

**2021年春季学期  
计算学部《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 王子奕 |
| 学号 | 1190200121 |
| 班号 | 1903001 |
| 电子邮件 | 654483577@qq.com |
| 手机号码 | 15531580911 |

**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 1](#_Toc72249753)

[3.1.1 Get the code and prepare Git repository 1](#_Toc72249754)

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

[3.1.3 Problem 2: Implement Graph <String> 1](#_Toc72249756)

[3.1.3.1 Implement ConcreteEdgesGraph 2](#_Toc72249757)

[3.1.3.2 Implement ConcreteVerticesGraph 2](#_Toc72249758)

[3.1.4 Problem 3: Implement generic Graph<L> 2](#_Toc72249759)

[3.1.4.1 Make the implementations generic 2](#_Toc72249760)

[3.1.4.2 Implement Graph.empty() 2](#_Toc72249761)

[3.1.5 Problem 4: Poetic walks 2](#_Toc72249762)

[3.1.5.1 Test GraphPoet 2](#_Toc72249763)

[3.1.5.2 Implement GraphPoet 2](#_Toc72249764)

[3.1.5.3 Graph poetry slam 2](#_Toc72249765)

[3.1.6 使用Eclemma检查测试的代码覆盖度 2](#_Toc72249766)

[3.1.7 Before you’re done 2](#_Toc72249767)

[3.2 Re-implement the Social Network in Lab1 2](#_Toc72249768)

[3.2.1 FriendshipGraph类 2](#_Toc72249769)

[3.2.2 Person类 3](#_Toc72249770)

[3.2.3 客户端main() 3](#_Toc72249771)

[3.2.4 测试用例 3](#_Toc72249772)

[3.2.5 提交至Git仓库 3](#_Toc72249773)

[4 实验进度记录 3](#_Toc72249774)

[5 实验过程中遇到的困难与解决途径 3](#_Toc72249775)

[6 实验过程中收获的经验、教训、感想 4](#_Toc72249776)

[6.1 实验过程中收获的经验和教训 4](#_Toc72249777)

[6.2 针对以下方面的感受 4](#_Toc72249778)

# 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

Java的环境以及Git的环境在上次实验已经配置好。

在这里给出你的GitHub Lab2仓库的URL地址（Lab2-学号）。

<https://github.com/ComputerScienceHIT/HIT-Lab2-1190200121>

# 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

实验的主要目的是测试ADT的规约设计和ADT的多种不同的实现，且ADT的规约已经给出。

要求实现的ADT是一个能存储图的数据结构，分别利用用点建图和用边建图。

### Get the code and prepare Git repository

### 从https://github.com/rainywang/Spring2021\_HITCS\_SC\_Lab2/tree/master/P1将这个项目的代码Clone到本地。

通过 git init指令新建工作区，使用git commit提交更改，然后使用git remote add origin [git@github.com:ComputerScienceHIT/Lab2-1190200121.git](mailto:git@github.com:ComputerScienceHIT/Lab2-1190200121.git) 命令与远程仓库关联。之后可以使用git push与git pull指令与远程仓库交互。

### Problem 1: Test Graph <String>

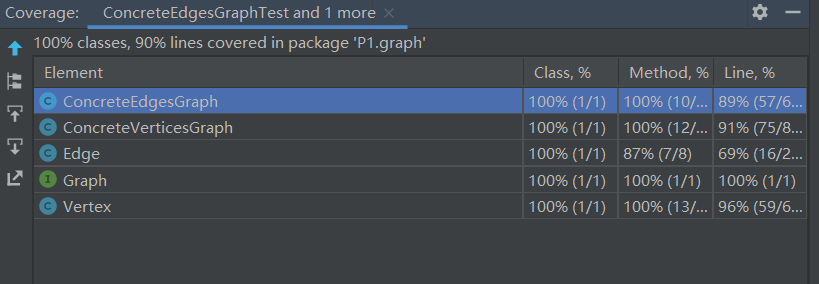
该设计主要是测试empty()函数的testing strategy。此处的测试主要就是测试在泛型L的情况下，只要L为immutable类型的数据类型就可以使用。所以测试策略就是使用不同的immutable类型的数据对于图进行构造，检测空图和已加入点边的图，此处选择Integer和String以及empty进行测试.

以下为instance方法的测试策略

主要采用检测点和边在修改前后是否在图中

|  |  |
| --- | --- |
| test add | /\*\*  \* Test for {@link P1.graph.Graph#add(Object)}  \* covers:  \* vertex A in the graph  \* vertex A not in the graph  \*/ |
| test  Set | /\*\*  \* Test for {@link P1.graph.Graph#set(Object, Object, int)}  \* covers:  \* set edge existing with the weight of one  \* change set edge existing with the weight of two  \* remove set edge not existing with weigth zero  \* set edge existing with the weight of zero  \*/ |
| test Remove | /\*\*  \* Test for {@link P1.graph.Graph#remove(Object)}  \* covers:  \* remove a vertex not existing  \* remove a vertex existing with edges  \* remove a vertex existing without edges  \*/ |
| test vertices | /\*\*  \* Test for {@link P1.graph.Graph#vertices()}  \* covers:  \* create an empty graph  \* create a graph with two vertices  \*/ |
| test source | /\*\*  \* Test for {@link P1.graph.Graph#sources(Object)}  \* covers:  \* make a graph  \* positive number of edges connected to the target  \* test the graph  \*/ |
| test target | /\*\*  \* Test for {@link P1.graph.Graph#targets(Object)}  \* covers:  \* make a graph  \* positive number of edges connected to the source  \* test the graph  \*/ |

代码测试覆盖率如下（IDEA自带代码覆盖率测试工具）



### Problem 2: Implement Graph <String>

#### Implement ConcreteEdgesGraph

首先需实现immutable辅助类Edge边，其包含三个元素分别为：起点source、终点target和边权weight。对于表示不变量的约束是权重必须为正整数（零和复数都是不合法的）同时起点和终点不能相同（即不能有自环）。为此实现了getter函数，并重载equals和toString。

之后来实现mutable类ConcreteEdgeGraph，其中包含两个元素：点集vertices和边表edges（list）。同时约束是满足Edge的不变量约束且边的起点和终点均在点集中。

其中对ConcreteEdgesGraph的保护是将vertices和edges有关的变量都设置为private final类型，目的是防止外部直接修改。

此外需要实现的函数有add、set、remove、sources、vertices和targets。同时还需要重载toString函数。

set函数首先判断边权是否非负，之后在edges中查找是否有起终点符合条件的边。如果有，则记录旧边权，更改或删除对应的边；如果没有，则添加（边权为正数时）一条新的边。由于可能修改图则需要调用checkRep函数，最后返回旧边权（若图中原本没有对应的边，则返回0）。

add函数通过向点集vertices中加入点，同样由于修改之后checkRep，返回Set.add函数的返回值。

remove函数则是首先在点集vertices中删除对应的点（如果点不在vertices中，则直接返回假值），之后将包含要删除的点的边从图中删除，同样由于修改之后checkRep，返回真值。

vertices函数目的返回点集vertices的一个复制。

sources和targets函数的实现类似，查找target（source）属性与查找的点相同的边，将它们置于一个Map中，之后返回。

#### Implement ConcreteVerticesGraph

首先需要实现一个mutable辅助类Vertices，其包含三个field：点对应的属性label，起始边集source和终点边集target。对于表示不变量的约束是属性不是null和所有的边权为正数。为此实现了getters和setters函数，同时重载toString函数。

之后来实现mutable类ConcreteVerticesGraph，其包含一个元素：点集vertices。

对ConcreteVerticesGraph的保护是将vertices有关的变量都设置为private final类型，防止外部直接修改。

需要实现的函数有add、set、remove、vertices、sources和targets。此外还需要重载toString函数。

set函数首先判断边权是否非负，之后在source点的边中查找是否有终点符合条件的边。如果有，则纪录旧边权，更改或删除对应的边；如果没有，则添加（边权为正数时）一条新的边。同理于target点。由于修改图则需要调用checkRep函数，最后返回旧边权（如果图中原本没有对应的边，则返回0）。

remove函数则是首先在点集vertices中删除对应的点（如果点不在vertices中，则直接返回假值），之后将包含要删除的点的边从其它的点中删除，之后checkRep，返回真值。

add函数首先在点集vertices中搜索是否存在。如果存在，则返回假值；如果不存在则添加新点，之后checkRep，返回真值。

vertices函数返回点集vertices的label的一个复制。

sources和targets函数的实现类似，直接返回target（source）的sources（targets）。

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

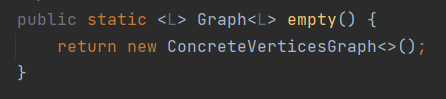
利用各种IDE中提供的替换工具将已有的两个Graph<String>的实现改为基于Graph<L>的实现，

#### Make the implementations generic

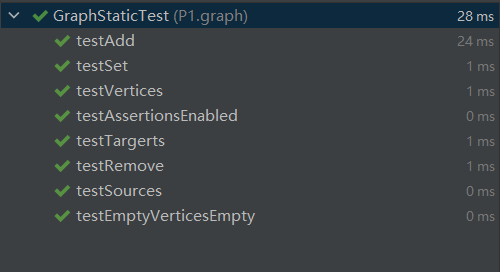
直接将string类型全部改为泛型实现即可，然后在更改结束后，重新测试ConcreteVerticesGraphTest和ConcreteEdgesGraphTest两个测试，若Graph<String>在两个泛型实现下仍然可以通过，则可表示更改成功。

#### Implement Graph.empty()

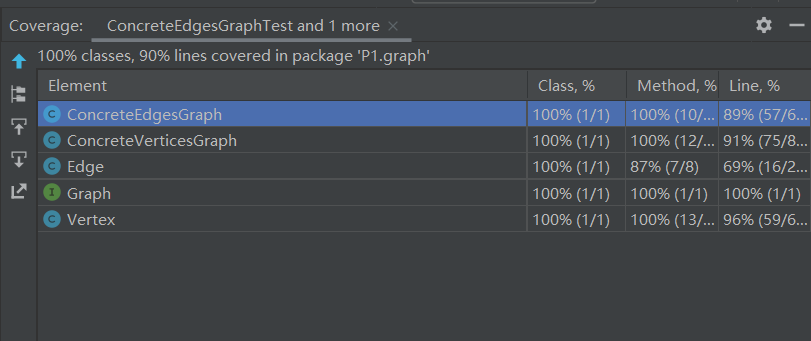
要求生成空图实例，可选择ConcreteEdgesGraph作为返回对象



使用不同的L泛型如integer进行测试，由于基本数据类型均是Immutable类的，主要可以利用这个进行测试。具体的测试结果如下：



代码覆盖率

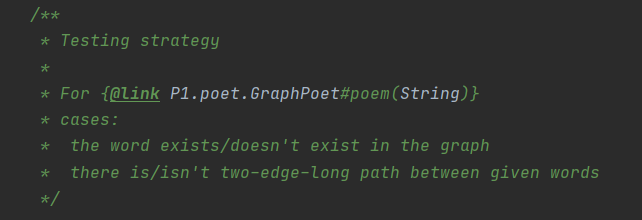


### Problem 4: Poetic walks

该任务根据一段诗生成其对应的图，然后根据这个生成的图再生成更多的诗，利用相同的搭配之间可能有的“桥梁词汇”来修饰更改。

#### Test GraphPoet

测试策略为：给定的两个词是否存在于图中；且两个词间是否存在一条长度为2的路径。



#### Implement GraphPoet

实现GraphPoet主要是实现Constructor和Poem两个函数。

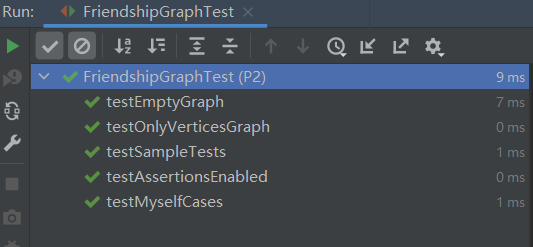
1、对于GraphPoet的构造器

将文件中的诗歌按照行读入，利用空格将其分开，去掉空的字符串，在任何相邻的两个中间加入一条边，并按照边出现的数量更新权值建图即可。

2、对于poem函数

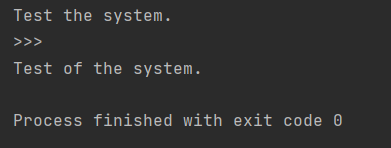
主要利用已有的诗歌生成的图，在输入的input字符串中增加bridge word。根据Graph<L>已有的target和source函数进行判断，前面的词的target和后面的词的source如果有重合的词，从中读取最大权值的一边作为bridge word加入即可。

测试结果



#### Graph poetry slam

利用GraphPoet生成一个诗歌进行测试，计划利用mugar-omni-theater作为生成Graph的选择。然后在input中输入“Test the system”，经过代码处理得到输出



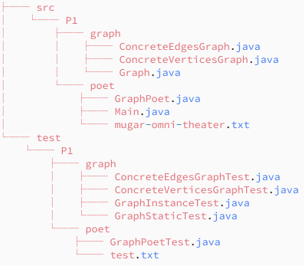
### 使用Eclemma检查测试的代码覆盖度

### 

### Before you’re done

使用git add将文件添加到工作区，用git commit提交更改。由于之前已经将本地仓库已经和GitHub远程仓库关联，使用git push即可将本地代码提交到GitHub。

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

### 需要实现Lab1中的Social Network，利用在P1中已经写好的Graph<L>接口来设计

### FriendshipGraph类

FriendShipGraph类主要是完成朋友圈的建立构造addVertex、addEdge和getDistance三个函数。

1、addVertex函数

利用Graph<L>里面提供的函数add，即可增加顶点。

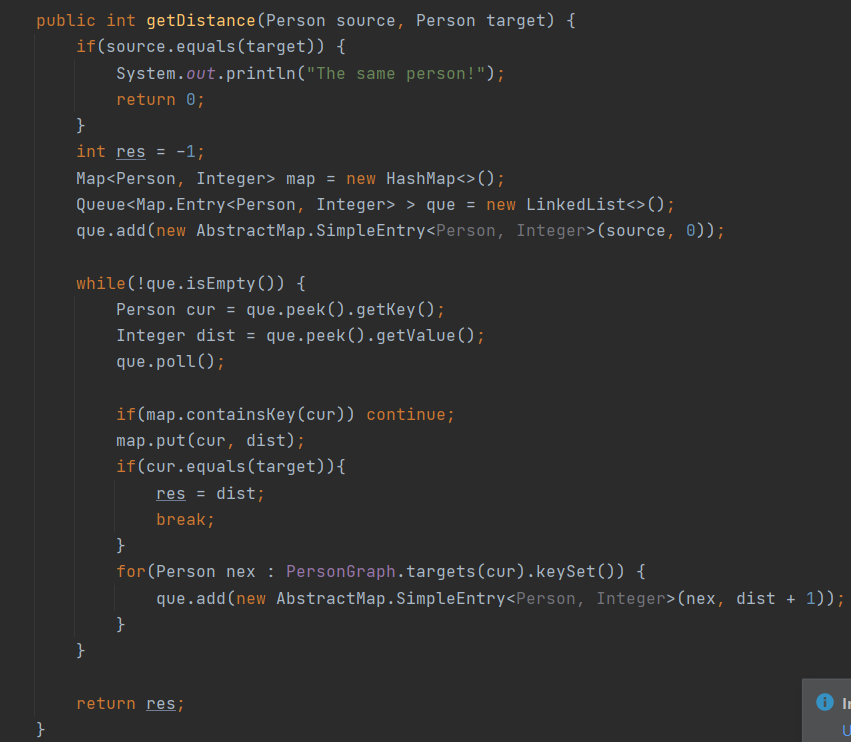
2、addEdge函数

充分利用Graph<L>里面提供的函数set，即可增加边。需要先判断自环，再利用set(person1, person2,1)增加边即可。

3、实现getDistance函数

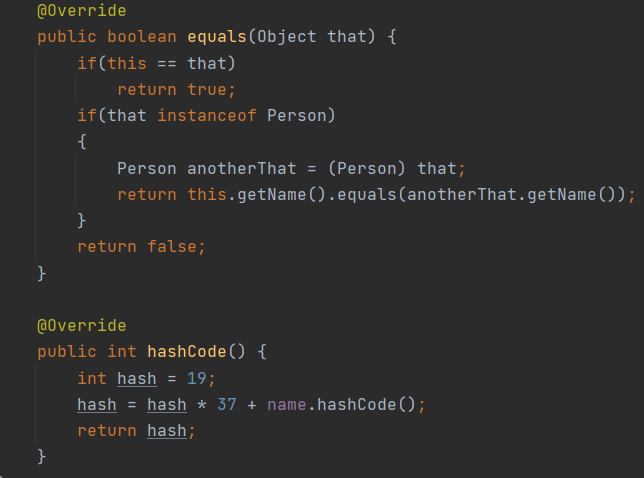
getDistance 函数得到两个人之间的最短距离：

实现此功能是基于BFS求最短路径的方法。首先我们需要知道若两个Person对象为同一个，则返回0。然后定义一个Map集合map和一个Person队列queue，队列queue用来储存广搜的遍历结果，map的Map集合用来储存广搜的所有元素及他们与第一个元素的距离。具体实现方法为首先将第一个元素source入队，并且把第一个元素source和下标0入集合（此处可利用键值对实现），当队列非空时，弹出队首元素top，并且得到top在集合map中的下标distance，若当前已经到达终点，则直接弹出循环反汇。反之则便利所有该店在图中指向的点再入队进行dis+1。如果直到队列为空还没找到target，则返回-1



### Person类

Person类中只需要有一个field为某一个人的名字即可。在此处需要重写equals和hashCode两个函数，可以在FriendshipGraph中对于Person类的判断。



### 客户端main()

可直接利用Lab1中已有的main客户端即可，因为所有Lab2中的所有实现仍是利用之前的addVertex、addEdge和getDistance三个函数

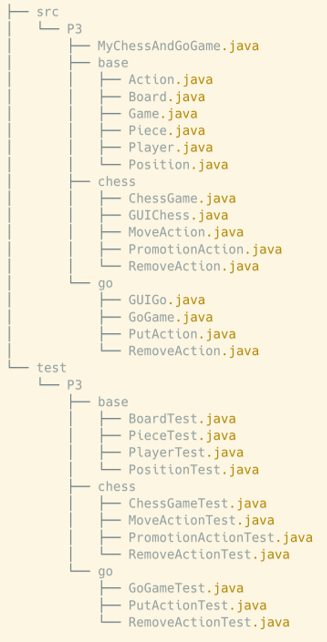
### 测试策略主要是根据FriendshipGraph中的图的类型进行测试，对空图、无边图、个人设计图的测试及lab1中的测试数据，来验证在不改变客户端的情况下仍然可以使用。测试结果

### 

### 提交至Git仓库

同上述指令

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 |
| 6.1 | 18:00-19:30 | 完成P1 Problem1 | 完成 |
| 6.2 | 20:00-21:30 | 完成P1 Problem2 | 完成 |
| 6.3 | 18:00-19:30 | 调试P1 Problem2 | 完成 |
| 6.4 | 20:00-21:30 | 完成P1 Problem3 | 完成 |
| 6.5 | 18:00-19:30 | 完成P1 Problem4 | 完成 |
| 6.6 | 22:00-23:00 | 完成P2 | 完成 |
| 6.7 | 20:00-22:00 | 完成P3 base | 完成 |
| 6.8 | 20:00-22:30 | 完成P3 chess | 未完成，存在Bug |
| 6.9 | 20:00-22:30 | 完成P3 go | 完成 |
| 6.10 | 20:00-23:30 | 完成P3收尾 | 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 |
| FriendshipGraph无法继承ConcreteVerticesGraph<Person>类 | 通过编译器提示，发现是构造函数未添加public关键字 |
|  |  |
|  |  |

# 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？

ADT的抽象程度更高，可以降低代码的耦合度。在使用ADT进行编程时，前期的规划会花费更多的时间，但是编程时的难度会下降，而且更利于复用。而直接面向应用场景编程前期的编写会比较快，但是当程序较大时不利于维护。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

使用泛类的编程可以更加灵活，使代码的复用更加容易。如果不使用泛类，则会增加很多重复的代码。

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？

优势是这样的方法可以纯粹的为规约编写测试用例，可以覆盖绝大部份情况，同时之后编写代码时的目的性可以更强。这种测试方法的确很优秀，但是有时部分特殊情况会在编写时才会想到，它对于测试策略的要求较高。

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？

减少重复劳动，使开发更加简单

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？

比较适应，这个任务的逻辑比较简单，所以对于这个场景还容易适应。但相比与P1，P3的的难度要高很多

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？

这些工作可以方便程序员快速了解ADT的作用和需要注意的地方，方便之后维护和拓展。

1. 关于本实验的工作量、难度、deadline。

较难，太早

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

希望能延长课时，太紧张了