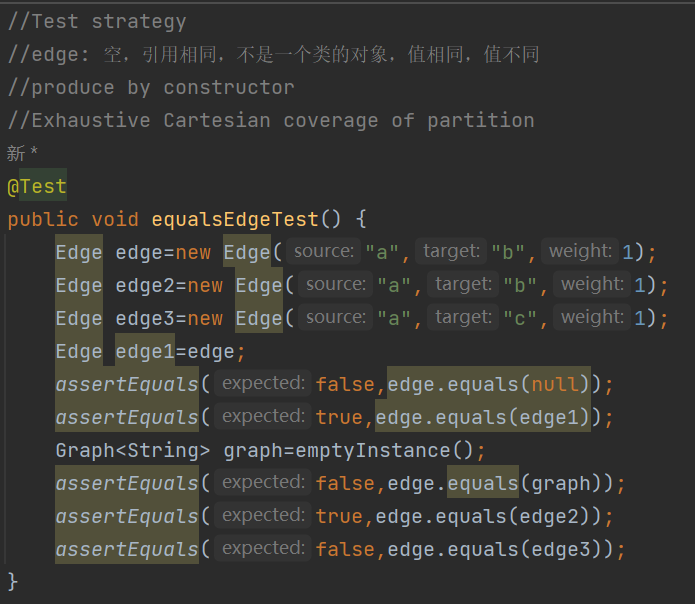


1. tests for operations of Edge：



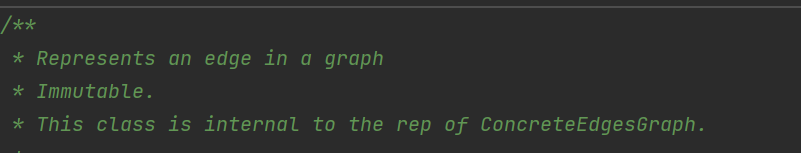




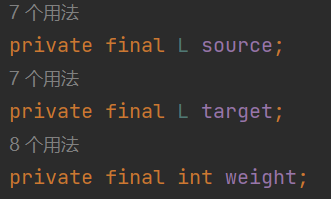


1. 编写类Edge：

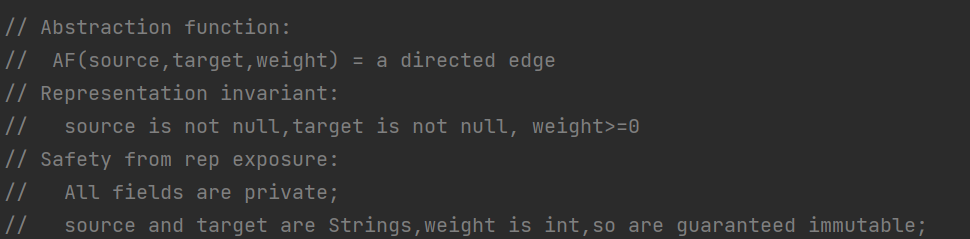
类Edge的规约：



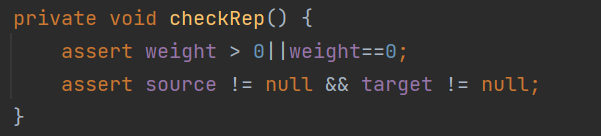
使用了三个用private修饰的成员变量（这里由于做到后面泛型，改了代码，L可以替换成String）：



编写类Edge的AF,RI,Safety from rep exposure:



根据上面编写checkRep：



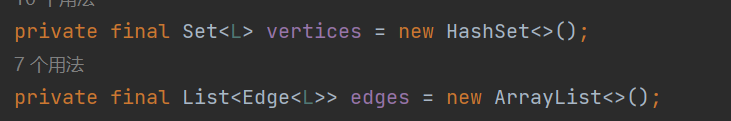
constructor:简单,不再赘述

methods包括：

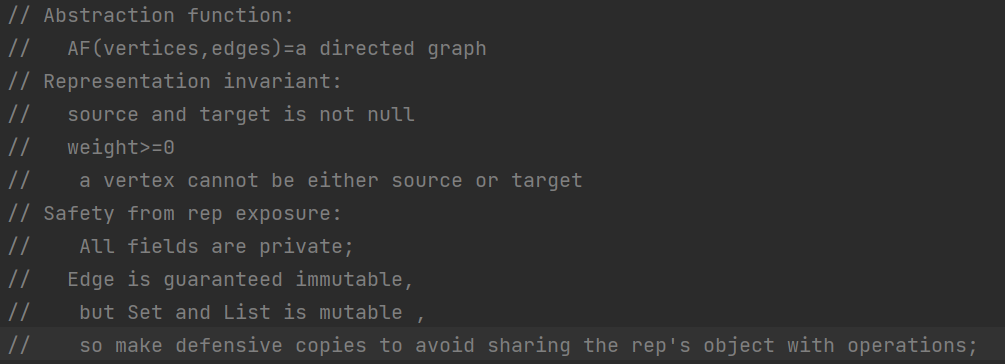
三个get方法：getSource(),getTarget(),getWeight();由于这个Edge类是不可变的，所以我要为其编写equals()和hashcode()来用于Edge类型的对象之间的比较；toString():由于实验中没有要求对其加强，直接使用最简单的return 相应成员变量

1. 实现ConcreteEdgesGraph类

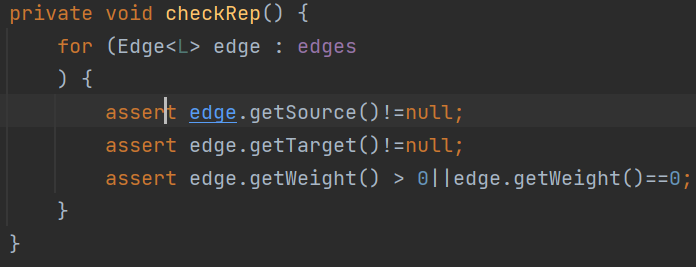
使用了两个用private修饰的成员变量（这里由于做到后面泛型，改了代码，L可以替换成String）：



编写类ConcreteEdgesGraph的AF,RI,Safety from rep exposure:



根据上面编写checkRep：



Constructor：里面没有具体实现，不赘述

methods包括：

1. add():

判断顶点集合中是否包含传过来的参数顶点，若包含return false;否则，把顶点加入到顶点集合中，return true。

1. set():
2. 先向顶点集中添加参数源点和终点，因为源点集是set，所以这里不用判断，直接添加，假设顶点集中已经有了添加的顶点，会直接覆盖。
3. 使用迭代器Iterator遍历边表，对于边表中的每一条边，如果图中有一条边的源点和终点**和**参数源点和终点相同，那么删除图中的这一条边
4. 检查边的权重，大于0就向边表中添加一条用传来的参数构造出来的新边，否则(weight=0)直接return weight, 因为边已经在前面删完了
5. remove()
6. 框架是如果顶点集合中包含传过来的参数顶点，若包含return true;否则，return true。
7. 在return true的if语句中，首先删除顶点:直接调用set的方法remove；然后使用迭代器Iterator遍历边表，对于边表中的每一条边，如果图中有一条边的源点和参数源点相同**或者**终点与参数源点相同，那么删除图中的这一条边
8. vertices():

防御式拷贝，返回vertices的拷贝

1. Sources():

新建一个map，使用迭代器Iterator遍历边表，对于边表中的每一条边，如果图中有一条边的终点**和**参数终点相同，那么将这条边的所有源点和对应边的权值放入新建集合中，返回新建集合

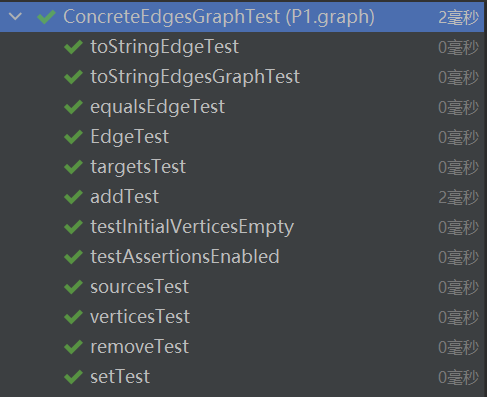
1. targets():

新建一个map，使用迭代器Iterator遍历边表，对于边表中的每一条边，如果图中有一条边的源点**和**参数源点相同，那么将这条边的所有终点和对应边的权值放入新建集合中，返回新建集合

1. toString()

实验要求加强toString,我的方法是使用StringBuilder，可变类型，可以减少空间的占用

至此，编写完ConcreteEdgesGraph.java，运行ConcreteEdgesGraphTest,通过测试：

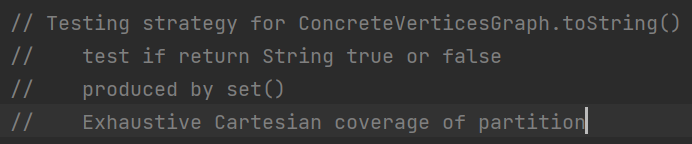


并且覆盖率如下：

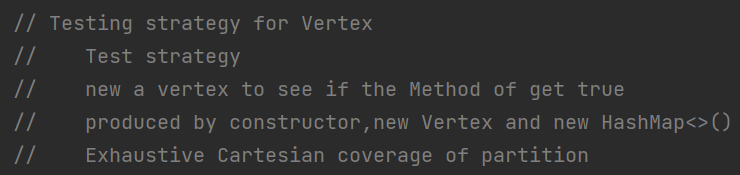


#### 5.Implement ConcreteVerticesGraph

1. ConcreteVerticesGraph.toString()：



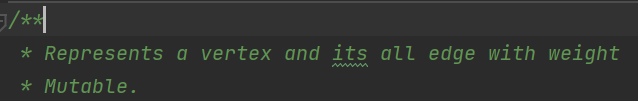
1. tests for operations of Vertex：





1. 编写类Vertex

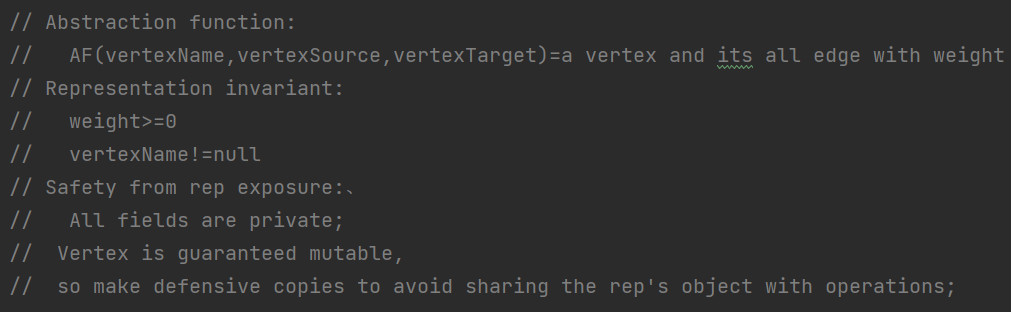
类Vertex的规约：



使用了三个用private修饰的成员变量（这里由于做到后面泛型，改了代码，L可以替换成String）：



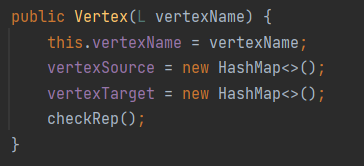
编写类Vertex的RI,AF,Safety from rep exposure:



根据上面编写checkRep(),略复杂，但代码中注释很清楚：



constructor:



methods包括：

五个get方法：

getVertexName():返回名字

getVertexSource():返回源点集合

getVertexTarget()：返回终点集合

getSourceWeight()：返回到源点的边的权重

getTargetWeight()：返回到终点的边的权重

两个add方法：为引用该方法的顶点增加一个终点/源点及其边

addTarget(L targetName , int weight);addSource(L sourceName , int weight)

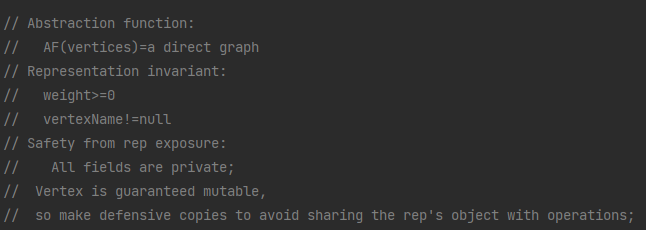
两个remove方法：将引用该方法的终点/源点集合中的一个终点/源点删去

1. 实现ConcreteVerticesGraph类

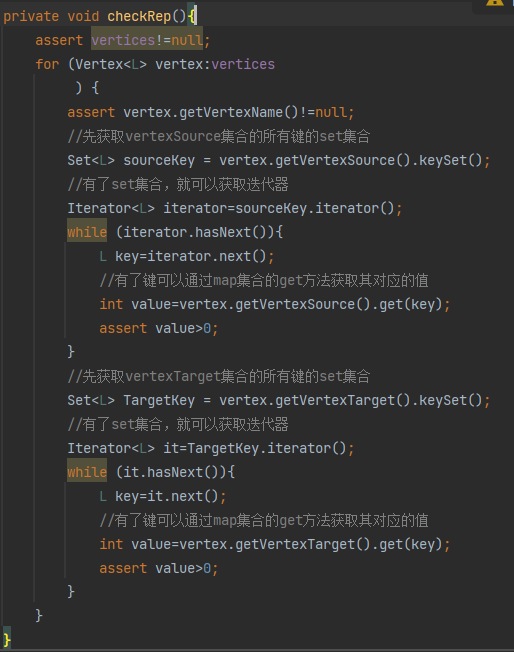
使用了一个用private修饰的成员变量（这里由于做到后面泛型，改了代码，L可以替换成String）：



编写类ConcreteVerticesGraph的AF,RI,Safety from rep exposure:



根据上面编写checkRep：



constructor:没有具体实现,不再赘述

methods包括：

1. add():

检查顶点集合里面是否有顶点的名字和传入的顶点的名字相同，若包含return false;否则，把顶点加入到顶点集合中，return true。

1. set():
2. 遍历顶点集合，如果有和参数源点或终点相同的点，就记录下来
3. 没有名字相同的就可以把顶点名字加入到顶点集合中
4. 遍历顶点集合，如果找到了和参数源点相同的顶点，判断该顶点的终点集是否包括参数终点：包括，则在该顶点的终点集中删除该参数终点；给该顶点添加终点为参数终点和参数权重。再次遍历顶点集合，如果找到了和参数终点相同的顶点，同理。
5. remove()
6. 使用flag标记顶点是否被删除：1为删除，0为没删除
7. 使用迭代器Iterator遍历顶点，如果该顶点和参数顶点名字相同，则删除该顶点，flag置为1；如果该顶点的源点集合或终点集合包含参数顶点，删除该顶点，并将flag置为1；
8. return flag!=0
9. vertices():

防御式拷贝，返回vertices的拷贝(成员变量只用到顶点名字)

1. Sources():

新建一个map，遍历顶点集合，如果该顶点的名字等于参数顶点，将该顶点的源点集合赋值给新建map

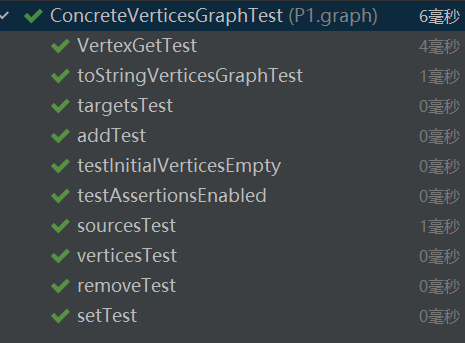
1. targets():

新建一个map，遍历顶点集合，如果该顶点的名字等于参数顶点，将该顶点的终点集合赋值给新建map

1. toString()

实验要求加强toString,我的方法是使用StringBuilder，可变类型，可以减少空间的占用

至此，编写完ConcreteVerticesGraph.java，运行ConcreteVerticesGraphTest,通过测试：



覆盖率如下：

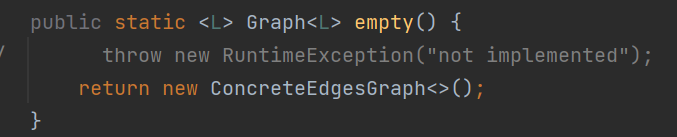


### Problem 3: Implement generic Graph<L>

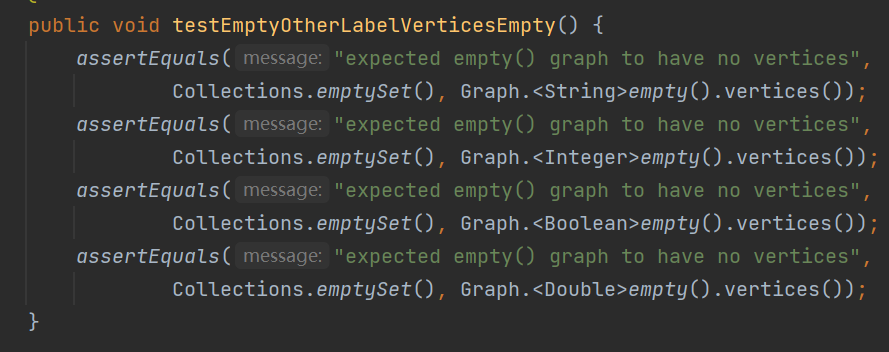
#### Make the implementations generic

使Concrete­EdgesGraph、ConcreteVerticesGraph、Edge和Vertex成为泛型类型，直接修改即可

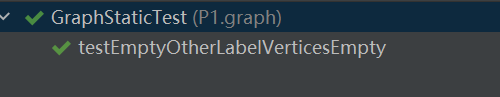
#### Implement Graph.empty()



在提供的GraphStaticTest.java中添加一些额外的测试，这些测试可以创建和使用具有几个不同类型（不可变）标签的Graph实例。



测试通过：



### Problem 4: Poetic walks

GraphPoet是用一个文本语料库初始化的，它用这个语料库来派生一个诗歌图。

图中的顶点是单词。单词被定义为非空格非换行字符的不区分大小写的非空字符串。它们在语料库中由空格、换行符或文件末尾分隔。

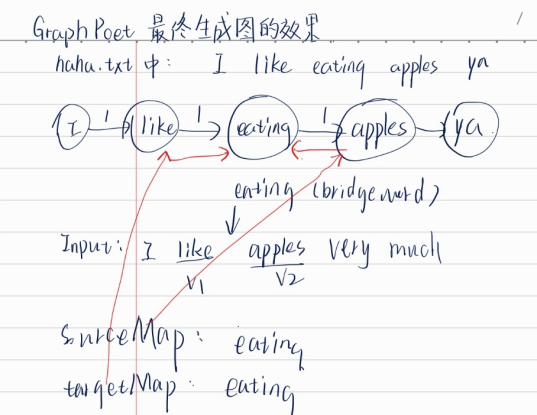
图中的边计数邻接：语料库中“w1”后面跟着“w2”的次数是从w1到w2的边的权重。

我对该任务的理解：

GraphPoet:生成的诗歌图：每一个顶点是语料文件中的单词，有边代表两个单词紧挨着，边的权重是前一个单词紧挨着后一个单词的次数

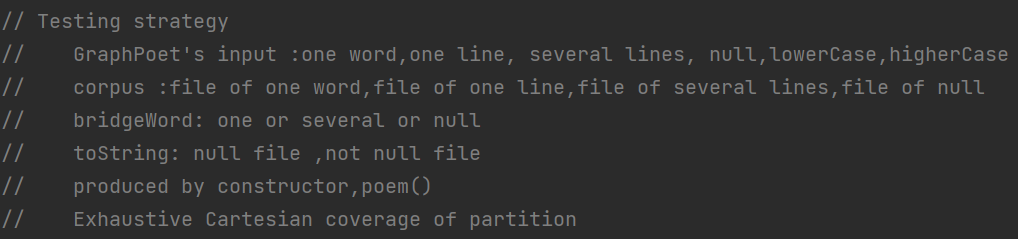
poem:检查输入的字符串语句，输入的字符串语句的相邻的两个顶点中加单词，加的一个单词是在生成的诗歌图中位于检查的两个单词中间的，且加完后的两条边的权值最大

画图分析理解：



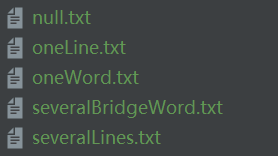
#### Test GraphPoet

测试策略划分如下：



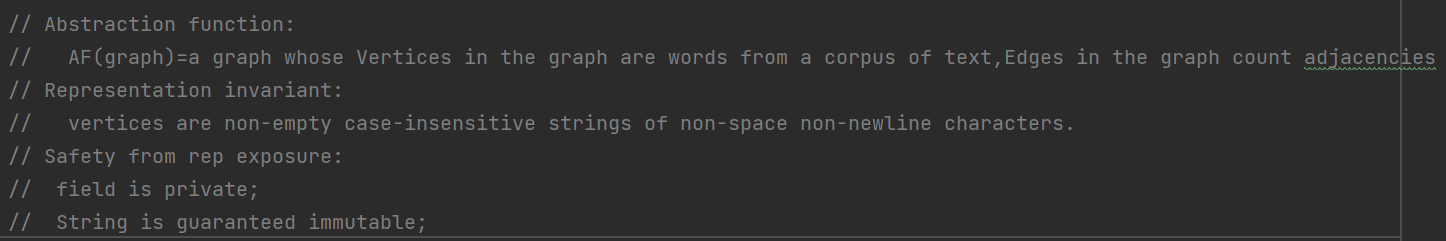
分别对input，corpus，bridgeWord，toString进行测试

在项目下新建如下txt文档：

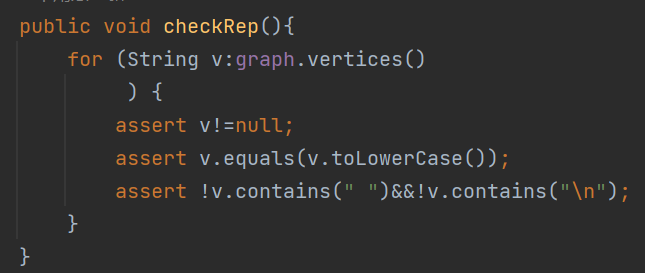


#### Implement GraphPoet

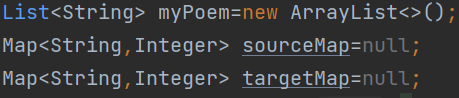
1. 成员变量：
2. AF,RI,Safety from rep exposure:



1. checkRep():



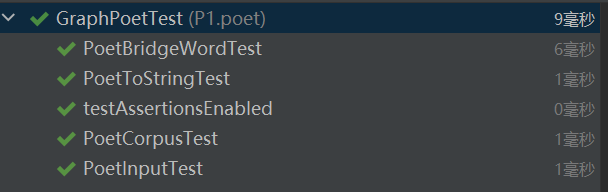
1. GraphPoet实现思路：
2. 首先读字符串文件,把文本用空格分割之后放到数组里面,数组的每一个元素都是顶点，是单词，新建map备用
3. 遍历数组，取v1，v2两个点（必须相邻），weight初始化为0；如果map中存在v1,v2这一顺序（紧挨着）的组合，将这两点间的边权重取出；v1,v2是相邻顶点，放入map中，并将这两个相邻节点的权重+1，放入map的原因是为了下一次迭代更新weight；v1,v2是相邻顶点，放入诗歌图中，并将这两个相邻节点的权重+1
4. 循环走完，诗歌图就写好了，写出来的诗歌图十元句子中每个相邻单词都放在了图中，且每一相邻的单词权重为1
5. poem实现思路：
6. 如果输入空，返回空
7. 使用的数据结构：



1. 把输入的文本中的所有换行符删去，把输入的句子切分为一个一个的单词放入myPoem列表中
2. 使用StringBuilder暂存要输出的诗歌。取列表中相邻两个单词v1,v2(有顺序)，求后一个单词在诗歌图中的所有源点集合s1，前一个单词在诗歌图中的所有终点集合s2；找桥接词：s1中有s2的元素j并且j在s1的对应权重+j在s2的对应权重大于当前记录的最大权重，则这个和为新的权重，桥接词为j
3. stringBuilder.append(myPoem.get(i)).append(" ").append(bridgeWord+" ") .append(myPoem.get(myPoem.size()-1));

(6)toString()实现

运行测试：

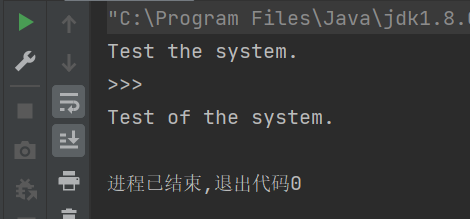


覆盖率：

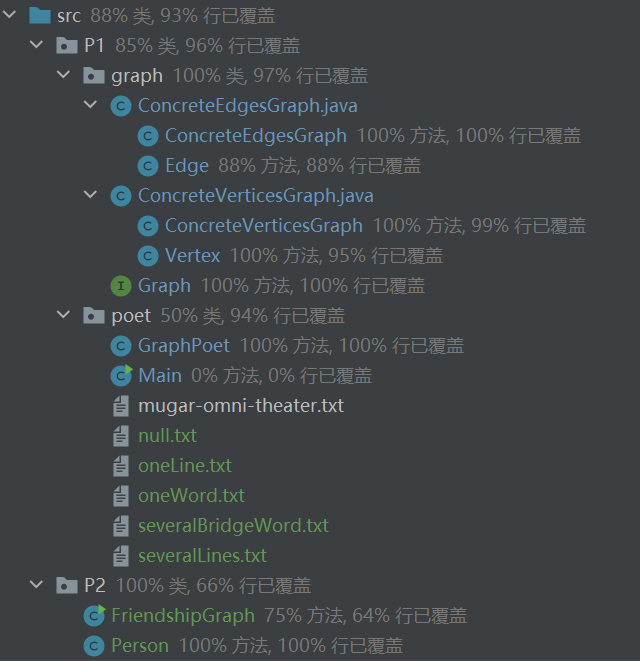


#### Graph poetry slam

运行Main.java



### 使用Eclemma检查测试的代码覆盖度

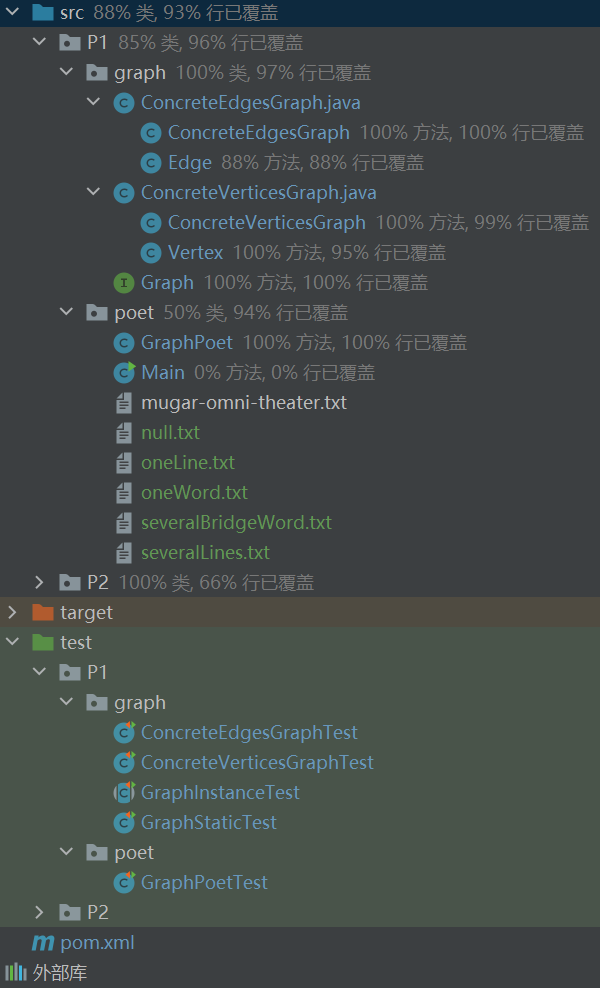


### Before you’re done

如何通过Git提交当前版本到GitHub上你的Lab2仓库。

1. 到项目文件夹右键 Git Bash Here
2. git init
3. git add .
4. git commit -m "提交信息"
5. git remote add origin [你的项目地址（包含token）]
6. git push -u origin master

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

基于在Poetic Walks 中定义的 Graph<L>及其两种实现，重新实现 Lab1 中 3.3 节的 FriendshipGraph 类。

新 FriendshipGraph 类要利用 3.1 节已经实现的 ConcreteEdgesGraph<L> 或 ConcreteVerticesGraph<L>，L 替换为 Person。

尽可能复用ConcreteEdgesGraph<L>或 ConcreteVerticesGraph<L>中已经实现的方法

### FriendshipGraph类

给出你的设计和实现思路/过程/结果。

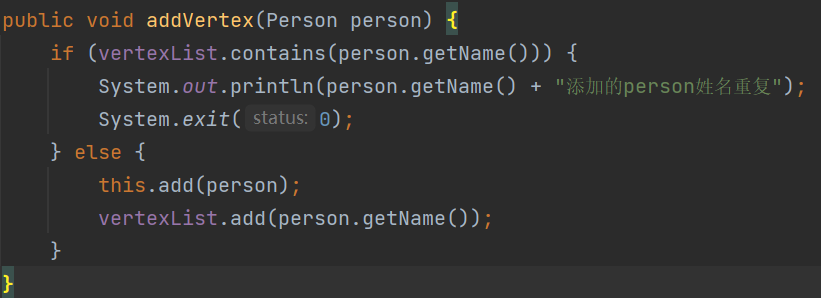
1. 

继承边图

1. 成员变量：存在的意义就是为了添加顶点的时候判断是否重名，否则直接引用顶点集是否包含看的是这个人的所有属性



1. AddVertex:

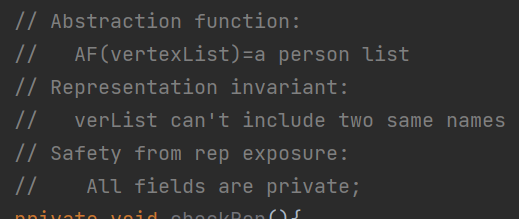


1. addEdge:

判断人是否存在时用this.vertices().contains(person1)

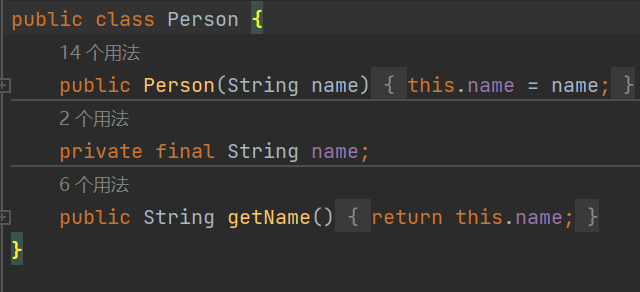
1. getDistance:

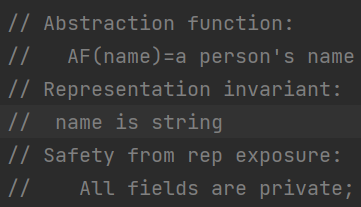
主要思想还是广搜，不过边界条件判断和遍历终点集合时用图中现有方法



### Person类

给出你的设计和实现思路/过程/结果。





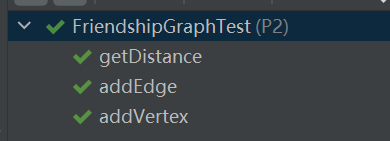
### 客户端main()

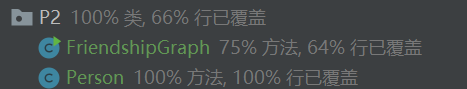
无需改动

### 测试用例

给出你的设计和实现思路/过程/结果。

延续使用lab1测试用例，稍加修改为观察器用vertices()，还需使用targets（），sources（）



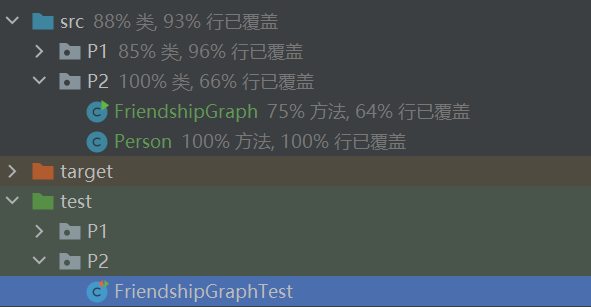


### 提交至Git仓库

如何通过Git提交当前版本到GitHub上你的Lab3仓库。

1. 到项目文件夹右键 Git Bash Here
2. git init
3. git add .
4. git commit -m "提交信息"
5. git remote add origin [你的项目地址（包含token）]
6. git push -u origin master

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 |
| 2023/3/28 | 15：40-17：30 | 浏览阅读mit上面的任务，理好框架 | 有看也看不明白的任务要求，但是浏览了一边大致有了方向，于是开始着手去做 |
| 2023/3/29 | 15：40-18：00 | 编写测试策略和测试方法 | 按计划完成，但由于没有实现后续代码，目前无法测试 |
| 2023/3/30 | 18：30-20：30 | 复习RI,AF,checkRep(),Safety from rep exposure | 按计划完成 |
| 2023/3/31 | 9：00-13：30 | 编写ConcreteEdgesGraph.java | 按计划完成 |
| 2023/3/31 | 15:30-18:30 | 编写ConcreteVerticesGraph.java | 遇到困难，未完成 |
| 2023/4/1 | 9：00-12：00 | 编写ConcreteVerticesGraph.java | 按计划完成 |
| 2023/4/2 | 9：00-14：20  15：30-18：30 | 编写poetic walks | 按计划完成 |
| 2023/4/3 | 18：00-21：00 | 重新实现lab1的社交网络图 | 提前完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 实现类的构造器怎么写？我其实有点怀疑到底用不用写实现类的构造器呢？ | 询问同学得知只是一个壳子，里面没有具体内容 |
| 现在测试类对  ConcreteVerticesGraph无法通过，看了一下报错，推测是add方法出了问题，add里的方法分析后，感觉没错，但是构造函数不知道写没写错 | 仔细分析代码，问题已解决，原因是else写在了循环里面报错 |
| 测试类对  ConcreteVerticesGraph无法通过，removetest失败 | 推测是set出问题了，增加了两次“a”所以第二次remove"a"的时候true了  推测是remove本身的方法出错了，删除顶点的同时，与其相连的顶点的边应删除--》这里的体现是，比如删除a，a-->c,那么除了删除a，还要删除c的source  最后求助了同学，发现是return true 那里有问题，故不用return true，使用flag来标记 |

# 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）

## 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？

面向过程是以“算法”为中心，通过使用函数实现各个步骤，并依次调用实现的函数解决问题；  
面向对象是以“对象”为中心，在设计好相应的对象后（对象的属性），将对象在整个问题中的相关行为封装到一起（对象的方法），然后各个对象共同将整个事情完成；

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

使用泛型可以在很大程度上扩展程序适用的数据类型，减少编写程序的工作量和函数数量。不使用泛型则不用担心调用方法时数据类型转换问题，但函数复杂繁多，编写工作量大

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？

只通过规约了解参数和返回值编写测试文件可以不用过多的顾虑方法中设置的条件，能更容易地检查出方法实现中出现的错误。这种做法是一种很好的设置防火墙的方法，刚开始可能不太适应，但必须强迫自己去适应，这对我们以后编程很有帮助

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？

缩短时间，提升进度：程序员开发新软件时，可以通过代码复用实现相同或者相似的功能，大大减少了整体开发的时间，可以节省时间去开发那些特有的功能。节省时间就是在省钱!使用已有代码，减少资源投入，可以控制开发成本，而且那些安全可靠的高质量代码还能在一定程度上提高产品的质量。

高质量的代码复用属于有效资源利用，并且有助于避免代码臃肿、冗长的问题。

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？

这些工作是为了让代码更容易被理解，让编程实现方法有一个思路和限制，以及保护rep信息防止其在客户端中泄露，造成不必要的错误。愿意

1. 关于本实验的工作量、难度、deadline。

大，难，但deadline合理，有充足时间逐一攻破难点

1. 《软件构造》课程进展到目前，你对该课程有何收获和建议？

收获相当大，将面向ADT的编程熟悉掌握，为以后开发铺好后路