

**2021年春季学期  
计算学部《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 袁野 |
| 学号 | 1190200122 |
| 班号 | 1903001 |
| 电子邮件 | youngsc30@qq.com |
| 手机号码 | 15831338797 |

**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(abstractionfunction)
* 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

## 实验环境配置

由于本次环境更换为IDEA，因此无需额外配置。

## GitHub Lab2仓库的URL地址

<https://github.com/ComputerScienceHIT/HIT-Lab2-1190200122>

# 实验过程

## Poetic Walks

本实验的目的是练习设计、测试以及编写ADT，而且ADT的规约已经给出。提供了ADT的spec（接口），要求我们写出测试并以两种不同的rep（用边存图以及用点存图）来实现该spec。然后利用构建完成的ADT实现一个根据语料库写诗的程序。

### Get the code and prepare Git repository

从实验指导书所给的仓库中通过git clone指令将代码下载。

通过git clone指令将自己需要提交作业的仓库克隆到本地从而实现链接。

打开IDEA配置工程文件，并将下载好的代码按照指定的目录结构存放。

导入Junit。

### Problem 1: Test Graph <String>

根据Graph的规约，我们需要对于Graph中的每个公开方法编写测试样例。需要我们对方法进行等价类划分，撰写注释文档，然后再编写测试用例。Graph#add(Object)：

* 需要添加的点存在。
* 需要添加的点不存在。

Graph#set(Object, Object, int)：

* 边不存在，起点存在，终点存在，边权为0。
* 边不存在，起点存在，终点存在，边权为正数。
* 边存在，起点存在，终点存在，边权为0。
* 边存在，起点存在，终点存在，边权为正数。
* 边不存在，起点存在，终点不存在，边权为0。
* 边不存在，起点存在，终点不存在，边权为正数。
* 边不存在，起点不存在，终点存在，边权为0。
* 边不存在，起点不存在，终点存在，边权为正数。
* 边不存在，起点不存在，终点不存在，边权为0。
* 边不存在，起点不存在，终点不存在，边权为正数。

Graph#remove(Object)：

* 需要被删除的点存在。
* 需要被删除的点不存在。

Graph#vertices()：

* 图为空图。
* 图中有点。

Graph#sources(Object)：

* 点不存在。
* 点存在且没有边指向它。
* 点存在且有若干条边指向它。

Graph#targets(Object)：

* 点不存在。
* 点存在且它没有指出任何边。
* 点存在且它指出若干条边。

### Problem 2: Implement Graph <String>

#### Implement ConcreteEdgesGraph

##### Edge

首先需要实现一个immutable辅助类Edge，其包含三个field：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数。

AF（source，target，weight）表示一条由source到target权值为weight的边。

采用 private final 的 rep，L 类型为 immutable，采用防御式拷贝。

设计的方法如下：

Creator：

* Edge(L source, L target, int weight)：创建从 source 到 target 边权为 weight 的边

Producer：

* Edge<L> modify(int w):修改边权并创建一个新边

Observer:

* L getSource()：获取起点
* L getTarget()：获取终点
* int getWeight()： 获取边权
* String toString()：获取边的字符串表示

##### ConcreteEdgesGraph

基于给定的两种 field，我们用 Set<L> vertices 来储存点集，用List<Edge<L>> edges 来储存边集。

AF（vertices，edges）表示一张点集为vertices，边集为edges的图。

用 private final 申请变量，L和Edge均为immutable，当返回Map等对象时采用防御式拷贝。

实现方式如下：

* boolean add(L vertex)：向点集中添加点，如果之前已经添加过，那么返回false，否则将其加入并返回true。
* int set(L source, L target, int weight)：遍历所有的edges，寻找是否存在从source到target的边，若存在则记录当前边权，否则记录0，然后根据weight是否为0来决定生成新边或者删除旧边。如果weight为正数则要将source和target中不存在的点添加到点集。
* boolean remove(L vertex)：在vertices中删除vertex，同时遍历 edges，把存在vertex的边全部删除，同时处理返回值（存在true，不存在false）。
* Set<L> vertices()：直接返回属性vertices， 注意采用return new 的防御式返回值以防止表示泄露。
* Map<L, Integer> sources(L target)：遍历edge，将所有终点为 target 的边添加到 map 中返回。
* Map<L, Integer> targets(L source)：遍历edge，将所有起点为 source 的边添加到 map 中返回。
* String toString()：输出"vertices:" "edges:"等标识信息同时调用 vertices 和 edges的toString方法直接输出。

#### Implement ConcreteVerticesGraph

##### Vertex：

由于要求我们设计 mutable 的 Vertex 类且ConcreteVerticesGraph中只有一个 Vertex 的 List，所以想到邻接链表类似的思路。Vertex 类中储存顶点信息以及该顶点的出边（以该顶点为起点的边）的信息。其包含三个field：点对应的属性label，起始边集source和终点边集target。用 private final 申请变量，L为immutable，当返回Map等对象时采用防御式拷贝。

Creator：

* Vertex(L source)：创建值为source的点。

Producer：

* int setSourceEdge(L source, int weight)：修改、新建、或删除一条由source指向当前边的点，权值为weight的边并返回旧的权值。
* int setTargetEdge(L target, int weight)：修改、新建、或删除一条由当前边指向target的点，权值为weight的边并返回旧的权值。

Observer:

* L getLabel()：获取点值
* L getTargets()：获取其指出的点集
* L getSources()：获取指向其的点集
* boolean isTarget(L target)：获取target是否为其指出点集中的一点。
* boolean isSource(L source)：获取source是否为其指入点集中的一点。
* int getTargetWeight ()： 获取其到某一个指出点的边权
* int getsourceWeight ()： 获取某一个指入点到其的边权
* String toString()：获取边的字符串表示

##### ConcreteVerticesGraph：

跟据 Vertex<L>的类似邻接链表的属性来构建我们的 graph。

AF（vertices）= vertices 中各个 Vertex<L>的 source（起点）表示图的顶点。

用 private final 申请变量，L 为 immutable，当返回 Map 等对象时采用防御式拷贝。

实现方式如下：

* boolean add(L vertex)：向点集中添加点，如果之前已经添加过，那么返回false，否则将其加入并返回true。
* int set(L source, L target, int weight)：如果权值为正，向sources的指出点集添加target，向target的指入点集中添加source。否则对应删除。如果weight为正数则要将source和target中不存在的点添加到点集。
* boolean remove(L vertex)：遍历vertices找到该点并删除，随后删除所有点的指入点集以及指出点集中的该点。
* Set<L> vertices()：直接返回属性vertices， 注意采用return new 的防御式返回值以防止表示泄露。
* Map<L, Integer> sources(L target)：遍历vertex，寻找target并返回其指入点集的备份。
* Map<L, Integer> targets(L source)：遍历vertex，寻找source并返回其指出点集的备份。
* String toString()：将所有点的toString整合输出。

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

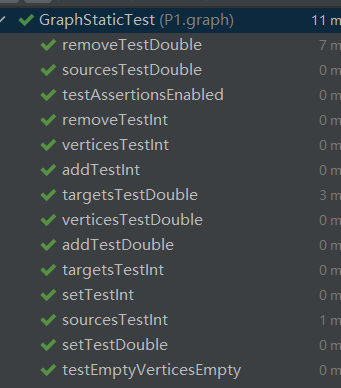
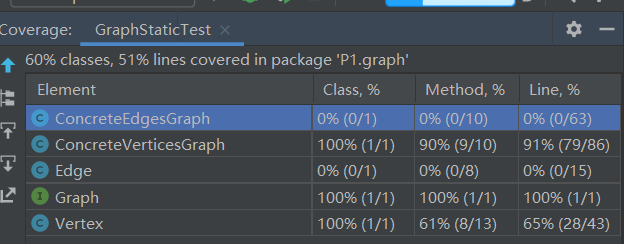
#### Make the implementations generic

在处理过String类型后，因为实现体不依赖 String的方法，只需要将所有的 String替换为L即可。同时写一组int和double的测试。

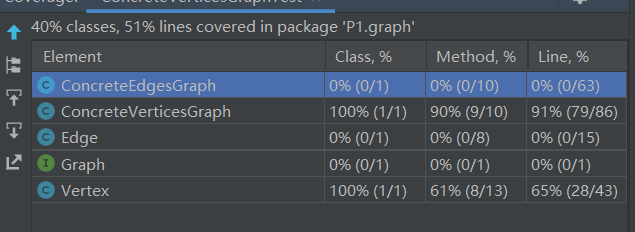
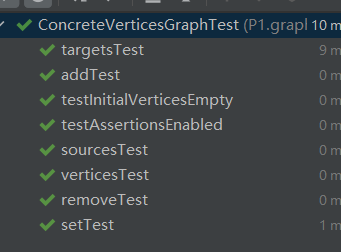
#### Implement Graph.empty()

选择相对时间效率较高的 ConcretVerticesGraph，直接调用 constructor即可下面给出 API 的所有测试用例的结果以及代码覆盖度：

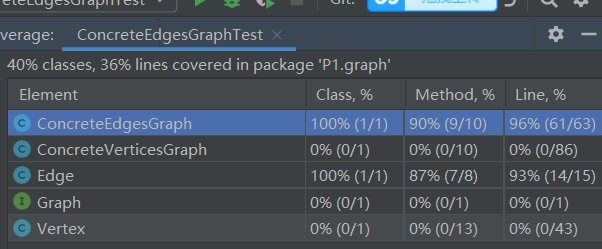
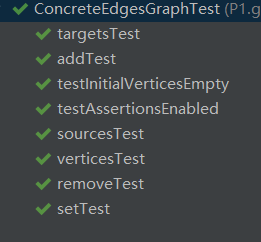
GraphStaticTest：

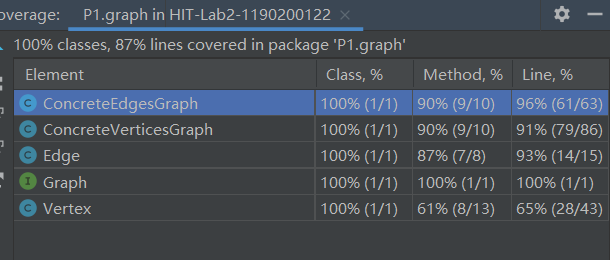
ConcreteVerticesGraphTest：



ConcreteEdgesGraphTest:



整体：



### Problem 4: Poetic walks

#### Implement GraphPoet

实现的GraphPoet包含两个field：一个Graph<String>类型的graph和一个List来存储原本诗中的各个词。

AF：

graph中的点表示文本中出现的单词，graph中的边表示两点邻接的次数

RI：

如果图中的点数超过2个，则只能有一个顶点的出边权值和-入边权值和=1，只能有一个点的出边权值和-入边权值和=-1。

Safety from rep exposure:

采用private final 的rep并且没有任何方法暴露内部表示

方法实现：

GraphPoet():

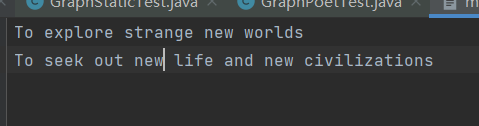
依次读取每相邻的两个单词a，b ，用set方法添加a到b边权为1的，边，若边已经存在，利用set将其权值+1

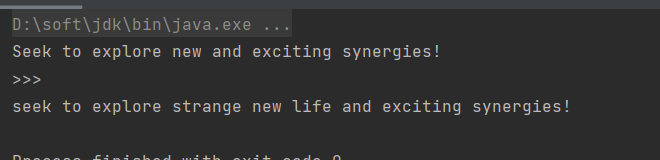
poem():

依次读取相邻两个单词a，b，调用targets方法寻找出a的每一个出边，

这些边指向的顶点都有可能成为中间的bridge，判断每个bridge是否有到b的边，选择其中边权最大的作为最后的bridge添加到a，b之间，最后返回对应的字符串。

#### Graph poetry slam





### Before you’re done

src

P1

graph

ConcreteEdgesGraph.java

ConcreteVerticesGraph.java

Graph.java

poet

GraphPoet.java

Main.java

main.txt

mugar-omni-theater.txt

test

P1

graph

ConcreteEdgesGraphTest.java

ConcreteVerticesGraphTest.java

GraphInstanceTest.java

GraphStatic.java

poet

GraphPoetTest.java

test1.txt

test2.txt

test3.txt

lib

hamcrest-core-1.3.jar

junit-4.13.2.jar

## Re-implement the Social Network in Lab1

利用我们已经设计好的ADT重写一个社交网络的类并进行测试

### FriendshipGraph类

FriendshipGraph():

创建空图即可

addVertex():

调用api的graph.add，如果添加失败（返回值为负）抛出异常终止程序

addEdge（）：

调用graph.vertices判断点是否在图中，随后调用set方法加边

getDistance（）:

依然采用广度优先搜索的方式进行处理，利用graph.targets可以方便找到出边

### Person类

Person类相对于实验1需要强化为immutable类型，也就是需要额外给出equals方法和hashCode方法，下给出这两个方法的实现。

equals():

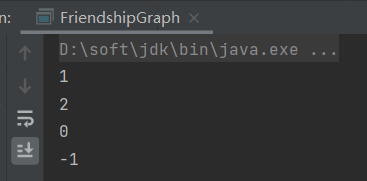
判断对象的String是否equals。

hashCode():

直接利用Objets.hash即可。

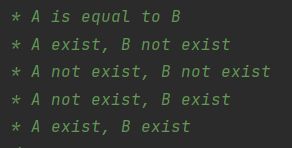
### 客户端main()

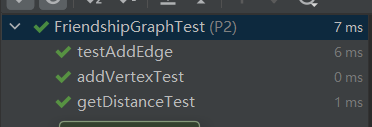
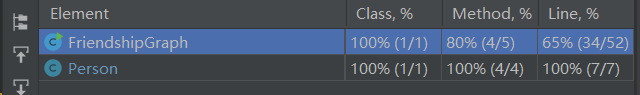
依然采用上次实验提供的main()



结果正确

### 测试用例



### 提交至Git仓库

git add . 将所有修改的文件添加到缓存区

git commit -m “update”

git push

src

P1 …

P2

FriendshipGraph.java

Person.java

test

P1 …

P2

FriendshipGraphTest.java

PersonTest.java

lib …

# 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 |
| 2021.6.1 | 16.00-19.00 | P1 test | 按时完成 |
| 2021.6.1 | 19.00-22.00 | P1 concreteEdge | 延期30min |
| 2021.6.3 | 10:00-12:00 | P1 concreteVertex | 提前15min |
| 2021.6.3 | 16.00-15.00 | P1 结束 | 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 |
| 一开始自己的checkRep()时间代价太高不知如何处理 | 听老师讲解，得知checkRep只在测试时帮助我们尽早发现问题，交给客户时需要注释掉。 |
|  |  |
|  |  |

# 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. 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？
6. 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？
7. 关于本实验的工作量、难度、deadline。
8. 《软件构造》课程进展到目前，你对该课程有何体会和建议？