哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2021 年春季学期
计算学部《软件构造》课程

Lab 2 实验报告

姓名	
学号	
班号	
电子邮件	
手机号码	

目录

目录

1 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。

2 实验环境配置

本次实验使用的 IDE 是 IDEA，环境为 JDK8，Junit4，由于 IDEA 中 Junit 自带测试代码覆盖度的功能，因此没有使用 Eclemma。
https://github.com/ComputerScienceHIT/HIT-Lab2-1190300321/tree/master

3 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!）。

3.1 Poetic Walks

该任务主要分为两部分，第一部分是使用两种实现方法实现一个图，第二部分就是使用设计好的图来实现一个单词插入的功能，主要就是 ADT 和 OOP 技术的使用。

3.1.1 Get the code and prepare Git repository

从网站获取代码之后，将代码修改成需要的结构之后使用 git 管理，并上传代码。

3.1.2 Problem 1: Test Graph <String>

这一部分主要是实现了对于 graph 的测试，对于每一个测试方法都使用划分等价类的方法，例如，对于 set 方法的测试划分等价类：按照需要操作的点划分：两点都存在，两点都不存在，两点有一个存在；按照 weight 划分：weight>0,weight<0,weight=0；按照边存在情况划分：存在，不存在。其余部分关键方法测试方法如下截图：

```
/* Testing strategy
 * 测试空图、非空图
 */
@Test
public void testConcreteEdgesGraphtoString() {
```

```
/* Testing strategy
 * 按照加入的点划分：点已经存在，点不存在
 */
@Test
public void testConcreteEdgesGraphadd() {
```

```
/* Testing strategy
 * 按照需要删除的点划分：点存在，点不存在
 */
@Test
public void testConcreteEdgesGraphremove() {
```

3.1.3 Problem 2: Implement Graph <String>

这一部分主要使用两种方式实现一个图，以下分别叙述。

3.1.3.1 Implement ConcreteEdgesGraph

这一部分主要以边来实现一个图，首先需要设计一个 Edge 类，这个类主要的 Rep 为边的出发节点、终止节点，由于这是一个 immutable 的类，因此不存在 mutator，主要实现的方法都是获取该边的信息的方法，例如 getSource 等。

实现一个 Edge 之后就需要使用它来实现一个 Graph，graph 的 Rep 如下截图：

```
private final Set<L> vertices = new HashSet<>();
private final List<Edge<L>> edges = new ArrayList<>();
```

由于这个类是接口 Graph 的一个实现，因此方法的 Spec 都已经定义好了，主要的工作就是完成每一个方法。例如其中比较复杂的是 Set 方法，主要功能是增加、修改、删除边的权重，实现的时候采用的是逐个匹配，如果出发节点和终止节点都匹配的话修改其权重，其余情况或是删除边，或是增加边，由于代码过长，不在此展示代码。

3.1.3.2 Implement ConcreteVerticesGraph

这一部分实现的时候和上述用边实现的基本一致。首选需要实现一个 Vertex 类，这个类是一个 mutable 的类，因此其存在 mutator，在设计的时候，主要的 mutator 是改变指向当前节点的节点集合与当前节点指向的节点的集合，而其余的方法主要是返回当前节点的一些信息的。

实现了 Vertex 之后主要的任务就是实现一个 Graph，基本的实现方法与用 Edge 实现的一致。不同之处主要在于在寻找边的时候 Edge 实现的图直接将每一条边的 source 和 target 与目标进行比较，如果相同则找到了需要的边；用顶点方法实现的时候需要遍历每一个节点，在指向当前节点的集合和节点指向的集合里面匹配目标，因此遍历的次数显然较多。但是对于节点的增删较为方便。

3.1.4 Problem 3: Implement generic Graph<L>

3.1.4.1 Make the implementations generic

这一部分主要是将 String 实现的类改成泛型 L，首先可以先把所有的 String 都改成 L，会发现有一定数量的报错，这些报错都是因为在之前实现的时候用了 String 类特有的一些方法，因此导致了错误。因此需要将这些方法修改成通用的方法，就可以消除报错。

3.1.4.2 Implement Graph.empty()

可以发现这是一个静态工厂方法，在实现的时候只需要将其返回一种图实现方式的一个示例即可，具体代码截图如下：

```
public static <L> Graph<L> empty() {
    return new ConcreteEdgesGraph<>();
}
```

3.1.5 Problem 4: Poetic walks

这一部分需要实现的就是使用上述实现的图的两种方式中的一种采用 OOP 技术实现一个在文本的特定位置插入特定字符的任务，主要叙述如下：

3.1.5.1 Test GraphPoet

在设计测试样例的时候首先进行等价类划分：从文本结构来分：空文本、单行文本、多行文本，从是否需要从多个单词中选择一个来看：需要、不需要，因此使用五个 txt 文件来存储对应的测试样例，并将结果与预先设定的结果比较，观察是否符合。

3.1.5.2 Implement GraphPoet

这一部分第一部分主要是为了读入语料库的文本的，在处理的时候使用一个 Map 来统计每一个字符对（相邻的两个字符组成一个字符对）的出现次数，同时将读入的语料库作为一个 List 备用。主要代码部分截图如下：

```
public GraphPoet(File corpus) throws IOException {
    BufferedReader readData = new BufferedReader(new FileReader(corpus));
    String rawData="";
    List<String>wordList=new ArrayList<>();
    Map<String, Integer>wordMap=new HashMap<>();
    //读取数据
    while((rawData=readData.readLine())!=null)
    {
        wordList.addAll(Arrays.asList(rawData.split( regex: " ")));
    }
    readData.close();
    //把字符对加入图中

    for(int i=0;i<wordList.size()-1;i++)
    {
        Integer preWeight=0;
        String source=wordList.get(i).toLowerCase().replaceAll( regex: "\\pP", replacement: "");//清除标点符号
        String target=wordList.get(i+1).toLowerCase().replaceAll( regex: "\\pP", replacement: "");
        String key=source+" "+target;//中间加上空格主要是为了防止出现类似于they+east==the+yeast这种问题
        if(wordMap.containsKey(key))
        {
            preWeight=wordMap.get(key);
```

接着是实现功能的主体部分。主要设计思想是建立一个 StringBuilder 来存储最后需要返回的文本。每次循环的时候都加入一个原有的文本，并根据原有文本的当前字符与下一个字符遍历比较在语料库中是否存在一个字符能连接两个字符，如果存在的话加入，如果存在多个的话选择在语料库中出现次数最多的那个字符，最终返回一个 String 即可。需要注意的是除了最后一次插入，每一次插入的时候都需要额外插入一个空格符。由于这一部分代码太长，不在此展示。

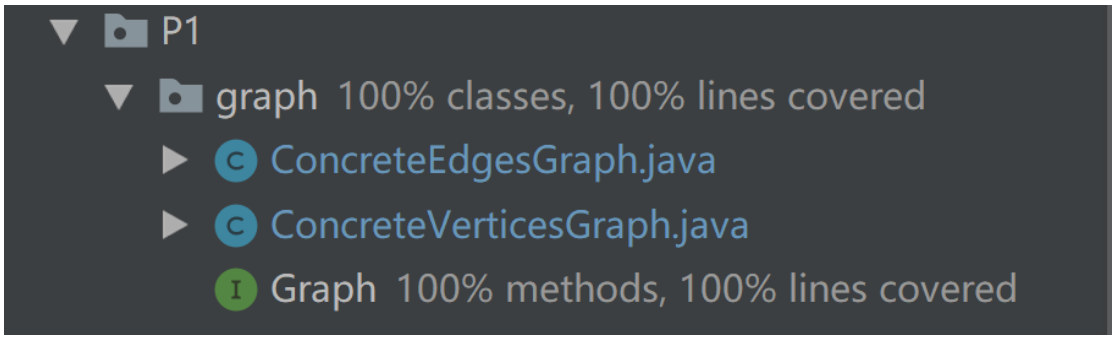
3.1.5.3 Graph poetry slam

这一部分主要是为了测试上述的功能是否完善的。因此在代码中采用了《生如夏花》的文本作为语料库进行测试，测试部分结果如截图所示：

```
life like flowers and I like life .
>>>
life like summer flowers and I like life
let->life weight:1
life->be weight:1
summer->flowerslife weight:1
flowerslife->thin weight:1
thin->and weight:1
and->lightoff weight:1
lightoff->time weight:1
time->and weight:1
and->time weight:1
time->againfrivolous weight:1
```

3.1.6 使用 Eclemma 检查测试的代码覆盖度

注：由于采用的 IDEA 的 Junit 自带检测代码覆盖度的功能，因此没有采用 Eclemma



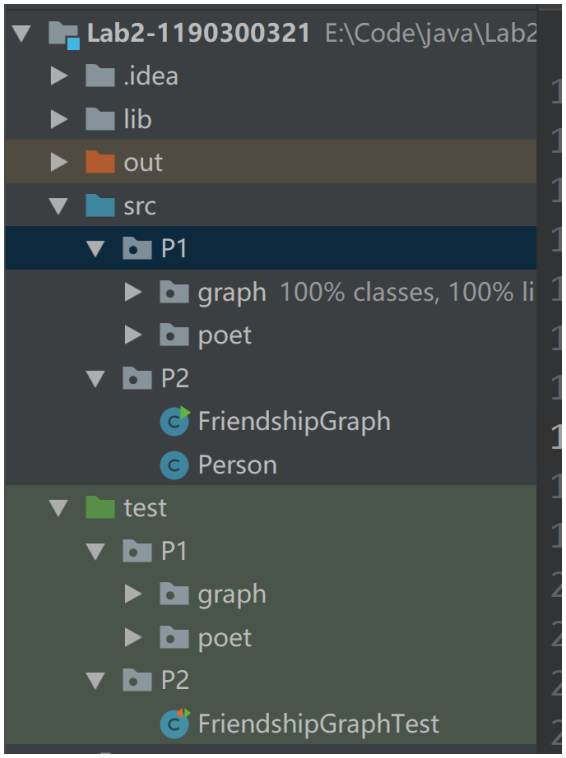
3.1.7 Before you're done

请按照 http://web.mit.edu/6.031/www/sp17/psets/ps2/#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。

使用 git add ,git commit, git push 等命令上传

在这里给出你的项目的目录结构树状示意图。



3.2 Re-implement the Social Network in Lab1

这一部分主要是重新实现 Lab1 中的社会网络部分，可以测试上述 graph 的

设计是否完善。

3.2.1 FriendshipGraph 类

使用上述的以边建立的图来实现这个类，将每一个节点设置为 Person 类型即可。

该类具有两个属性与三个方法。首先两个属性分别是 People 列表与已经存在的人名的集合 nameSet，后一个集合主要是为了处理两个人重名的不合法输入情况。在处理完输入不合法情况之后需要对三个方法进行处理。

其中加入朋友关系的方法只需要使用 graph 的 set 方法即可。

较为复杂的 getDistance 方法使用了 java 中的 Map，并使用 BFS，在 BFS 每一次执行中都比对当前的节点是不是所求节点，如果是直接输出距离，如果不是且该节点未访问过，那么使用 Map 将其与一个当前距离+1 联系起来，直到寻找到需要的那个节点或是寻找不到输出-1。getDistance 部分代码截图如下：

```
public int getDistance(Person a,Person b)
{
    if(a.equals(b))return 0;
    Queue<Person>personQueue=new LinkedList<>();
    Map<Person,Integer>distanceMap=new HashMap<>();
    personQueue.add(a);
    distanceMap.put(a,0);
    while(!personQueue.isEmpty())
    {
        Person topPerson=personQueue.poll();
        Integer distance=distanceMap.get(topPerson);
        Set<Person>personSet=personGraph.targets(topPerson).keySet();
        for(Person p:personSet)
        {
            if(!distanceMap.containsKey(p))
            {
                distanceMap.put(p,distance+1);
                personQueue.add(p);
                if(p.equals(b))
                    return distanceMap.get(p);
            }
        }
    }
    return -1;
FriendshipGraph › getDistance()
```

3.2.2 Person 类

Person 类设计与 Lab1 完全一致，首先 Person 类中有两个属性，分别是名字与朋友列表，分别为 String 与 List。在定义了构造函数的同时，还有三个方法，分别是 addFriend，getName，getFriends，功能分别是增加朋友，获得 Person 的 name 与获得 Person 的 friend 列表。

3.2.3 客户端 main()

使用的是 Lab1 的 pdf 中给出的测试样例，首先定义 Person，接着向生成的 graph 中添加节点、添加关系并测试两个人之间的关系是不是符合我们预先设定

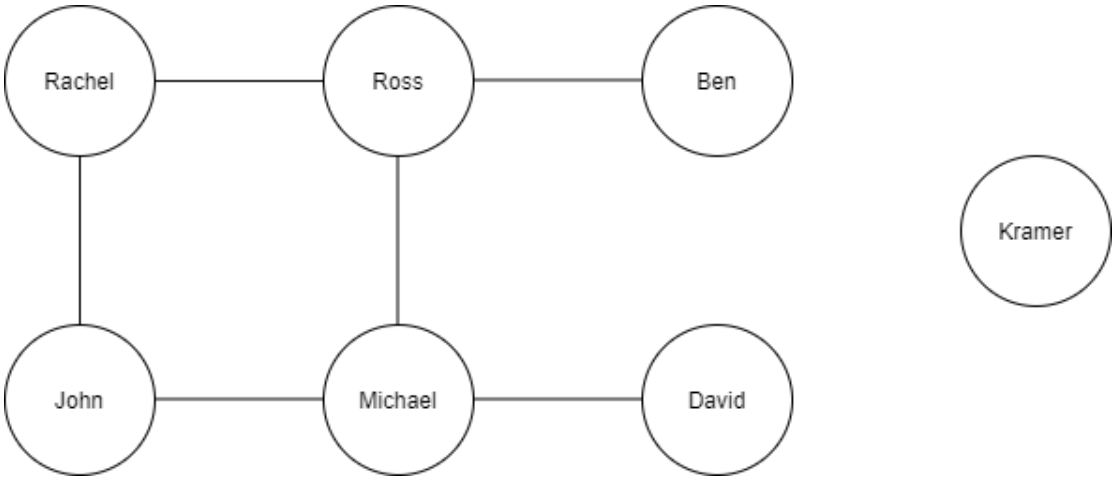
的值。测试结果如下：

```
1
2
0
-1
```

```
System.out.println(graph.getDistance(rachel, ross));
// should print 1
System.out.println(graph.getDistance(rachel, ben));
// should print 2
System.out.println(graph.getDistance(rachel, rachel));
// should print 0
System.out.println(graph.getDistance(rachel, kramer));
// should print -1
```

3.2.4 测试用例

根据等价类划分，可以分为：无朋友、直接相连的朋友、间接相连的关系。设计的测试样例如下：

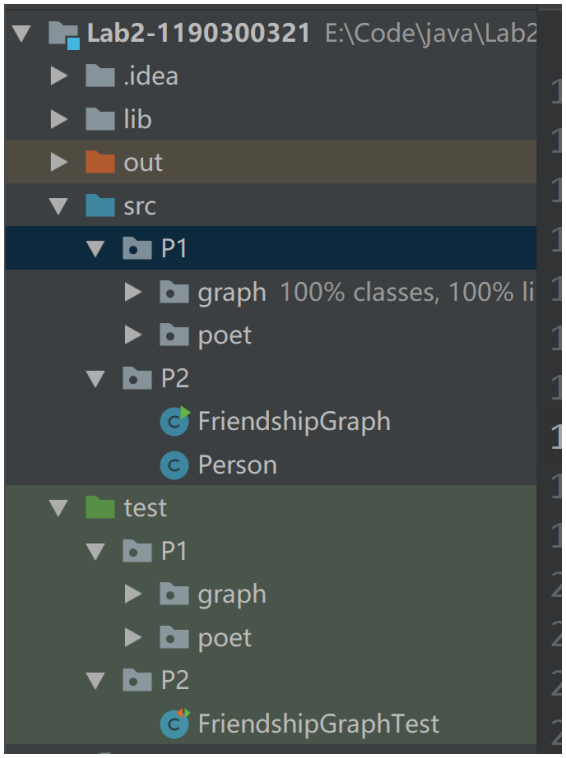


3.2.5 提交至 Git 仓库

如何通过 Git 提交当前版本到 GitHub 上你的 Lab3 仓库。

使用 git add ,git commit, git push 等命令上传

在这里给出你的项目的目录结构树状示意图。



4 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
2021.5.25	13:45-15:30	写完测试样例	未完成
2021.5.29	14:30-17:30	完成测试样例，完成 P2 图的设计	16:45 提前完成
2021.5.30	14:30-17:00	完成 P3 和 P4 测试样例和设计	完成
2021.6.1	13:45-15:30	完成 lab-1 任务重写，完善 P2 测试样例	完成
2021.6.5	18:45-20:45	修改 P2 代码，完成报告	完成

5 实验过程中遇到的困难与解决途径

遇到的难点	解决途径
对于任务不太清晰，同时对于 RI 等概念不熟悉	对于任务问题多读了几遍要求，并着手完成就逐渐了解了；对于概念问题上课之后有了更深的了解。
对于泛型不是很理解	查找资料之后，比对 c++的模板学习，有了更深的了解

6 实验过程中收获的经验、教训、感想

6.1 实验过程中收获的经验和教训

6.2 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 的编程复用性更强，虽然设计过程可能更加繁琐，但是在使用的时候更灵活。

(2) 使用泛型和不使用泛型的编程，对你来说有何差异？
使用泛型设计的时候稍显繁琐，但是使用起来较为方便，不需要为每一种数据类型设计一个类。

(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：对于目标更加清晰，从而编写代码的时候思路更加清晰。正在逐渐适应。

(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？
代码量减少，同时使用的时候也更加方便也显得更加统一。

(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？
正在逐渐适应；自主设计的时候感觉还是有一定难度的，需要在不断练习

中不断熟悉。

(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？

意义：防止出现不安全的代码；愿意

(7) 关于本实验的工作量、难度、deadline。

适中，可以接受

(8) 《软件构造》课程进展到目前，你对该课程有何体会和建议？

软件构造是大学至今第一门真正接触 ADT 和 OOP 设计模式的课程，并在实验课中亲身体会，有了更加深刻的认识，收获很大。