

**2022年春季学期  
计算学部《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 郑旭然 |
| 学号 | 120L020719 |
| 班号 | 2003005 |
| 电子邮件 | zxrshawn@icloud.com |
| 手机号码 | 16645013218 |

**目录**

[1 实验目标概述 1](#_Toc72247573)

[2 实验环境配置 1](#_Toc72247574)

[3 实验过程 1](#_Toc72247575)

[3.1 Poetic Walks 1](#_Toc72247576)

[3.1.1 Get the code and prepare Git repository 2](#_Toc72247577)

[3.1.2 Problem 1: Test Graph <String> 2](#_Toc72247578)

[3.1.3 Problem 2: Implement Graph <String> 2](#_Toc72247579)

[3.1.3.1 Implement ConcreteEdgesGraph 3](#_Toc72247580)

[3.1.3.2 Implement ConcreteVerticesGraph 3](#_Toc72247581)

[3.1.4 Problem 3: Implement generic Graph<L> 4](#_Toc72247582)

[3.1.4.1 Make the implementations generic 5](#_Toc72247583)

[3.1.4.2 Implement Graph.empty() 5](#_Toc72247584)

[3.1.5 Problem 4: Poetic walks 5](#_Toc72247585)

[3.1.5.1 Test GraphPoet 5](#_Toc72247586)

[3.1.5.2 Implement GraphPoet 5](#_Toc72247587)

[3.1.5.3 Graph poetry slam 5](#_Toc72247588)

[3.1.6 Before you’re done 6](#_Toc72247589)

[3.2 Re-implement the Social Network in Lab1 7](#_Toc72247590)

[3.2.1 FriendshipGraph类 8](#_Toc72247591)

[3.2.2 Person类 8](#_Toc72247592)

[3.2.3 客户端main() 8](#_Toc72247593)

[3.2.4 测试用例 10](#_Toc72247594)

[3.2.5 提交至Git仓库 11](#_Toc72247595)

[4 实验进度记录 12](#_Toc72247596)

[5 实验过程中遇到的困难与解决途径 12](#_Toc72247597)

[6 实验过程中收获的经验、教训、感想 12](#_Toc72247598)

[6.1 实验过程中收获的经验和教训 12](#_Toc72247599)

[6.2 针对以下方面的感受 13](#_Toc72247600)

# 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：

(1)针对给定的应用问题，从问题描述中识别所需的 ADT；

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

(3)根据 ADT 的规约设计测试用例；

(4)ADT 的泛型化；

(5)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）

(6)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）；

(7)测试 ADT 的实现并评估测试的覆盖度；

(8)使用 ADT 及其实现，为应用问题开发程序；

(9)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

覆盖度测试为IDEA自带。

我的GitHub Lab2仓库的URL地址：

<https://github.com/ComputerScienceHIT/HIT-Lab2-120L020719>

# 实验过程

## Poetic Walks

（覆盖度测试在Poetic Walks最后给出！）

任务要求实现两种不同形式的图，给出了一个图接口来规定图所包含的各种方法，两种不同的实现需要继承这个接口，实现给定方法，同时需要满足泛型要求。在完成了两种实现后，任选一种实现形式完成自动扩展诗歌的任务。

1.完善Graph接口类，并运用泛型的思想，将String拓展为泛型L类；

2.实现Graph类的方法：add、set、remove、vertices、sources、targets；

3.用Graph类实现GraphPoet类，扩展诗歌。

### Get the code and prepare Git repository

由于已经提前进行了这个过程所以此处再输入命令进行示意。

首先在本地创建文件夹后git init。然后创建IDEA工程。

文本

描述已自动生成

之后从GitHub获取该任务的代码：



然后将P1与P4都放入src文件夹中。

然后我将报告事先放入doc文件夹，并初次提交至github仓库。

文本

描述已自动生成

### Problem 1: Test Graph <String>

图形用户界面, 文本

中度可信度描述已自动生成

### Problem 2: Implement Graph <String>

Problem 2是要求我们实现3.1.3.1和3.1.3.2里的两个图的类，并能通过同一个实例的测试。具体分析如下。

#### Implement ConcreteEdgesGraph

包括几个部分：Edge类、ConcreteEdgesGraph类，最后进行Junit测试。

Edge类中source为起始点，访问是通过L source()；target为终点，访问是通过L target()；weight是边权，访问是通过int weight()。三者都是Immutable的。

Edge下的方法还有如下：toString()，返回类型为String，输出起始、终止和边权；sameEdge()，返回类型为boolean，可以判断两条边是否相同（起点终点是否相同）；checkRep()，边权为正整数，起点终点不同；Edge()，参数有source，

target，weight，初始化一条边的三个成员变量。

ConcreteEdgesGraph类中是以Edge为基础重写Graph<L>，存储点和边，增加边或删除点等也需要更改集合。

Set<L> vertices存储点集，且点的类型为L；List<Edge<L>> edges存储边集，且边的类型为Edge；ConcreteEdgesGraph()；checkRep()，这里检查所有点和边均不能为空。

add()向点集vertices中添加点，如果已经有点会抛出异常；set()是向edges中添加边，判断是否存在相同边，有边则更新，边权不为正则抛出异常；remove()是删除一个点，删除该点涉及到的边，不存在该点则抛出异常；vertices()是用来返回vertices()；sources()在edges中遍历，返回以该点为终点的集合；targets()在edges中遍历，返回以该点为起点的集合；toString()输出图的边数和点数，可以调用size()。

最后进行Junit测试。

文本

描述已自动生成

#### Implement ConcreteVerticesGraph

包括几个部分：Vertex类、ConcreteVerticesGraph类，最后进行Junit测试。

Vertex是点的集合的抽象类，包含点名称、该点的入边和出边。Vertex将访问以上特征，以及增删边。以下将说明其中的方法。

Vertex类中，ThisVertex()存储点的名称，也可以说是标号；inEdges()是存储入边的，key是点标号，值是边权；outEdges是存储出边的，key是点标号，值是边权；Vertex()用于初始化标识；checkRep()中检查checkRep()的inEdges和outEdges边权大于0；source()返回inEdges；targets()返回outEdges；setEdge()添加边，若边已存在则删除后重新增加，并返回之前的边权；removeEdge()删除边，若边已存在则删除，返回之前的边权，边不存在则返回0；

ConcreteVerticesGraph以点为基础，通过名称进行区分，set和remove都依靠Vertex类中的增删操作，sources和targets也依靠Vertex中的方法。List<Vertex<L>> vertices是点集合，通过vertices()访问；ConcreteVerticesGraph()构造；checkRep()检查点的名称不能为空，所有点的入边出边不能为空且边权大于0；add()中会检查名称是否唯一，如果合法的话添加进vertices；set()在集合中找到点并添加边，边权为负或起点终点相同会抛出异常；remove()删除所有该点涉及的边；sources()调用Vertex()类中的sources()；targets()调用Vertex()类中的targets()。

最后进行Junit测试。

电脑屏幕截图

描述已自动生成

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

#### Make the implementations generic

在程序中重构并更改所有匹配项，并且单独注意一下toString()，即可实现泛型。

#### Implement Graph.empty()

这里我们要使Graph.empty()能返回一个新的empty instance。

文本

描述已自动生成

### Problem 4: Poetic walks

在Problem 4中，我们给出一个语料库corpus，根据语料库中的文本生成一个图，内含words，接下来输入语句，在图中找词与词之间的关系，对句子进行补充。

在语料库中相邻单词之间有一条有向边，相邻单词出现多少次，那么图中两个单词的节点之间就会有多少权值。在补充句子时，考察相邻单词之间是否有一个桥，使得能通过这个桥连接之前相邻的单词，那么就在单词之间补全这一单词，补全的优先级是权值越大越优先补充。

#### Test GraphPoet

我们需要划分等价类：两个单词之间有0/1/>1个单词连接。

#### Implement GraphPoet

表示不变量和检查不变量中，不变量是说所有的点都不应为空。

输入单词后，字符串首先分割再转为小写。建立图的过程中加点，加边时判断边是否存在，通过set()来判断之前是否存在一条边。

如果有一个单词不在图中，则直接加入不在的单词即可。桥的长度只能为2，则分别求两个单词的起点终点，求交集。若交集为空则没有桥；若不空则找最短的桥。

#### Graph poetry slam

txt文件：This is a test of the Mugar Omni Theater sound system.

图形用户界面, 文本

描述已自动生成

Junit测试：（此处用了绝对路径，因为使用相对路径一直有错误，且IDEA给定的路径也无法跑通。与同学讨论及上网查资料也没能解决。此处若还是跑不通烦请换成本机绝对路径）

图形用户界面, 文本, 应用程序

描述已自动生成

### Before you’re done

首先给出覆盖度测试：

图形用户界面, 文本, 应用程序

描述已自动生成

图形用户界面

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

树状结构示意图：

图示

低可信度描述已自动生成

## Re-implement the Social Network in Lab1

利用我们在刚刚写的ADT，把FriendshipGraph重新实现一遍，节点仍然是Person类型，所以泛型L一律为Person。对于现成的FriendshipGraph中的方法，要用刚刚Grapgh中的ADT中的方法来实现它们。

### FriendshipGraph类

Graph<Person> graph：调用Graph的静态方法.empty()生成一个空图。

boolean addVertex()：调用graph.add()添加点。

int addEdge()：调用graph.set()，添加双向边且权值为1，并记录之前边权值（如果有）。

int getDistance()：首先判断起点终点是否相等。再新建Map<Person, Integer> dis表示从起点开始到Person的距离，Map<Person, Boolean> vis指Person之前是否被经过。将Map初始化后，把起点标记为已经到达。最后是广度优先搜索，直到找到终点。

### Person类

该类是将每个人映射到Person对象并存上名字。为了safety of rep exposure，String Name设置为private且immutable的，另外在构造函数中将Name初始化。

### 客户端main()

1. **package** P2;
3. **import** **static** org.junit.Assert.\*;
5. **import** org.junit.Test;
7. **public** **class** FriendshipGraphTest {
9. /\*\*
10. \* Basic Network Test
11. \*/
12. @Test
13. **public** **void** Test1() {
14. **final** P2.FriendshipGraph graph = **new** P2.FriendshipGraph();
16. **final** P2.Person rachel = **new** P2.Person("Rachel");
17. **final** P2.Person ross = **new** P2.Person("Ross");
18. **final** P2.Person ben = **new** P2.Person("Ben");
19. **final** P2.Person kramer = **new** P2.Person("Kramer");
21. assertEquals(**true**, graph.addVertex(rachel));
22. assertEquals(**true**, graph.addVertex(ross));
23. assertEquals(**true**, graph.addVertex(ben));
24. assertEquals(**true**, graph.addVertex(kramer));
26. assertEquals(0, graph.addEdge(rachel, ross));
27. assertEquals(1, graph.addEdge(ross, rachel));
28. assertEquals(0, graph.addEdge(ross, ben));
29. assertEquals(1, graph.addEdge(ben, ross));
31. assertEquals(1, graph.getDistance(rachel, ross));
32. assertEquals(2, graph.getDistance(rachel, ben));
33. assertEquals(0, graph.getDistance(rachel, rachel));
34. assertEquals(-1, graph.getDistance(rachel, kramer));
35. }
37. /\*\*
38. \* Further Test
39. \*/
40. @Test
41. **public** **void** Test2() {
42. **final** P2.FriendshipGraph graph = **new** P2.FriendshipGraph();
44. **final** P2.Person a = **new** P2.Person("A");
45. **final** P2.Person b = **new** P2.Person("B");
46. **final** P2.Person c = **new** P2.Person("C");
47. **final** P2.Person d = **new** P2.Person("D");
48. **final** P2.Person e = **new** P2.Person("E");
49. **final** P2.Person f = **new** P2.Person("F");
50. **final** P2.Person g = **new** P2.Person("G");
51. **final** P2.Person h = **new** P2.Person("H");
52. **final** P2.Person i = **new** P2.Person("I");
53. **final** P2.Person j = **new** P2.Person("J");
55. assertEquals(**true**, graph.addVertex(a));
56. assertEquals(**true**, graph.addVertex(b));
57. assertEquals(**true**, graph.addVertex(c));
58. assertEquals(**true**, graph.addVertex(d));
59. assertEquals(**true**, graph.addVertex(e));
60. assertEquals(**true**, graph.addVertex(f));
61. assertEquals(**true**, graph.addVertex(g));
62. assertEquals(**true**, graph.addVertex(h));
63. assertEquals(**true**, graph.addVertex(i));
64. assertEquals(**true**, graph.addVertex(j));
66. assertEquals(0, graph.addEdge(a, b));
67. assertEquals(0, graph.addEdge(a, d));
68. assertEquals(0, graph.addEdge(b, d));
69. assertEquals(0, graph.addEdge(c, d));
70. assertEquals(0, graph.addEdge(d, e));
71. assertEquals(0, graph.addEdge(c, f));
72. assertEquals(0, graph.addEdge(e, g));
73. assertEquals(0, graph.addEdge(f, g));
74. assertEquals(0, graph.addEdge(h, i));
75. assertEquals(0, graph.addEdge(i, j));
77. assertEquals(2, graph.getDistance(a, e));
78. assertEquals(1, graph.getDistance(a, d));
79. assertEquals(3, graph.getDistance(a, g));
80. assertEquals(3, graph.getDistance(b, f));
81. assertEquals(2, graph.getDistance(d, f));
82. assertEquals(2, graph.getDistance(h, j));
83. assertEquals(0, graph.getDistance(i, i));
84. assertEquals(-1, graph.getDistance(d, j));
85. assertEquals(-1, graph.getDistance(c, i));
86. assertEquals(-1, graph.getDistance(f, h));
87. }
88. }

### 测试用例

已在代码中体现。首先是对于四个人来说，Rachel和Ross有关系，Ross和Ben有关系；其他关系均不存在。

更复杂的：测试用例中分成两个互不连通的子图，一个3节点，一个7节点。

以下是测试结果：

图形用户界面, 文本

描述已自动生成

覆盖度测试：

图形用户界面, 文本, 应用程序

描述已自动生成

### 提交至Git仓库

文本

描述已自动生成

文本

描述已自动生成

树状结构示意图：

图片包含 图示

描述已自动生成

# 实验进度记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 5.13 | 13:00-14:00 | 撰写实验目标概述与实验环境配置 | 按计划完成 |
| 5.14 | 9:30-12:00 | 完成ConcreteEdgesGraph及相关报告 | 按计划完成 |
| 5.18 | 10:00-11:45 | 完成ConcreteVerticesGraph及相关报告 | 按计划完成 |
| 5.22 | 13:00-16:30 | 完成剩余的Problem3-4 | 按计划完成 |
| 5.24 | 20:00-22:20 | 撰写了Poetic Walks的相应报告 | 按计划完成 |
| 5.28 |  | 最终完善Poetic Walks并push | 按计划完成 |
| 5.29 | 8:00-11:00 | 完成P2 | 按计划完成 |
| 5.29 | 23:00-23:30 | 完成“6 实验过程中收获的经验、教训、感想”部分 | 按计划完成 |
| 5.29 | 截止前 | 进行了检查。检查后push | 按计划完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 在开始做实验时没能读懂题意，对着Test Graph <String>做了很久，一直跑不通 | 线下询问老师解决。发现是理解错题意了，应该先完成相应的implementation等再测试。 |
| AF,RI,Safety from rep exposure概念不清晰 | 回看课程PPT，上网查阅资料 |
| 给出ADT规约后再编写测试用例；给ADT撰写spec、AF、RI、注意rep exposure | 其实一开始有些不习惯，不过这样的习惯是好的，万事开头难，应该多多练习编程基本功。 |

# 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

1. 为ADT撰写mutability/immutability说明、AF、RI、safety from rep exposure、ADT中各个方法的spec、为test写testing strategy是十分重要的。
2. 总是因为一些小错误耽误很长时间，因此要加强代码的基础功底。
3. 在开始一个项目或工程时要事先考虑好spec；要认真写每个方法的spec，其实一开始完成这个实验的时候我没体会出spec等的重要性，事实上在后面吃了很大的亏。spec还要写好方法细节之处的功能和注释，“可以使程序与客户端之间达成一致，明确双方的责任，并且对正确实现进行定义”。
4. 完成一个工程需要从ADT开始构思和设计；测试完ADT的正确性后再检查整个工程是否完善。

## 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？

* 面向ADT编程复用性会更强，可能在前期的设计与实现过程会比较繁琐，但在以后使用的时候会更灵活，更加模块化。面向应用场景编程的复用性相比之下不会很高。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

* 泛型使用起来更方便，不用为每种数据类型都设计一个类，可以兼容指定类型。

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？

* 优势是可以在写测试时就将AF和RI考虑好，进一步完善spec；但由于刚开始接触并不是很适应，在整个编程过后还是会补充测试用例，不过我相信这种方法可以在以后给我带来更多益处。

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？

* 代码量减少了，复用性增强，效益增大，使用起来更有规范更加方便，也会更加省时间。

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？

* 能够是程序正确性和鲁棒性得到保证，更大可能避免了不该发生的错误；是的，因为这其实是很有意义的，对于后面的测试有好处，虽然前期适应可能需要花费功夫。

1. 关于本实验的工作量、难度、deadline。

* 工作量与难度均适中；deadline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？

* 我认为这门课非常好，为我打开了OOP语言世界的大门。建议可以将实验拆散成几个小实验也许会更好，也能在配置Project Structure时省下不少功夫。