

**2018年春季学期  
计算机学院大二软件构造课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 任庆吉 |
| 学号 | 1160300422 |
| 班号 | 1603004 |
| 电子邮件 | 1091298335@qq.com |
| 手机号码 | 18846088521 |

**目录**

[1 实验目标概述 1](#_Toc507927440)

[2 实验环境配置 1](#_Toc507927441)

[3 实验过程 1](#_Toc507927442)

[3.1 Poetic Walks 1](#_Toc507927443)

[3.1.1 Get the code and prepare Git repository 1](#_Toc507927444)

[3.1.2 Problem 1: Test Graph <String> 1](#_Toc507927445)

[3.1.3 Problem 2: Implement Graph <String> 1](#_Toc507927446)

[3.1.3.1 Implement ConcreteEdgesGraph 2](#_Toc507927447)

[3.1.3.2 Implement ConcreteVerticesGraph 2](#_Toc507927448)

[3.1.4 Problem 3: Implement generic Graph<L> 2](#_Toc507927449)

[3.1.4.1 Make the implementations generic 2](#_Toc507927450)

[3.1.4.2 Implement Graph.empty() 2](#_Toc507927451)

[3.1.5 Problem 4: Poetic walks 2](#_Toc507927452)

[3.1.5.1 Test GraphPoet 2](#_Toc507927453)

[3.1.5.2 Implement GraphPoet 2](#_Toc507927454)

[3.1.5.3 Graph poetry slam 2](#_Toc507927455)

[3.1.6 Before you’re done 2](#_Toc507927456)

[3.2 Re-implement the Social Network in Lab1 2](#_Toc507927457)

[3.2.1 FriendshipGraph类 2](#_Toc507927458)

[3.2.2 Person类 3](#_Toc507927459)

[3.2.3 客户端main() 3](#_Toc507927460)

[3.2.4 测试用例 3](#_Toc507927461)

[3.2.5 提交至Git仓库 3](#_Toc507927462)

[3.3 The Transit Route Planner（选做，额外给分） 3](#_Toc507927463)

[4 实验进度记录 3](#_Toc507927464)

[5 实验过程中遇到的困难与解决途径 3](#_Toc507927465)

[6 实验过程中收获的经验、教训、感想 4](#_Toc507927466)

# 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

示外泄（ 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。

在这里给出你的GitHub Lab2仓库的URL地址（Lab2-学号）。

git@github.com:ComputerScienceHIT/Lab2-1160300422.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

图是一个比较基本的数据类型，本次实验通过实现图的数据结构来熟悉软件构造中ADT和OOP技术。

### Get the code and prepare Git repository

git clone https://github.com/rainywang/Spring2018\_HITCS\_SC\_Lab2

### Problem 1: Test Graph <String>

通过lab1的训练我知道了写test在软件构造中的必要性，而且应该是先写test，在我的理解，根据spec先写test可以避免测试样例陷入程序的逻辑中，达不到测试的目的。而且根据spec尽可能写出各种可能的测试样例，也可以使自己在实现方法时考虑得更周密些。

因为我们要实现Graph中的方法，add，set等等。所以先写出test。

Test strategy已经在注释中写清了，在这里在整理一下。

add()：这个函数返回boolean，所以测试样例中要包括这两种情况。添加重复点时返回false。还要考虑这个函数造成的影响是否是正确的，即这个点是否被添加到图中。

set():这个函数返回int，如果原来有这条边返回原来的边权，否则返回0；所以要设计至少return 0 ，return n（n>0)两种测试。还要考虑set的应用环境，可能图中有AB两点然后要添加AB这条边，或者修改AB的权，也有可能原来图中没有AB两点，对于这些情况都要设计至少一种测试。

Remove（）：删除图中一个点，如果有返回true，没有返回false。设计相应两种test即可。接下来考虑remove函数带来的影响，如果删去一个点A，那么与A相关的边都应当删去，所以要设计test来测试remove是否具有这种功能。

Sources和targets：test很好写，设计返回一个空map或者普通map进行测试。

### Problem 2: Implement Graph <String>

#### Implement ConcreteEdgesGraph

这是继承自Graph的一个类，表示用边来表示一个图。这里的边指的是Edge类的一个实例化。首先编写Edge类，根据OOP的原则，Edge类至少应该具有边的两个点，以及权重。

**private** String source;

**private** String target;

**private** **int** weight;

相应的写出observers

**public** String getSource() {

**return** **this**.source;

}

**public** String getTarget() {

**return** **this**.target;

}

**public** **int** getWeight() {

**return** **this**.weight;

}

这个类应该是immutable所以，不提供更改的方法。

然后重写toString，equals，hashcode。为Edge类编写checkRep。

然后编写ConcreteEdgesGraph类，根据OOP的原则，要想表示一个图，则这个类至少具有记录所有点，所有边的属性。

**private** **final** Set<String> vertices = **new** HashSet<String>();

**private** **final** List<Edge> edges = **new** ArrayList<Edge>();

图的结构就是这样，接下来实现这个抽象数据型的一些方法。

add：如果图中有这个点返回false，什么都不用做，如果没有就要把这个新点加入vertices中，返回true。

set：首先对输入set（A，B，n）分类。第一种，A，B两点已在图中，但没有这条边，则构造出Edge（A，B，n）添加到edges的list中。第二种，如果A或B不在图中，则在vertices的set中添加。然后重复第一种情况。第三种，n==0，遍历edges的list找到AB边，然后删掉。第四种，AB存在，n！=0，则遍历edges的list找到AB边然后更改权。具体实现见代码。

remove：函数功能删掉某个点，及相关的边。首先判断目标点是否已存在，不存在就不用做操作了。如果存在则先在vertices中删去，然后遍历edges找到跟他相关的边删掉。

vertice：返回由图中所有点构成的集合。

targets：找到某一点e能到达的所有点，以及权重，构成一个map。遍历edges，如果某个edge的source等于e，则把edge的target，weight放到映射。

sources：找到某一点e，由哪些点可以到达。遍历edges，如果某个edge的target等于e，则把edge的source和weight放到映射。

#### Implement ConcreteVerticesGraph

与ConcreteEdgesGraph类似，这次用点来组成一个图。用OOP的思想来设计，如果每个点都能记录他所能到达的点，那么通过记录这些点就可以得到整个图。

首先我们设计Vertex类，根据我们的设计，这个类，除了要记录这个点的名字，还要保存与其他点的关系。这里我采用记录他能到达的点，当然也可以记录能到达他的点。

**private** String name;

**private** Map<Vertex, Integer> Edges = **new** HashMap<Vertex, Integer>();

observers和mutators

**public** Map<Vertex<L>, Integer> getEdges() {

Map<Vertex<L>, Integer> edges = **new** HashMap<Vertex<L>, Integer>();

edges.putAll(Edges);

**return** edges;

}

因为要防泄露，所以返回了edges的一个复制。

**public** **void** changeEdges(Map<Vertex<L>, Integer> m) {

Edges.clear();

Edges.putAll(m);

}

提供了一个更改edges的方法。

ConcreteVerticesGraph只需记录点的集合

**private** **final** List<Vertex<L>> vertices = **new** ArrayList<>();

接下来仍然是实现图的基本操作。

设计原理和上一个没有什么不同，但因为图的实现变了，所以一些操作也会变。

add：先判断vertices中有没有这个点，如果没有就new一个vertex加进去。

set：仍然是分那几种情况考虑，区别是在更改边时要通过更改点的属性，而不是在图里操作。

remove：遍历vertices中的每个点，如果当前点是要删的v，把他从vertices删掉，如果不是，遍历当前点的边，如果有跟v相关的点都要删去。

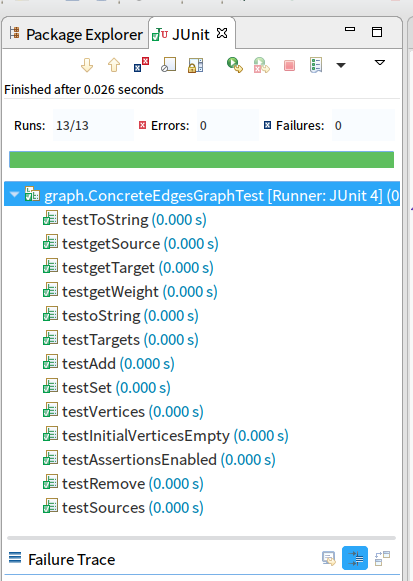
vertices：遍历vertices把所有点放在一个集合。

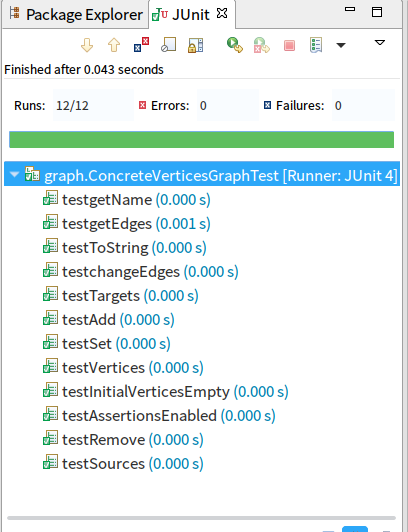
targets：这个很好写，因为我们设计Vertex类时就是这么做的，首先遍历vertices找到目标点然后把他的edges属性返回

sources：正好跟我们设计Vertex类时的想法相反，所以比较麻烦。需要遍历vertices，对于每一个点都要遍历他的edges属性，寻找目标点v，如找到就把当前点和weight放到map。

还要写一些toString，checkRep等等，不具体说了。

下面是测试截图：





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

#### Make the implementations generic

之前的Graph是用String来存储的，具有一定局限性，我们想用各种类做图的顶点、边就能在任何地方使用。所以使用泛型L，根据之前的进行一点修改就可以了。

#### Implement Graph.empty()

随便选择一种之前实现的方法来生成图就可以了。

**public** **static** <String> Graph<String> empty() {

**return** **new** ConcreteEdgesGraph<String>();

}

### Problem 4: Poetic walks

这是一个应用图结构的例子。把一句诗看作是由词构成的图。诗中所有出现的词就是点，两个词挨着就是一条边，边出现的次数就是边的权重。

#### Test GraphPoet

先写test，对于constructor，编写一个测试文档，看是否能生成一个正确的graphPoet即可。但要强一点，增加一些特殊情况，比如某边多次出现。出现了环，出现了回退边等等。

对于poem，要检测，他是否能根据某一输入得到正确的诗句。测试样例应包括，可以添词，不可以添词，大小写，多种添词可能，等几种情况。

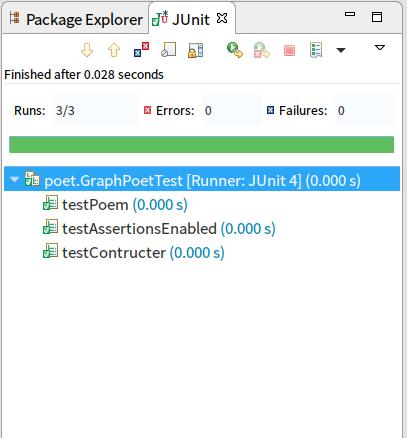
#### Implement GraphPoet

GraphPoet中一共有两个方法，constructor和poem，constructor是根据一个文件中的语句生成一个图，原理就是上面解释的。首先将文件读进来，然后利用图的set方法把边一条一条添加到图中。

poem方法是根据已生成的语料库，把你的输入丰富变成一首诗。插入单词的原则是，对于输入，A B，寻找是否存在点C，使得AC，CB两条边都存在，如果存在多个C，则选择一个AC和CB权重之和最大的。

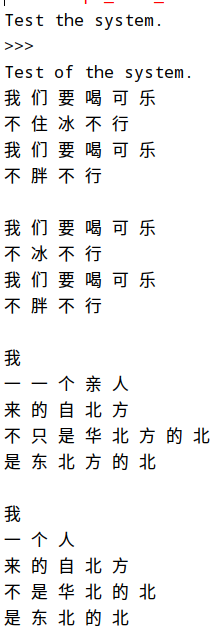
实现方法就是这样，对于每两个相邻的词AB去生成的graphPoet中寻找合适的C。寻找C可以利用图的sources和targets方法，如果某个点的targets包括B，sources包括A就符合条件，找到所有符合条件的再选出最合适的。当然如果图中不包括A或B也就没必要寻找C了，一定找不到。

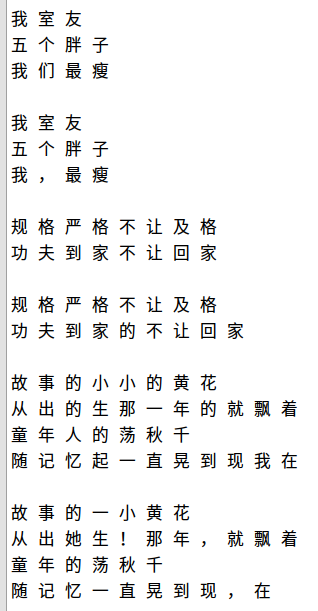
测试结果：

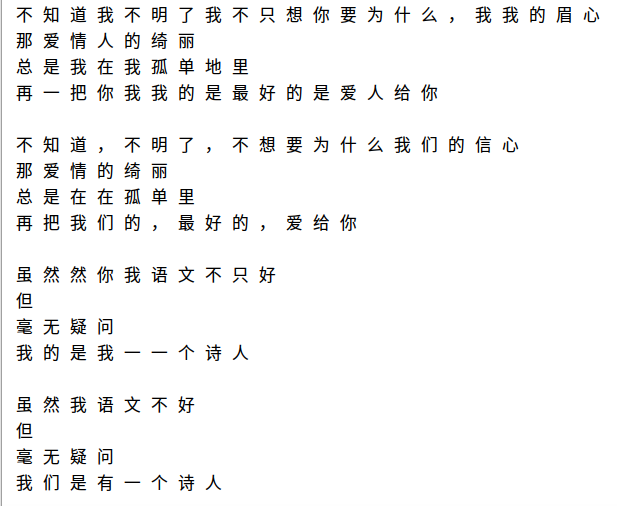


#### Graph poetry slam

可以自己制作一些语料库，看看能输出什么。我在网上找了海子和徐志摩的一些诗，格式化处理后生成graphPoet，然后自己编写了一些输入，得到一些有趣的东西。





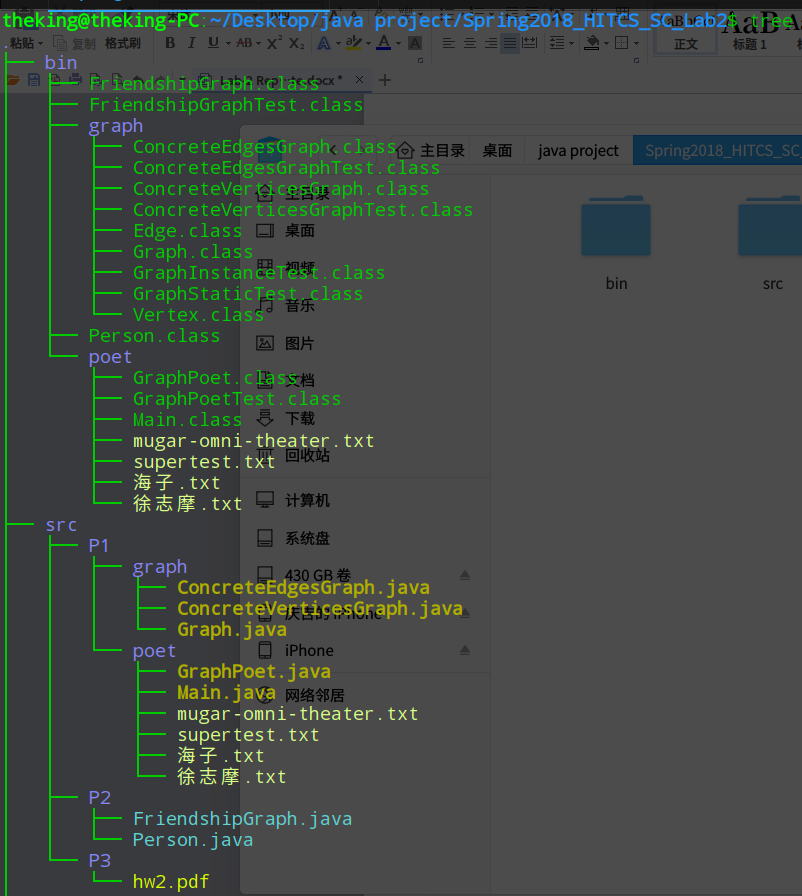


### Before you’re done

git push git@github.com:ComputerScienceHIT/Lab2-1160300422.git

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

因为已经实现了Graph的泛型实现，所以将L用Person来实现就是社交图了。

### FriendshipGraph类

**private** **final** Graph<Person> graph = Graph.*empty*();

用之前的方法来生成空图，接下来不用改什么，就用原来的就好。

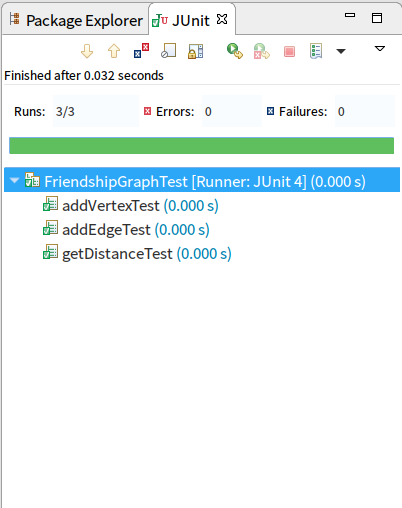
### Person类

并没有修改什么，只是学过override后，我重写了Person类中的equals，hashcode等方法。

### 客户端main()

用之前的了。

### 测试用例



### 提交至Git仓库

完全同上。

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。

## The Transit Route Planner（选做，额外给分）

请自行组织本节内的目录结构。

# 实验进度记录

请尽可能详细的记录你的进度情况。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 3.26 | 下午+晚上 | graph的test | 完成 |
| 3.29 | 晚上 | concreteEdgeGraph | 完成 |
| 3.30 | 下午+晚上 | concreteEdgeGraph | 完成 |
| 3.31 | 下午 | Graph<L> | 完成 |
| 4.3 | 下午 | 写注释 | 完成 |
| 4.4 | 下午 | graphPoet的test | 完成 |
| 4.5 | 下午 | graphPoet | 完成 |
| 4.7 | 下午一小会 | friendshipgraph | 完成 |

# 实验过程中遇到的困难与解决途径

本次实验遇到最多的困难是对于RI、AF等名词的理解，事实上这些东西是软件构造过程中必须认真考虑的，我就是一开始没看这些东西，直接根据自己的想法编程，后来发现好多漏洞和bug，改起来很麻烦。但是，由于课程进度与实验进度的不匹配，这种情况是难免的，老师第一节课就提醒过我们这种情况。但是在不断试错的过程，记忆会更深。就像第一次实验我还不知道先写test，但这一次实验我已经很清楚了，先写test有相当多的好处。

# 实验过程中收获的经验、教训、感想

本节除了总结你在实验过程中收获的经验和教训，也可就以下方面谈谈你的感受（非必须）：

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？ 面向应用场景编程需要将实际问题抽象，写清楚抽象函数。面向ADT编程，先不用考虑实现，只需先考虑，要实现哪些功能。
2. 使用泛型和不使用泛型的编程，对你来说有何差异？使用泛型编程应用的场景更多，不用泛型有局限性。
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？优势很多，我在报告中说的很多，简而言之，先写test可以更好的帮你思考用户的需求，想到方方面面，使编程更容易。
4. 本实验设计的ADT在三个应用场景下使用，这种复用带来什么好处？省时省力不容易出bug
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？这些问题如果不想清楚，在编程过程中就会陷入停滞，甚至要推倒重来，这个代价是相当大的，坚持写注释，可以帮助自己思考这些问题，而且在写大项目时，时不时回来看看，可以帮助自己理清思路，我会坚持。
6. 关于本实验的工作量、难度、deadline。有点难，附加没写完。
7. 《软件构造》课程进展到目前，你对该课程有何体会和建议？希望在每次实验结束后，可以把助教用来检查的测试样例发给我们，或者是一份符合要求的源码，我不想只要一个分数，更希望通过一次次实验，找到自己的不足，不断进步。