

**2021年春季学期  
计算学部《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 童彦澎 |
| 学号 | 1190201701 |
| 班号 | 1903011 |
| 电子邮件 | [1308091501@qq.com](mailto:1308091501@qq.com) |
| 手机号码 | 18959076997 |

**目录**

[1 实验目标概述 1](#_Toc74492839)

[2 实验环境配置 1](#_Toc74492840)

[3 实验过程 3](#_Toc74492841)

[3.1 Poetic Walks 3](#_Toc74492842)

[3.1.1 Get the code and prepare Git repository 3](#_Toc74492843)

[3.1.2 Problem 1: Test Graph <String> 3](#_Toc74492844)

[3.1.3 Problem 2: Implement Graph <String> 5](#_Toc74492845)

[3.1.3.1 Implement ConcreteEdgesGraph 6](#_Toc74492846)

[3.1.3.2 Implement ConcreteVerticesGraph 10](#_Toc74492847)

[3.1.4 Problem 3: Implement generic Graph<L> 15](#_Toc74492848)

[3.1.4.1 Make the implementations generic 15](#_Toc74492849)

[3.1.4.2 Implement Graph.empty() 16](#_Toc74492850)

[3.1.5 Problem 4: Poetic walks 17](#_Toc74492851)

[3.1.5.1 Test GraphPoet 17](#_Toc74492852)

[3.1.5.2 Implement GraphPoet 18](#_Toc74492853)

[3.1.5.3 Graph poetry slam 19](#_Toc74492854)

[3.1.6 Before you’re done 20](#_Toc74492855)

[3.2 Re-implement the Social Network in Lab1 20](#_Toc74492856)

[3.2.1 FriendshipGraph类 21](#_Toc74492857)

[3.2.2 Person类 22](#_Toc74492858)

[3.2.3 客户端main() 23](#_Toc74492859)

[3.2.4 测试用例 24](#_Toc74492860)

[3.2.5 提交至Git仓库 25](#_Toc74492861)

[4 实验进度记录 26](#_Toc74492862)

[5 实验过程中遇到的困难与解决途径 26](#_Toc74492863)

[6 实验过程中收获的经验、教训、感想 26](#_Toc74492864)

[6.1 实验过程中收获的经验和教训 26](#_Toc74492865)

[6.2 针对以下方面的感受 26](#_Toc74492866)

# 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。

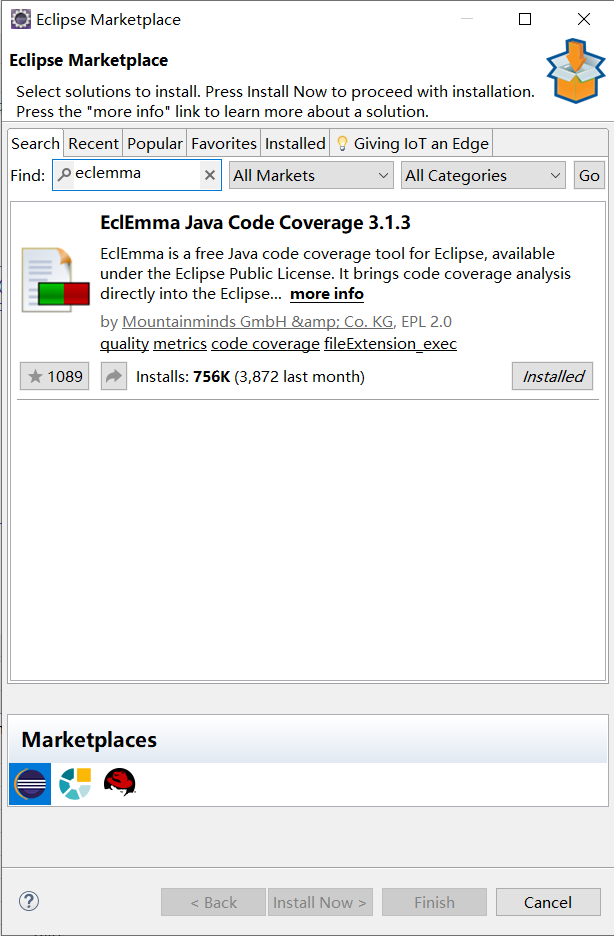
# 实验环境配置

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。

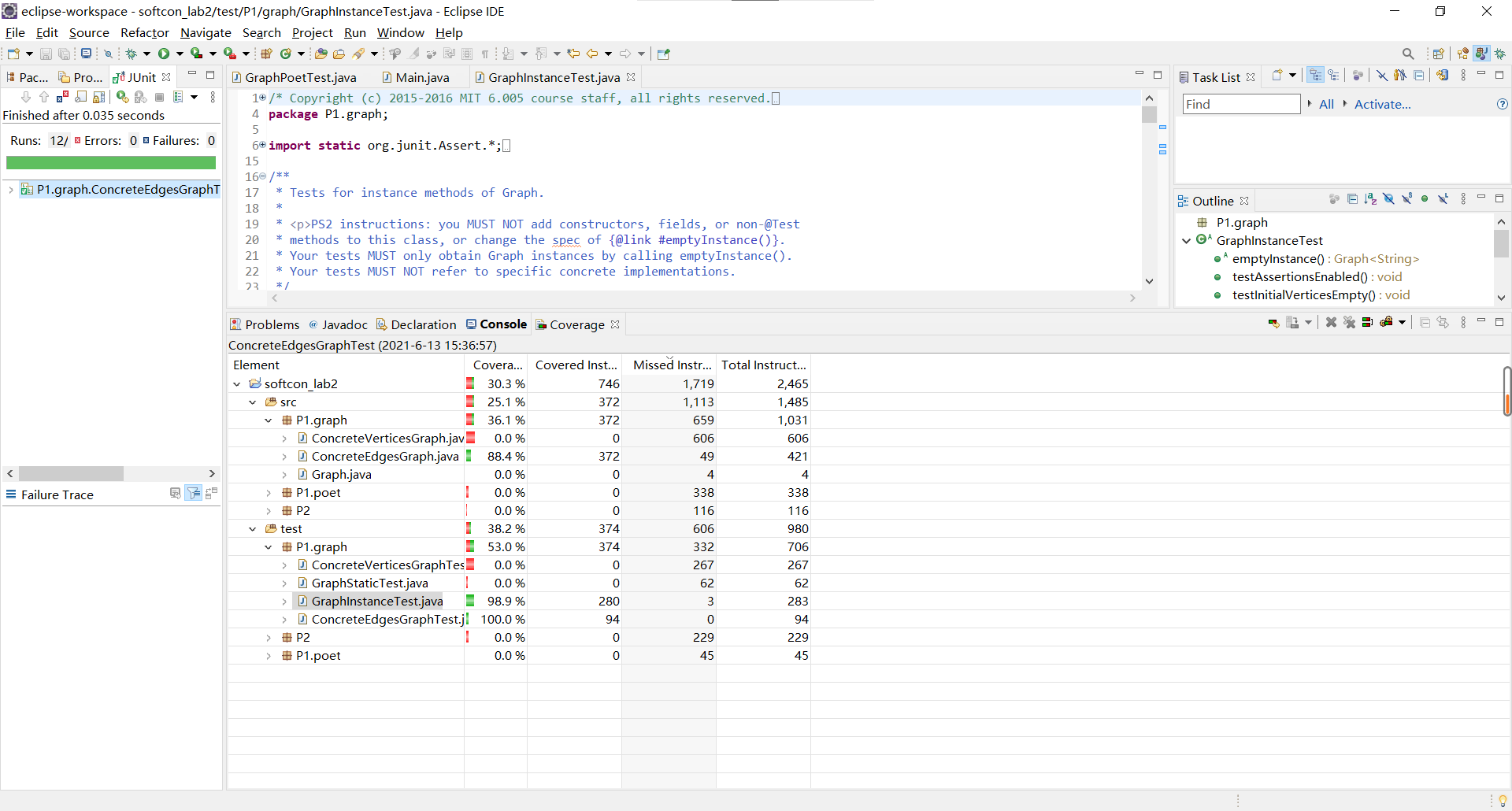
特别是要记录配置过程中遇到的问题和困难，以及如何解决的。

本次环境配置在lab1的基础上安装eclipse的Eclemma插件，

官网查询配置方法为内置的marketplace，如图所示：



安装后，进行代码覆盖度测试，每个Test类覆盖率均达到90%左右（把main函数代码注释），如图(ConcreteEdgesTest)：



在这里给出你的GitHub Lab2仓库的URL地址（Lab2-学号）。

https://github.com/ComputerScienceHIT/HIT-Lab2-1190201701

# 实验过程

## Poetic Walks

本实验通过先实现基于顶点与基于边描述的图，对顶点的标识先用String，而后采用泛型将标识符扩展到immutable类型，实现了面向接口的编程，一个图类型，两种具体实现，调用Graph.empty静态方法还可以避免具体实现的暴露。而后采用设计完备的Graph ADT进行PoeticWalks应用。

### Get the code and prepare Git repository

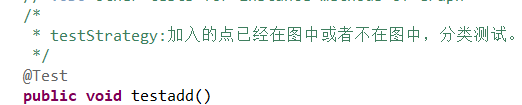
获取代码：git clone git@github:xxx.

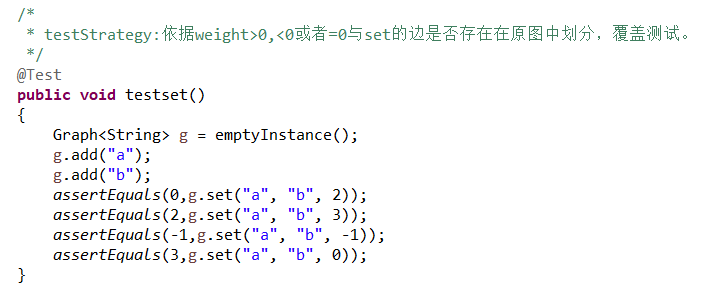
本地管理：git init git add git commit -m

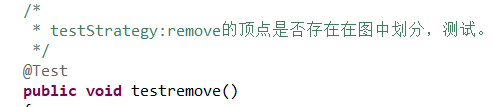
### Problem 1: Test Graph <String>

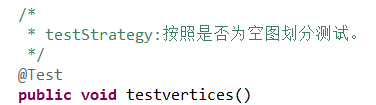
以下各部分，请按照MIT页面上相应部分的要求，逐项列出你的设计和实现思路/过程/结果。

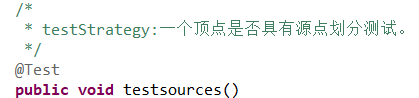
各个方法的测试策略：

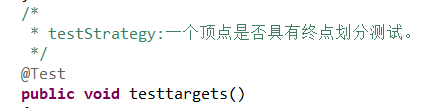




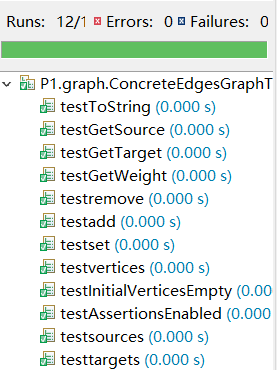


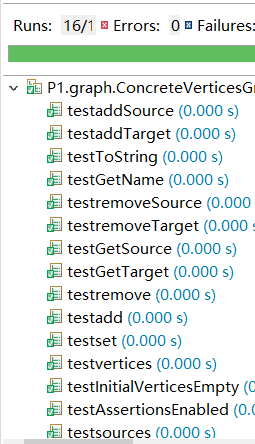






详细的测试关键代码与Concrete实现的两个类类似，3.1.3中有相类似代码，不赘述，测试结果（只能在Concrete类中得到测试，GraphInstanceTest是抽象类）：





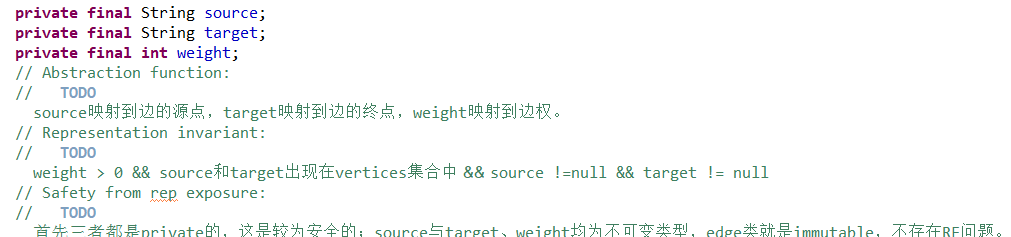
### Problem 2: Implement Graph <String>

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

设计思路：给定的rep为：****

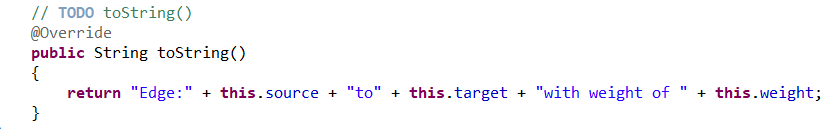
重点设计edge类即可。

Edge类设计如下：

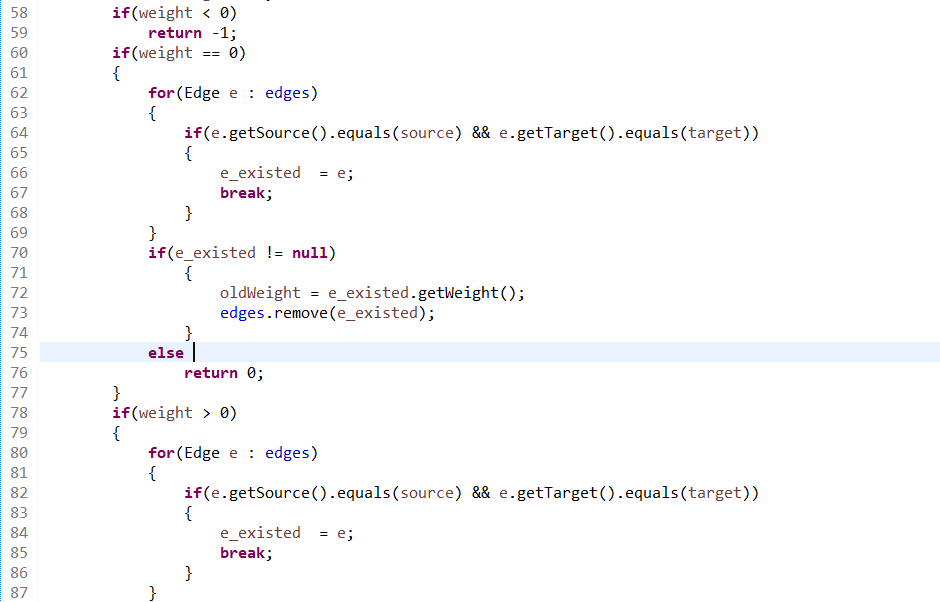
只需给定源点、 终点、权即可描述一条边。RI主要要求weight>0,source,target非空。

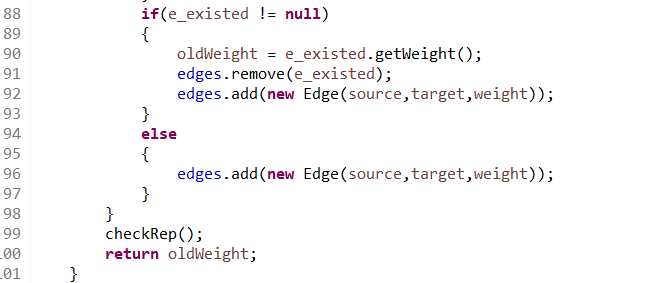
不存在表示泄露问题。其余是基本的get方法、构造器的设计，略。

为了更好地实现图上的tostring方法，Edge类的toString方法Override为：



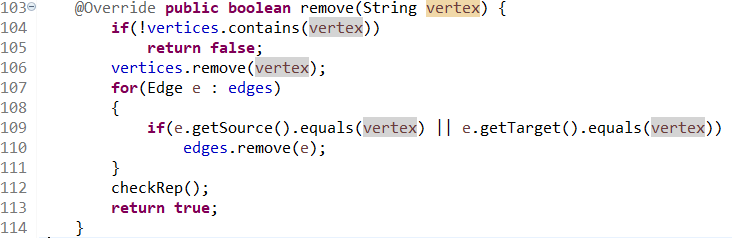
在此基础上实现ConcreteEdgeGraph类，设计如下：

Set方法（关键代码）：



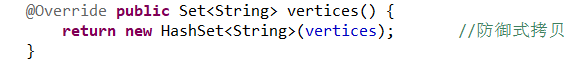
主要思想为：先将源点与终点加入图中，对权值分情况修改。权值小于0非法，权值=0只需将原有的边删去即可，若指定的顶点不在原有的图中，则不需要改动边集，加入顶点即可，oldweight=0；若权值>0,将原有的边删去后再加入新边，并更新oldweight（edge是immutable），同样的，若指定的顶点不在原来的图中，无需改动边集，增加新边即可。

Remove方法关键代码：



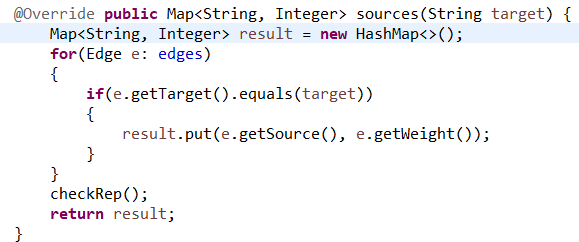
主要思想：若顶点本就不在图中，无需改动，返回false；若顶点在图中，首先删去顶点集中的该顶点，而后将边集中与该顶点相关联的边全部删去。

Vertices方法关键代码：



返回vertices属性即可，由于是可变对象，必须采取防御式拷贝。

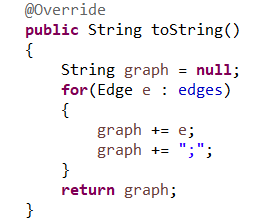
Source方法关键代码：



思想：在边集中查找所有终点是target的边，获取源点信息即可。

Targets相类似，不赘述。

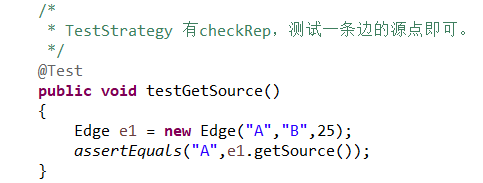
Tostring：



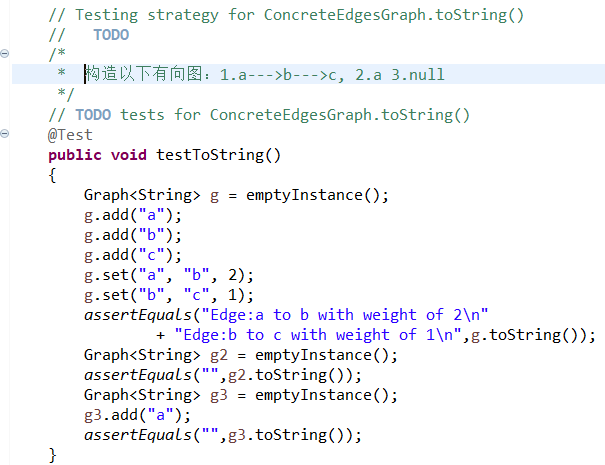
在重写edge的tostring方法的基础上，将每一条边都打印出来，即可表示一幅图。

ConcreteEdgesGraphTest:

EdgeTest:

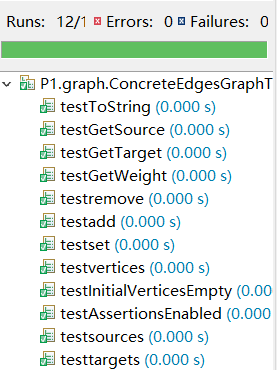


简易测试即可。



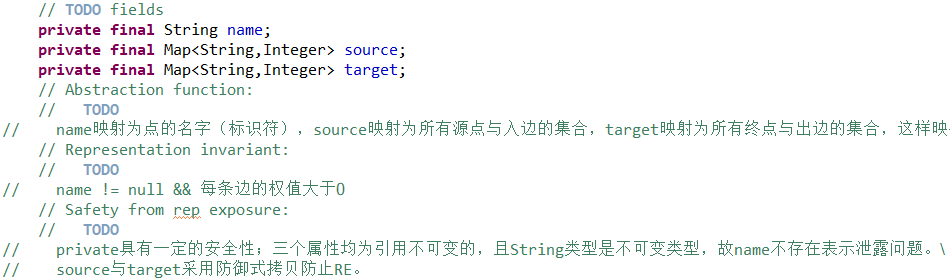
分别测试了正常图、单顶点图、空图。

测试结果如下：



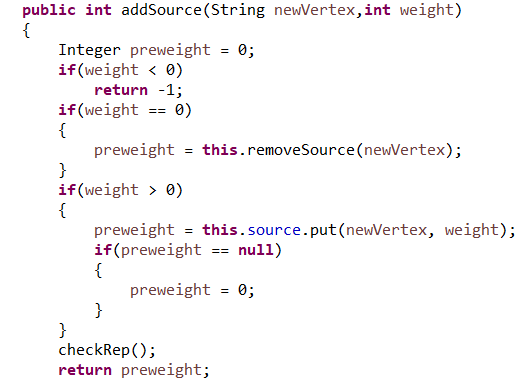
#### Implement ConcreteVerticesGraph

Vertex类的设计：



一个顶点包含一个名字（标识符），极其相关联的源点、终点信息。略去getter。

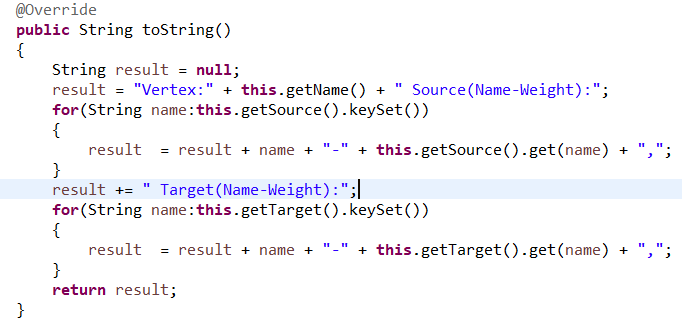
Addsource方法：



思想：为this顶点增加一个新的源点，返回旧权值。若weight==0，将newvertex从源点集中去除即可；若weight>0，覆盖原有/增加一个源点即可。

RemoveSource直接在源点集合中删去该顶点即可，返回oldweight。Addtarget与removetarget类似，不赘述。

Vertex类的tostring方法：

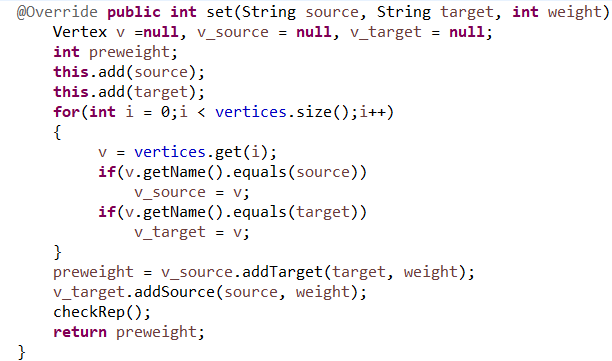


采用打印顶点名字+所有源点与终点信息的方式来打印该顶点。

在此基础上完成ConcreteVertexGraph类的设计：

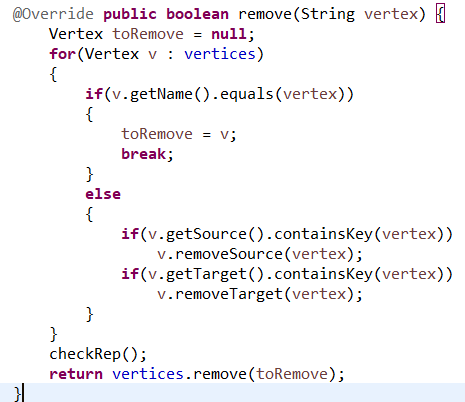
Add方法只需注意判断顶点是否已经加入图中，否则直接加入即可。

Set方法：



思想：首先加入source与target两个顶点，分别更新source与target的target与source属性即可，返回oldweight。

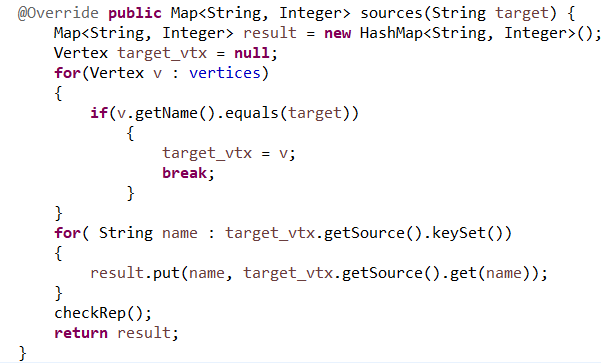
Remove：



关键点为在去掉一个顶点时应遍历所有其余所有顶点，删去所有与该顶点相关的边。

Vertices方法：直接返回所有顶点名称即可。

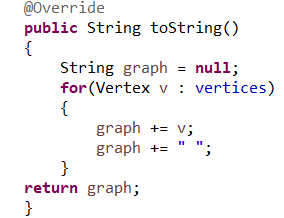
Sources方法：



思想：遍历该顶点的源点集合，依次复制即可，防御式拷贝防止RE（表示泄露）。

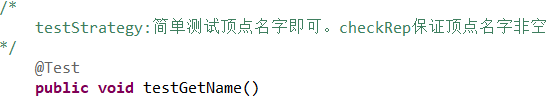
Targets方法类似不赘述。

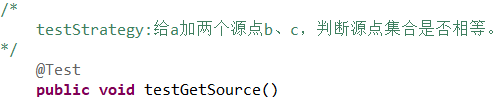
ConcreteVerticesGraph的tostring方法：

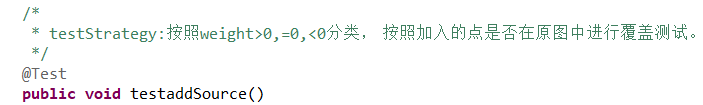


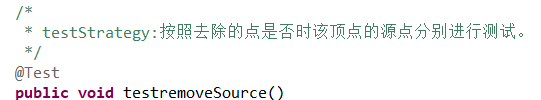
打印每个顶点信息，即可打印出图的结构。

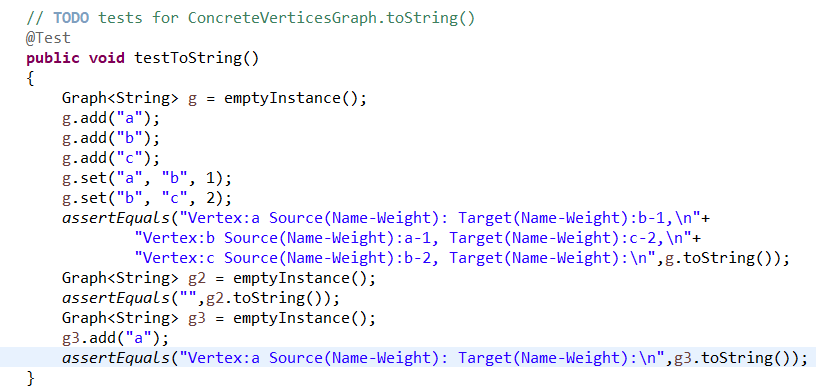
测试：Vertex类测试：





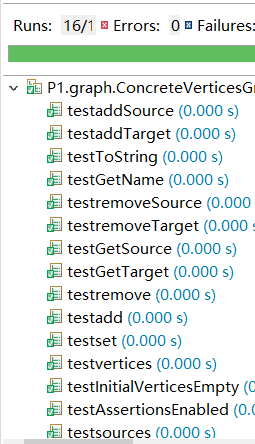






Tostring方法的测试仍然沿用a🡪b🡪c；a；空图三种测试方法。

测试结果如下：



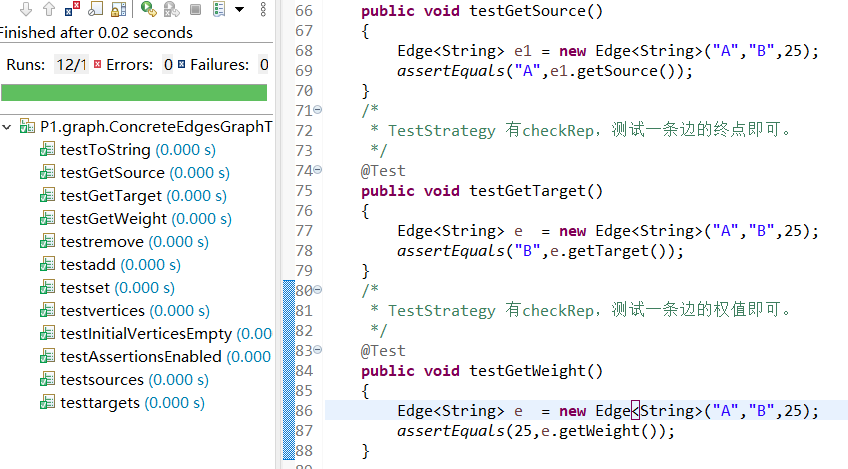
### Problem 3: Implement generic Graph<L>

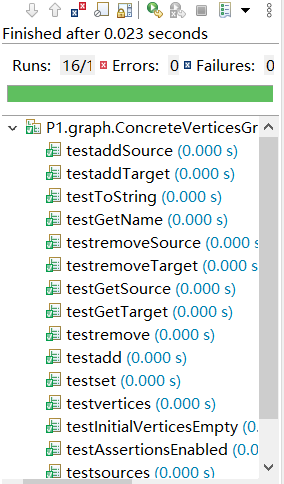
#### Make the implementations generic

实现方法较简单：ConcreteEdgesGraph<L>与将所有String替换为L，Edge应声明通配符<L>。

ConcreteVerticesGraph<L>也是类似做简单替换即可。

按照网站要求，测试用例应依旧能通过，如图所示：





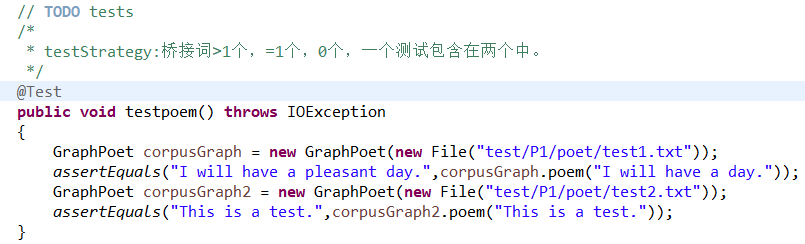
#### Implement Graph.empty()

采用ConcreteEdgesGraph<L>实现，并按照网站要求，对Integer作为immutable对象进行测试，测试策略以及结果如图：

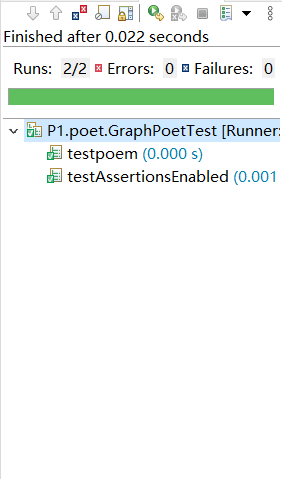


### Problem 4: Poetic walks

#### Test GraphPoet

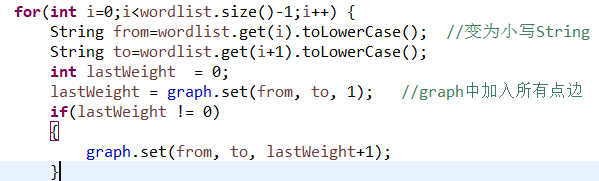
测试策略如下：

Test1测试有两个桥接词的情况，选择最大权的桥接词；test2测试没有桥接词，即返回原句。

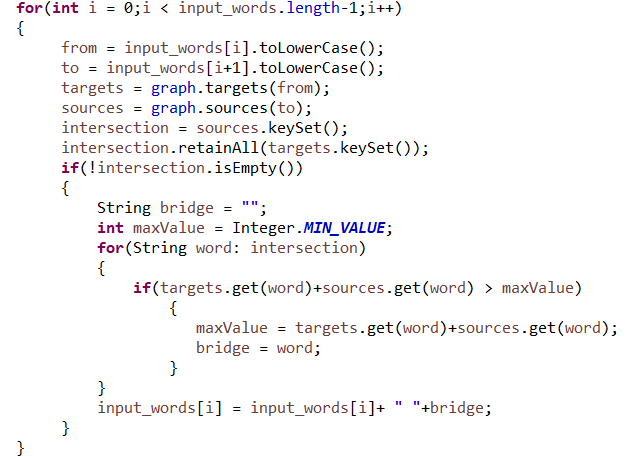
测试结果如图所示：

#### Implement GraphPoet

构造器关键代码：由语料库构造语料图时，遍历每一对词，将边权加1即可，采用set方法的返回值判断是否为0实现。



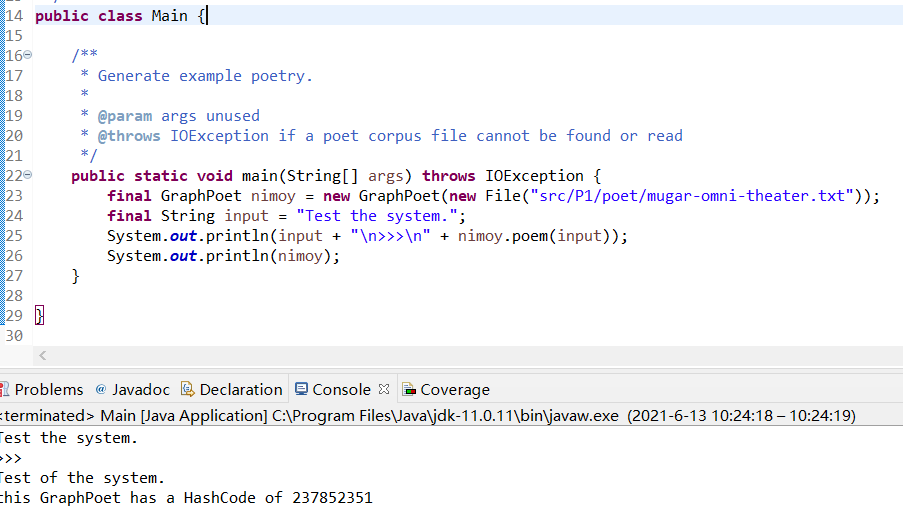
Poem关键代码：即遍历每一对词，寻找桥接词的过程：



首先对这对词的targets与sources做交集运算，若交集非空，则意味着有桥接词：接下来就是选择桥接词的过程，通过选择最大权实现，在选定桥接词后，将桥接词并入from中去（这里指在input\_words中的位置），而后做简单的拼接，结尾.字符的处理即可。

#### Graph poetry slam

测试结果如图：还测试了GraphPoem的tostring方法。



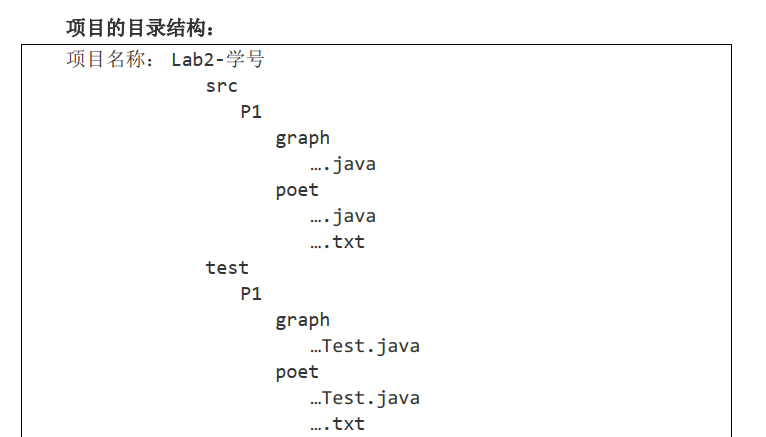
### Before you’re done

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。

Git init git add xxx git commit -m”” git push -u origin master

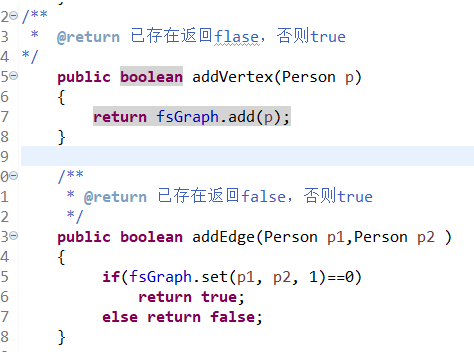
在这里给出你的项目的目录结构树状示意图。



## Re-implement the Social Network in Lab1

本实验为ADT的复用，利用3.1设计好的Graph类，再实现lab1中的社交网络，可以明显感觉到代码与逻辑简洁了许多。

### FriendshipGraph类



Addvertex与edge方法均调用Graph中相应方法，edge设置权为1即可。

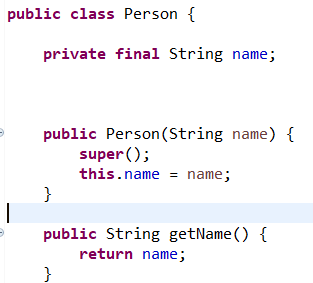
GetDistance代码：



思想：与lab1中相同，采用队列做BFS，不同点为此处visited用Set标记，将访问过的顶点全部添加至visited中，每次队首的熟人用targets方法与keyset方法获得。

### Person类

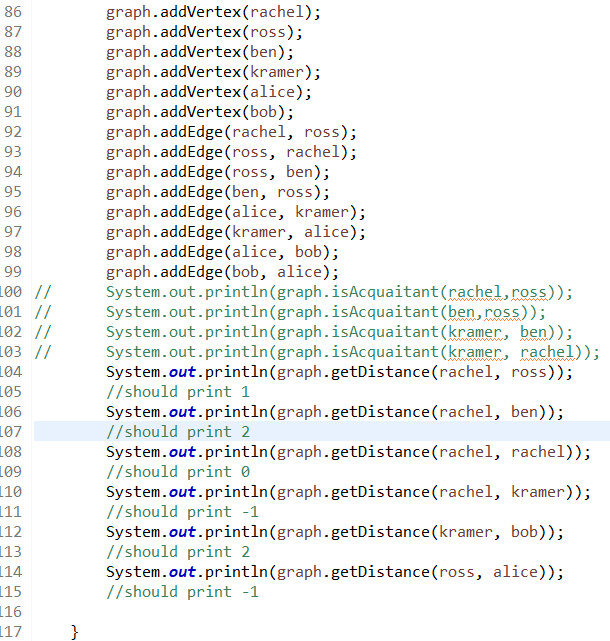
给出你的设计和实现思路/过程/结果。



删去了上次实验为了便于测试做的setter，加上了final，防止表示泄露。

### 客户端main()

给出你的设计和实现思路/过程/结果。



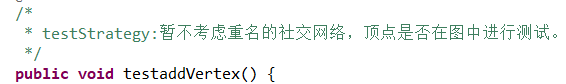
构造如下社交网络（假定为无向图，互相认识）

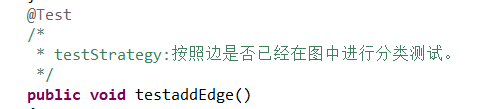
Rachel—Ross—Ben, Bob—Alice—Kramer.

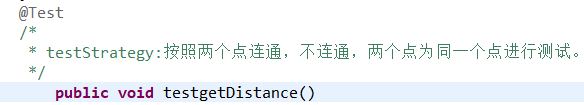
输出结果与lab1一致。

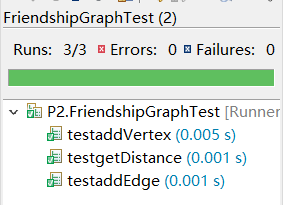
### 测试用例

给出你的设计和实现思路/过程/结果。







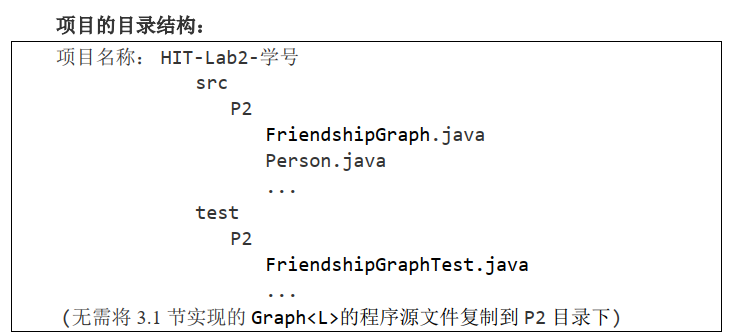
测试结果：

### 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 init git add xxx git commit -m”” 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 |
| 2021.6.5 | 21:20-22:50 | 完成ConcreteVerticesGraph类的设计 | 按时完成 |
| 2021.6.8 | 20:00-23:00 | 完成ConcreteEdgeGraph类的设计 | 按时完成 |
| 2021.6.12 | 19:30-22:00 | 完成poetric walks | 按时完成 |
| 2021.6.13 | 9:00-10:30 | 完善实验报告 | 按时完成 |
| 2021.6.13 | 11:00-12:30 | 完成re-implement in lab1 P3 | 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 |
| 在测试add方法时，不知道如何对比测试 | 查询后得知Set的equals是对应元素equals，从而进行对比测试。 |
| GraphPoet类设计时，不清楚如何构建语料库图，不明白边权具体含义以及桥接词可能有多个 | 仔细阅读javadoc以及网站说明。 |

# 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

在本次实验中发现很多集合类型的方法都是基于<L>的equals方法实现的，因此equals和hashcode方法就显得格外重要，要注意是否Override一个类的equals方法，以及@Override标识符的应用，避免方法变为重载方法。在本次实验中，编写代码的同时编写了大量的javadoc注释，AF、RI、Safety……感受到在OOP中每个方法，每个类的注释重要性，包括test中的strategy，以后还应该加强代码编写规范。

## 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？

最主要的差异体现在可复用性上，由于ADT的高度抽象，可以在多个场景下得到应用。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

使用泛型易于接受多种数据类型，增加了程序的可扩展性与复用性。

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？

不被具体的实现所约束，无论采用哪种实现，都应该能通过ADT的测试（抽象层面）。较不适应，仍然习惯于用测试用例检验。

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？

利用代码的重用性，大大降低了代码量、重复代码，使客户端代码简洁。其次，基于泛型的复用也体现了ADT的可扩展性，一次开发，多场景使用，代码的应用性更强。

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？

意义在于为使用类，使用方法的人员提供清晰的说明，大大地便利了对类、方法的使用，清晰的注释也使得客户端的误解减少，减少程序错误。愿意。

1. 关于本实验的工作量、难度、deadline。

工作量较大，难度始终，ddl合理。

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

到lab2，基本学习完成了ADT相关的基本概念，学习并实践了OOP，体会到面向对象抽象编程的强大之处，但感觉往往代码量大，且考虑的方面较多，概念多，需要注重细节的学习。暂无建议，期待相应设计模式的学习。