



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

# 2022 年春季学期 计算学部《软件构造》课程

## Lab 2 实验报告

姓名	李世轩
学号	120L022109
班号	2003007
电子邮件	1146887979@qq.com
手机号码	15234117960

## 目录

1 实验目标概述 .....	1
2 实验环境配置 .....	1
3 实验过程 .....	2
3.1 Poetic Walks .....	2
3.1.1 Get the code and prepare Git repository .....	2
3.1.2 Problem 1: Test Graph <String> .....	3
3.1.3 Problem 2: Implement Graph <String> .....	4
3.1.3.1 Implement ConcreteEdgesGraph .....	4
3.1.3.2 Implement ConcreteVerticesGraph .....	7
3.1.4 Problem 3: Implement generic Graph<L> .....	10
3.1.4.1 Make the implementations generic .....	10
3.1.4.2 Implement Graph.empty() .....	12
3.1.5 Problem 4: Poetic walks .....	13
3.1.5.1 Test GraphPoet .....	13
3.1.5.2 Implement GraphPoet .....	14
3.1.5.3 Graph poetry slam .....	15
3.1.6 使用 Eclemma 检查测试的代码覆盖率 .....	15
3.1.7 Before you're done .....	16
3.2 Re-implement the Social Network in Lab1 .....	17
3.2.1 FriendshipGraph 类 .....	17
3.2.2 Person 类 .....	18
3.2.3 客户端 main() .....	18
3.2.4 测试用例 .....	19
3.2.5 提交至 Git 仓库 .....	20
4 实验进度记录 .....	21
5 实验过程中遇到的困难与解决途径 .....	22
6 实验过程中收获的经验、教训、感想 .....	22
6.1 实验过程中收获的经验教训（必答） .....	22
6.2 针对以下方面的感受（必答） .....	22

## 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。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。

## 2 实验环境配置

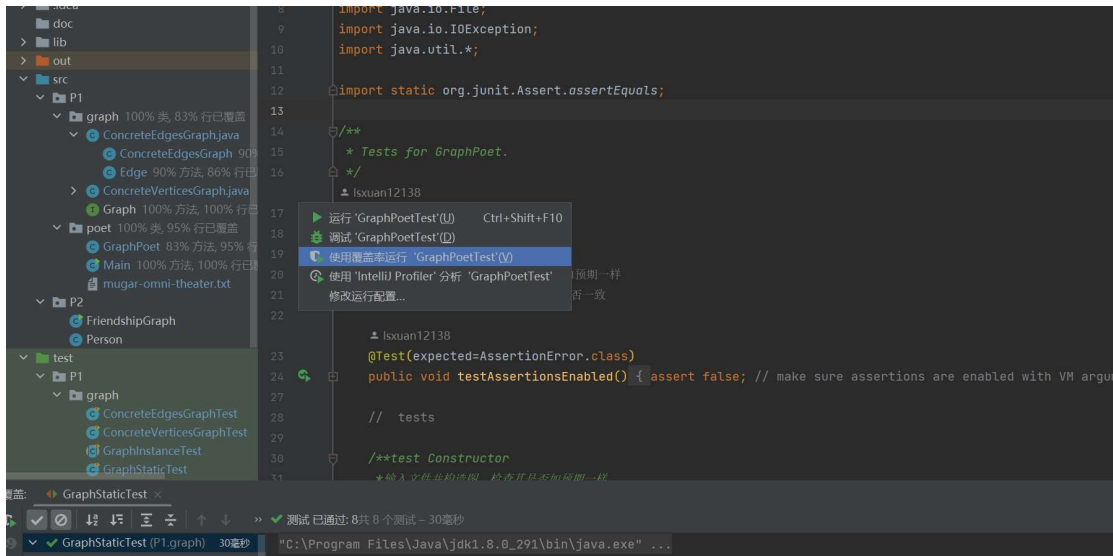
简要陈述你配置本次实验所需环境的过程, 必要时可以给出屏幕截图。

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。

大部分配置在前一次实验中以及配置过, 这里不再赘述。

这里简要说明 idea 中进行代码覆盖度测试的方法

在代码左侧的绿色三角, 也就是代表运行的地方, 右键即可打开菜单, 如图点击使用覆盖率运行, 即可对该方法覆盖到的类, 方法, 代码行等进行查看



在这里给出你的 GitHub Lab2 仓库的 URL 地址 (Lab2-学号)。

<https://github.com/ComputerScienceHIT/HIT-Lab2-120L022109.git>

## 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

在这个任务总需要处理以下几个任务

- 为接口 `Graph<L>` 编写通用的测试用例
- 实现接口的实现类 `ConcreteEdgesGraph<String>`，并编写相应的测试方法
- 实现接口的实现类 `ConcreteVerticesGraph<String>`，并编写相应的测试方法
- 将 `ConcreteEdgesGraph<String>` 和 `ConcreteVerticesGraph<String>` 修改为接受泛型 `L` 的类
- 编写诗意漫步的测试方法
- 实现诗意漫步中的方法

#### 3.1.1 Get the code and prepare Git repository

如何从 GitHub 获取该任务的代码、在本地创建 git 仓库、使用 git 管理本地开发。

如何从 GitHub 获取该任务的代码

在 Git Bash 中输入

```
git clone "仓库地址"
```

即可从 GitHub

在本地创建 git 仓库

进入某一目录后, 使用

```
git init
```

即可在本地创建 git 仓库

使用 git 管理本地开发。

添加到缓存区

```
git add <filename>
```

```
git add *
```

提交

```
git commit -m "代码提交信息"
```

### 3.1.2 Problem 1: Test Graph <String>

分别为 Graph 中定义的接口设计测试方法

testAdd 方法

- 测试添加一个普通的节点的情况

- 测试添加一个重复节点的情况

testSet 方法

- 测试添加正常的权值非零的边

- 测试添加节点不存在的边

- 测试改变权值

- 测试删去存在的边

- 测试删去不存在的边

testRemove 方法

- 删去一个存在的节点

- 删去一个不存在的节点

testVertices 方法

- 测试一个构建好的图

- 删除节点后再次测试

- 增加节点后再次测试

testSources 方法

- 测试一个正常情况

- 测试删去边的情况
  - 测试增加边的情况
- testTarget 方法
- 测试一个正常情况
  - 测试删去边的情况
  - 测试增加边的情况

### 3.1.3 Problem 2: Implement Graph <String>

#### 3.1.3.1 Implement ConcreteEdgesGraph

首先实现 Edge 类

根据要求 Edge 是一个不可变的类

使用 `private final` 来修饰其中所有关键字，且其中所有类型均是不可变的类型，如图 3-1 所示。

因为需要修改权值的操作，我们编写一个 `setWeight` 方法，但是在其中返回一个新的修改权值后的副本，如图 3-2 所示。

```
5 个用法
private final L source;

5 个用法
private final L target;

3 个用法
private final int weight;
// Abstraction function:
//   表示由source指向target的一条边
// Representation invariant:
//   source and target are not null
// Safety from rep exposure:
//   使用不可变的字符串类和int来作为rep
//   set方法返回新副本
```

图 3-1

```

1 个用法
public Edge<L> setWeight(int weight) {
    if(weight==0){
        return null;
    }
    return new Edge<>(this.source,this.target,weight);
}

```

lsxuan12138

图 3-2

接下来实现 ConcreteEdgesGraph

其中的 rep 已经规定好, 我们不再修改, 图 3-3 中是相对应的 RI 和 AF

```

11 个用法
private final Set<L> vertices = new HashSet<>();
17 个用法
private final List<Edge<L>> edges = new ArrayList<>();

// Abstraction function:
//   vertices中元素表示图的顶点
//   edges中每个元素表示图中一条有向边
// Representation invariant:
//   vertices中应该是无重复的
//   edges中应该无重复的
// Safety from rep exposure:
//   采用防御式拷贝
//   除变值器外, 其他方法不会改变值
//   在变值器中使用checkRep检查循环不变量

```

图 3-3

add 方法 public boolean add(L vertex)

若在图的节点集 vertices 中包含 vertex, 则直接返回 false

若不包含, 则向 vertices 中增加 vertex

set 方法 public int set(L source, L target, int weight)

首先对异常情况对象处理

若 source 或者 target 为 null 则抛出异常

若权值 weight 小于 0, 则爆出异常

然后进行处理, 分为两种情况

若图中已经包含边则对边的权值进行修改, 使用 setweight 方法, 得到新的边 edge 并删去原来的边

若此时 weight 是 0, 那么 edge 为 null, 此时, 不进行操作, 返回

若 weight>0, 那么把新得到的 edge 加入的边集 edges 中, 此时, 若 source 或者 target 不再顶点集中, 则添加进去

若图中没有相同的边,

若 weight=0, 不做修改, 直接返回

若 weight>0, 向 edges 中添加新的边此时, 若 source 或者 target 不再顶

点集中, 则添加进去

`remove` 方法 `public boolean remove(L vertex)`

首先对异常情况判断

若 `vertex` 为空则抛出异常

若顶点集中不包含 `vertex`, 直接返回 `false`

然后进行处理

将顶点集中的 `vertex` 直接移除

然后遍历边集, 若其中某一条边的 `source` 或者 `target` 与 `vertex` 相等, 将其移除

返回 `true`

`vertices` 方法 `public Set<L> vertices()`

直接拷贝顶点集并返回

`sources` 方法 `public Map<L, Integer> sources(L target)`

首先对异常情况判断

若 `target` 为空则抛出异常

然后进行处理

创建一个新的 `map` 用来存放结果

遍历边集, 若边的终点与 `target` 相同, 将其放入 `map`

返回 `map`

`targets` 方法 `public Map<L, Integer> targets(L source)`

首先对异常情况判断

若 `source` 为空则抛出异常

然后进行处理

创建一个新的 `map` 用来存放结果

遍历边集, 若边的起点与 `source` 相同, 将其放入 `map`

返回 `map`

测试 `ConcreteEdgesGraph`

首先测试 `Edge` 的各个方法



```

    public void testGetSource() {
        Edge edge = new Edge( source: "LiLei", target: "HanMeiMei", weight: 3);
        assertEquals( expected: "LiLei", edge.getSource());
    }

    @Test
    public void testGetTarget() {
        Edge edge = new Edge( source: "LiLei", target: "HanMeiMei", weight: 3);
        assertEquals( expected: "HanMeiMei", edge.getTarget());
    }

    @Test
    public void testGetWeight() {
        Edge edge = new Edge( source: "LiLei", target: "HanMeiMei", weight: 3);
        assertEquals( expected: 3, edge.getWeight());
    }

    @Test
    public void testEdgeEquals(){
        Edge edge = new Edge( source: "LiLei", target: "HanMeiMei", weight: 3);
        Edge edge1 = new Edge( source: "LiLei", target: "HanMeiMei", weight: 3);
        Edge edge2 = new Edge( source: "LiLei", target: "HanMeiMei", weight: 4);
        assertTrue(edge.equals(edge1));
        assertFalse(edge.equals(edge2));
    }

    @Test
    public void testSetWeight(){
        Edge edge = new Edge( source: "LiLei", target: "HanMeiMei", weight: 3);
        assertEquals(new Edge<String>( source: "LiLei", target: "HanMeiMei", weight: 2), edge.setWeight(2));
    }

```

然后对 ConcreteEdgesGraph.toString()进行测试

- 测试空图
- 测试有节点但没有边的图
- 测试有边有点的图

```

// tests for ConcreteEdgesGraph.toString()
@Test
public void testGraphToString(){
    Graph<String> graph = emptyInstance();
    assertEquals( expected: "", graph.toString());
    graph.set( source: "LiLei", target: "HanMeiMei", weight: 0);
    assertEquals( expected: "", graph.toString());
    graph.set( source: "LiLei", target: "HanMeiMei", weight: 4);
    graph.set( source: "LiLei", target: "XiaoHong", weight: 7);
    assertEquals( expected: "LiLei --4--> HanMeiMei\nLiLei --7--> XiaoHong\n", graph.toString());
}

```

### 3.1.3.2 Implement ConcreteVerticesGraph

实现 Vertex 类

首先先看 ConcreteVerticesGraph 中的 rep

```

18 个用法
private final List<Vertex<L>> vertices = new ArrayList<>();

// Abstraction function:
//   vertices的元素中
//   vertex为节点
//   edges表示以vertex为source的边
// Representation invariant:
//   vertices中元素不能有相同的节点vertex
// Safety from rep exposure:
//   防御式拷贝

```

因为 ConcreteVerticesGraph 中只有一个 Vertex 的 list  
 那么显然 Vertex 中应该同时有顶点与边的信息  
 我们使用 String vertex 来表示节点名称, 用 Map<String,Integer> edges 来表示  
 以 vertex 为起点的所有边的终点和对应的权值

```

5 个用法
private final L vertex;
6 个用法
private final Map<L,Integer> edges;
// Abstraction function:
//   vertex为节点
//   edges表示以vertex为source的边集
// Representation invariant:
//   edges中无重复元素
// Safety from rep exposure:
//   防御式拷贝

```

这里直接使用常规的 setter 与 getter 不做特殊的修改

然后实现 ConcreteVerticesGraph

add 方法 public boolean add(L vertex)

遍历图的点集 vertices 若其中包含 vertex, 则直接返回 false

若不包含, 则向 vertices 中增加 vertex

set 方法 public int set(L source, L target, int weight)

首先对异常情况对象处理

若 source 或者 target 为 null 则抛出异常

若权值 weight 小于 0, 则爆出异常

然后进行处理, 分为两种情况

若顶点集中有 source

遍历 source 的每一条边

若找到对应边,

若 weight>0, 则修改边, 返回

若 weight=0, 则删除边, 返回

若找不到对应边, 查看点集中是否存 target  
若有  
    weight=0, 则不处理  
    若 weight>0, 则向 source 的 map 中添加对应边  
    返回改变之前的权值  
若没有  
    添加节点 target, 并添加对应边, 返回 0

若顶点集中没有 source  
    若 weight 为 0, 则不处理, 直接返回 0  
    若 weight>0, 则在 vertices 中添加节点及对应的边  
        遍历 vertices 寻找 target, 若其中没有则添加节点 target, 返回 0

remove 方法 public boolean remove(L vertex)  
    首先对异常情况判断  
        若 vertex 为空则抛出异常  
    遍历顶点集中  
        若 vertices 中无 vertex, 直接返回 false  
        若有将顶点集中的 vertex 直接移除  
    遍历 vertices, 若其中某一个顶点的 edges 中包含 key: target, 将其移除  
    返回 true

vertices 方法 public Set<L> vertices()  
    new 一个新的 set  
    遍历 vertices 将每一个节点的 vertex 放入 set 中  
    返回 set

sources 方法 public Map<L, Integer> sources(L target)  
    首先对异常情况判断  
        若 target 为空则抛出异常  
    然后进行处理  
        创建一个新的 map 用来存放结果  
        遍历边集, 若边的终点与 target 相同, 将其放入 map  
    返回 map

targets 方法 public Map<L, Integer> targets(L source)  
    首先对异常情况判断  
        若 source 为空则抛出异常  
    然后进行处理  
        创建一个新的 map 用来存放结果  
        遍历边集 vertices, 若边的起点与 source 相同, 返回该 vertex 的 edges 的副本  
        若未找到, 返回空的 map

测试 ConcreteVerticesGraph

```

@Test
public void testGetVertex(){
    Vertex vertex = new Vertex( vertex: "XiaoMing", new HashMap<>());
    assertEquals( expected: "XiaoMing", vertex.getVertex());
}
新 *
@Test
public void testGetEdges(){
    Vertex vertex = new Vertex( vertex: "XiaoMing", new HashMap<>());
    Map<String,Integer> map = new HashMap<>();
    assertEquals(map, vertex.getEdges());
}
新 *
@Test
public void testAddEdge(){
    Vertex vertex = new Vertex( vertex: "XiaoMing", new HashMap<>());
    Map<String,Integer> map = new HashMap<>();
    assertEquals(map, vertex.getEdges());
    vertex.addEdge( target: "HanMei", weight: 3);
    map.put("HanMei", 3);
    assertEquals(map, vertex.getEdges());
}
新 *
@Test
public void testRemoveEdge(){
    Vertex vertex = new Vertex( vertex: "XiaoMing", new HashMap<>());
    Map<String,Integer> map = new HashMap<>();
    assertEquals(map, vertex.getEdges());
    vertex.addEdge( target: "HanMei", weight: 3);
    map.put("HanMei", 3);
    assertEquals(map, vertex.getEdges());
    map.remove( key: "HanMei");
    vertex.removeEdge( target: "HanMei");
}

```

测试 ConcreteVerticesGraph.toString()

```

@Test
public void testToString(){
    Graph<String> graph = emptyInstance();
    assertEquals( expected: "", graph.toString());
    graph.set( source: "LiLei", target: "HanMeiMei", weight: 0);
    assertEquals( expected: "", graph.toString());
    graph.set( source: "LiLei", target: "HanMeiMei", weight: 4);
    graph.set( source: "LiLei", target: "XiaoHong", weight: 7);
    assertEquals( expected: "LiLei --4--> HanMeiMei\nLiLei --7--> XiaoHong\n", graph.toString());
}

```

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

#### 3.1.4.1 Make the implementations generic

这一步只要将 ConcreteEdgesGraph 和 ConcreteVerticesGraph 在类名后增加中<L>, 并把其中的 String 都换成泛型 L, 也就是将 Edge 换成 Edge<L> 将 List<Edge>换成 List<Edge<L>>

等等

这里简单展示几个

```
0 1 用法 2  Isxuan12138
@Override public Map<L, Integer> targets(L source) {
    if (source == null) {
        throw new IllegalArgumentException("source is null");
    }
    Map<L, Integer> result = new HashMap<>();
    for (Edge<L> edge:
        edges) {
        if (source.equals(edge.getSource())) {
            result.put(edge.getTarget(), edge.getWeight());
        }
    }
    checkRep();
    return result;
}
```

```
Isxuan12138
public List<Edge<L>> getEdges() {
    List<Edge<L>> result = new ArrayList<>(this.edges);
    for (Edge<L> edge:
        edges) {
        result.add(new Edge(edge.getSource(), edge.getTarget(), edge.getWeight()));
    }
    checkRep();
    return result;
}
```

```
9 个用法  Isxuan12138
@Override public Set<L> vertices() {
    Set<L> result = new HashSet<>();
    for (Vertex<L> v:
        vertices) {
        result.add(v.getVertex());
    }
    return result;
}
```

```
8 个用法  Isxuan12138 *
@Override public Map<L, Integer> sources(L target) {
    if (target == null) {
        throw new RuntimeException();
    }
    Map<L, Integer> result = new HashMap<>();
    for (Vertex<L> v:
        vertices) {
        for (L str:
            v.getEdges().keySet()) {
            if (target.equals(str)) {
                result.put(v.getVertex(), v.getEdges().get(str));
            }
        }
    }
    return result;
}
```

### 3.1.4.2 Implement Graph.empty()

在这个任务中需要完成两个部分

-完成 Graph.empty()

-使用其他不可变类型完成 GraphStaticTest

实现 Graph.empty()

只要任选一个实现类, new 对象并返回即可

这里我选择了 ConcreteEdgesGraph

```
*/
9 个用法  · lsxuan12138
public static <L> Graph<L> empty() {
    return new ConcreteEdgesGraph<>();
}

/**
 * Add a vertex to this graph.
```

完成 GraphStaticTest

这里的基本测试策略与 GraphInstanceTest 相同, 故不再重复

只需要将其中的 String 换成其他的不可变类型即可, 这里选择了以下几个类

Integer, Long, Character, Byte, BigDecimal, BigInteger

这里举一个例子

```
@test
public void testTargets(){
    Graph<BigInteger> graph = Graph.empty();
    BigInteger liLei = new BigInteger( val: "777");
    BigInteger hanMeiMei = new BigInteger( val: "888");
    BigInteger xiaoMing = new BigInteger( val: "999");
    graph.add(liLei);
    graph.add(hanMeiMei);
    graph.add(xiaoMing);

    graph.set(liLei,hanMeiMei, weight: 5);
    graph.set(liLei,xiaoMing, weight: 4);
    BigInteger xiaoHong = new BigInteger( val: "1000");
    graph.set(xiaoHong,xiaoMing, weight: 6);
    Map<BigInteger,Integer> sources= graph.targets(liLei);
    Map<BigInteger,Integer> map = new HashMap<>();
    map.put(hanMeiMei,5);
    map.put(xiaoMing,4);
    assertEquals(map,sources);
    //测试删去一条边
    graph.set(liLei,xiaoMing, weight: 0);
    map.remove(xiaoMing);
    sources= graph.targets(liLei);
    assertEquals(map,sources);
    //test add an edge
    graph.set(liLei,xiaoMing, weight: 7);
    map.put(xiaoMing,7);
    sources= graph.targets(liLei);
    assertEquals(map,sources);
}
```



### 3.1.5 Problem 4: Poetic walks

在该任务中需要完成两个子任务

- 完成编写对 GraphPoet 中方法的测试
- 根据给出的要求实现 GraphPoet 中的方法

#### 3.1.5.1 Test GraphPoet

这个任务需要完成编写对 GraphPoet 中方法的测试

测试 GraphPoet 中的构造方法

```
public void testConstructor()
```

输入文件并构造图，检查其是否如预期一样

这里举一个例子

```
File text1 = new File( pathname: "test/P1/poet/simpleTest.txt");
GraphPoet graphPoet1 = new GraphPoet(text1);
String[] strs1 = {"hello,", "goodbye!"};
Set<String> vertices1 = new HashSet<>(Arrays.asList(strs1));
assertEquals(vertices1, graphPoet1.vertices());

Map<String, Integer> source1 = new HashMap<>();
source1.put("hello,", 1);
assertEquals(source1, graphPoet1.sources( target: "goodbye!"));

Map<String, Integer> target1 = new HashMap<>();
target1.put("hello,", 2);
target1.put("goodbye!", 1);
assertEquals(target1, graphPoet1.targets( source: "hello,"));
```

// 构造一个与给出的例子一样的图

另外，构造一个与给出例子一样的图来进行测试

### Problem 4: Poetic walks

Graphs — what are they good for? Poetry!

The specification of GraphPoet explains how a poet is initialized with a corpus and then, given an input, uses a word affinity graph defined by that corpus to transform the input poetically.

For example, here's a small but inspirational corpus:

```
To explore strange new worlds
To seek out new life and new civilizations
```

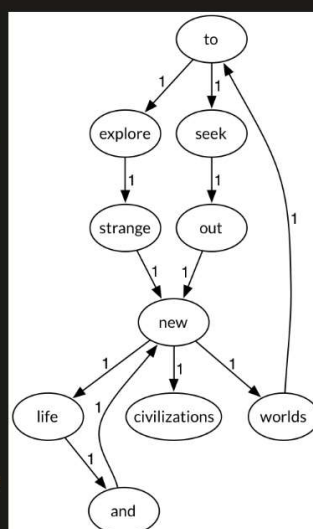
GraphPoet will use this corpus to generate a word affinity graph where:

- vertices are case-insensitive words, and
- edge weights are in-order adjacency counts.

The graph is shown on the right. In this example, all the edges have weight 1 because no word pair is repeated.

From there, given this dry bit of business-speak input:

```
Seek to explore new and exciting synergies!
```



```

Map<String, Integer> source = new HashMap<>();
source.put("strange",1);
source.put("out",1);
source.put("and",1);
assertEquals(source, graphPoet.sources( target: "new"));

Map<String, Integer> target = new HashMap<>();
target.put("life",1);
target.put("worlds",1);
target.put("civilizations",1);
assertEquals(target, graphPoet.targets( source: "new"));

```

测试 GraphPoet 中的 poem 方法

输入 input, 检查输出的 poet 是否一致

```

@Test
public void testPoem(){
    try {
        File text1 = new File( pathname: "test/P1/poet/simpleTest1.txt");
        GraphPoet graphPoet1 = new GraphPoet(text1);
        assertEquals( expected: "Test of the system.",
            graphPoet1.poem( input: "Test the system."));
        //构造一个与给的例子一样的图
        File text = new File( pathname: "test/P1/poet/seven-words.txt");
        GraphPoet graphPoet = new GraphPoet(text);

        assertEquals( expected: "Seek to explore strange new life and exciting synergies!",
            graphPoet.poem( input: "Seek to explore new and exciting synergies!"));
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

### 3.1.5.2 Implement GraphPoet

构造方法

public GraphPoet(File corpus) throws IOException

使用 Scanner 作为文件输入流

读入文件 corpus, 读入每一行并处理

将读入的一行以空格分隔, 并放入数组

遍历数组

若我们处理的是第一行, 那么直接进行处理

先设置

set(line[i].toLowerCase(), line[i + 1].toLowerCase(),1);

若返回的权值不为 0, 则重新设置 set(line[i].toLowerCase(),

line[i + 1].toLowerCase(), beforeWeight + 1);

若处理的不是第一行

对上一行的最后一个单词, 和这一行最后一个词, 进行相同处理



再进行同上处理

poem 方法

public String poem(String input)

创建一个 StringBuilder 存放结果

对输入用空格进行分割放入数组 words 中

遍历数组

先将数组中的元素 words[i]放入结果中

分别获取 words[i]在图中的 targets 和 words[i+1]的 sources

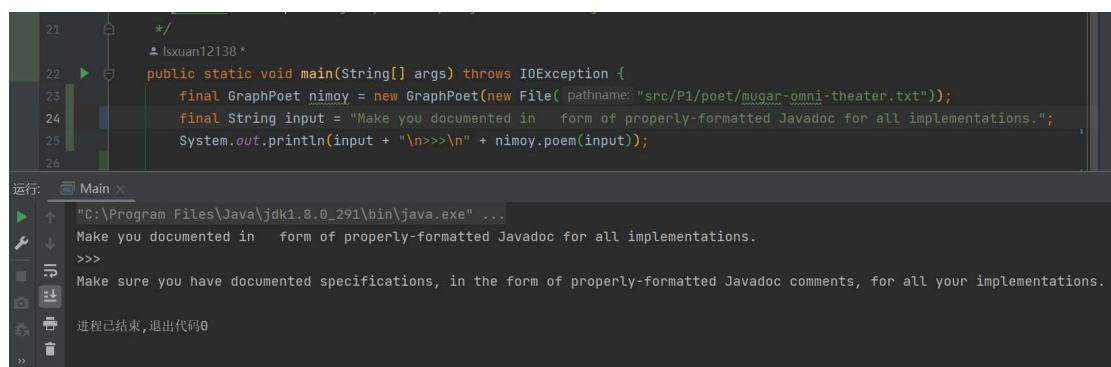
遍历 targets, 获取 targets 和 sources 共有的且路径权值最大的元素

将得到的元素放入结果中

将数组中最后一个元素放入结果

返回结果

### 3.1.5.3 Graph poetry slam



The screenshot shows a Java IDE with a code editor and a console window. The code in the editor is as follows:

```
21  */
22  * Isxuan12138 *
23  public static void main(String[] args) throws IOException {
24      final GraphPoet nimoy = new GraphPoet(new File( pathnames: "src/P1/poet/mugar-omni-theater.txt"));
25      final String input = "Make you documented in form of properly-formatted Javadoc for all implementations.";
26      System.out.println(input + "\n>>>\n" + nimoy.poem(input));
27  }
```

The console window shows the execution of the main method:

```
运行: Main <
"C:\Program Files\Java\jdk1.8.0_291\bin\java.exe" ...
Make you documented in form of properly-formatted Javadoc for all implementations.
>>>
Make sure you have documented specifications, in the form of properly-formatted Javadoc comments, for all your implementations.
进程已结束,退出代码0
```

### 3.1.6 使用 Eclemma 检查测试的代码覆盖率



The screenshot shows an IDE with a project explorer on the left and a code editor on the right. The project explorer shows the following structure:

- src
  - P1
    - graph 100% 类, 86% 行已覆盖
      - ConcreteEdgesGraph.java
        - ConcreteEdgesGraph 90% 方法, 83% 行已覆盖
        - Edge 90% 方法, 87% 行已覆盖
      - ConcreteVerticesGraph.java
        - ConcreteVerticesGraph 100% 方法, 89% 行已覆盖
        - Vertex 88% 方法, 80% 行已覆盖
      - Graph 100% 方法, 100% 行已覆盖
    - poet 100% 类, 95% 行已覆盖
      - GraphPoet 83% 方法, 95% 行已覆盖
      - Main 100% 方法, 100% 行已覆盖
      - mugar-omni-theater.txt
  - P2

The code editor shows the following code:

```
17  * well as
18  *
19  * Tests ag
20  */
21  * Isxuan12138
22  public clas
23  /*
24  * Prov
25  */
26  10 个用法
27  @Overri
28
29
```

### 3.1.7 Before you're done

请按照 [http://web.mit.edu/6.031/www/sp17/psets/ps2/#before\\_youre\\_done](http://web.mit.edu/6.031/www/sp17/psets/ps2/#before_youre_done) 的说明，检查你的程序。

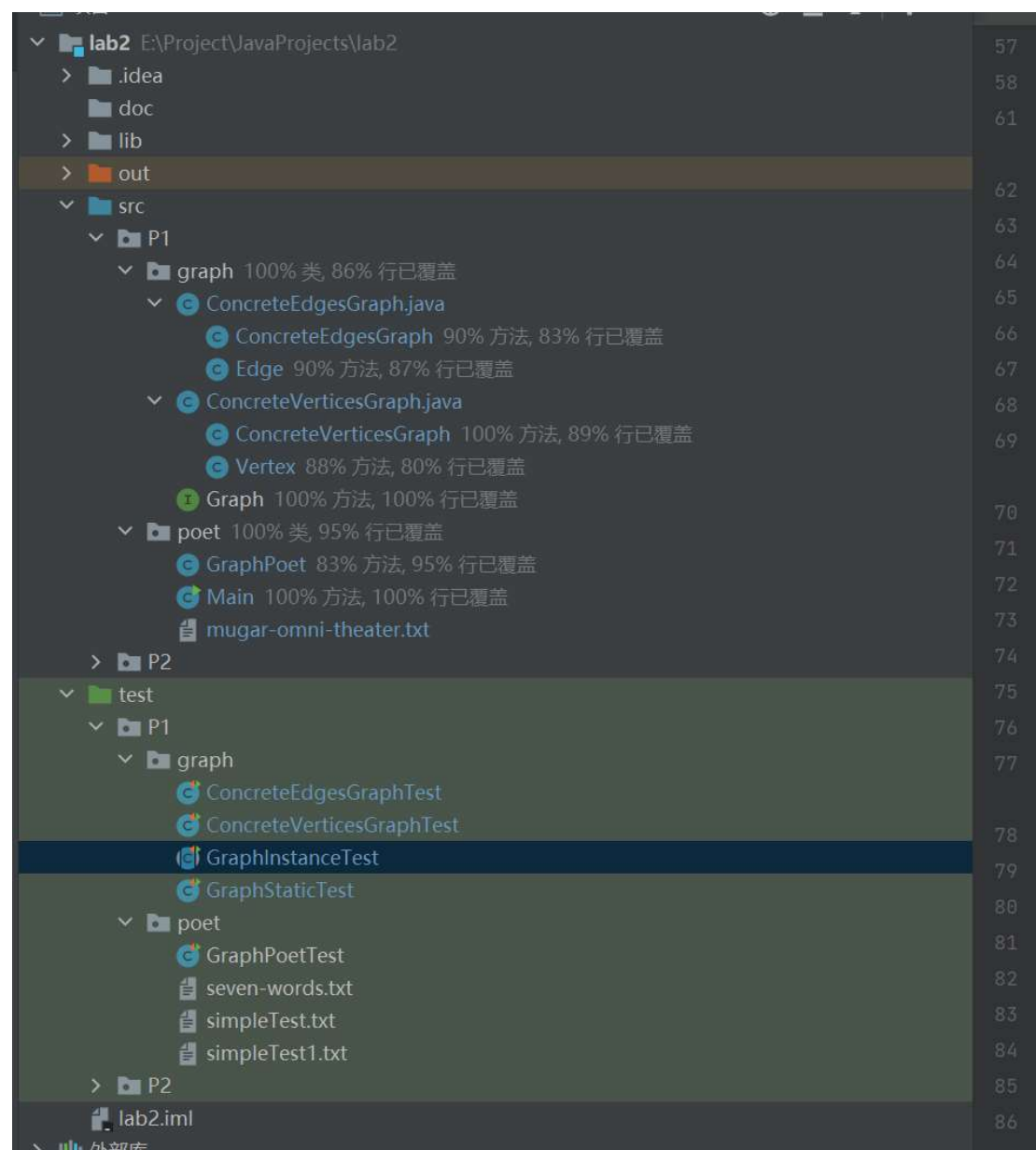
如何通过 Git 提交当前版本到 GitHub 上你的 Lab2 仓库。

使用指令

```
git push origin master
```

其中的 master 可以换为任何其他分支

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

该任务要求使用 P1 中实现的接口 `Graph<L>` 来重新实现在 lab1 中的 Social Network

### 3.2.1 FriendshipGraph 类

使用 `Graph<Person>` 作为 `FriendshipGraph` 的 rep

```
public class FriendshipGraph {  
    4 个用法  
    private final Graph<Person> friendshipGraph = Graph.empty();  
  
    /**  
     * 添加一个节点  
     * @param person 节点  
     */  
    public boolean addVertex(Person person)  
    public boolean addEdge(Person source, Person target)  
    public int getDistance(Person source, Person target)
```

`addVertex` 方法 `public boolean addVertex(Person person)`

若图的点集中已经包含 `person`, 则输出信息返回 `false`

若不包含, 则加入 `person`, 返回 `true`

`addEdge` 方法 `public boolean addEdge(Person source, Person target)`

先对异常情况处理

若 `source` 或者 `target` 为 `null`, 抛出异常

若 `source.equals(target)`, 输出不能添加自己为好友, 返回 `false`

若 `friendshipGraph.set(source, target, 1)` 的返回值不为 `0`, 则输出已经是好友了, 返回 `false`

若返回值为 `0`, 返回 `true`

`getDistance` 方法 `public int getDistance(Person source, Person target)`

先对异常情况处理

若 `source` 或者 `target` 为 `null`, 抛出异常

若 `source` 与 `target` 相同, 返回 `0`

接下来计算不同的两点间的最短路径长度

进行广度优先遍历, 这里做出如下处理

设置标记, 表示读完一层

若标记为 `true` 读完一层, 添加空节点, 作为记录, 将标记设为 `false`

若读到空节点, 表明读完一层, 设置标记为 `true`

若找到 `target` 节点, 返回

若在广度优先遍历后未返回

说明两个节点未连接, 返回 `-1`

### 3.2.2 Person 类

```
5 个用法
private final String name;
/**
 * 常量池，确保一次运行中，不会出现相同名字
 */
3 个用法
public static final Set<String> namePool;
static {
    namePool = new HashSet<>();
}
```

使用 String name 作为 Person 的 rep

设置一个静态的属性 Set<String> namePool 作为常量池，确保一次运行中，不会出现相同名字

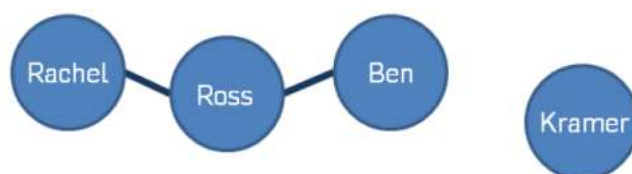
在构造方法中进行判断

```
16 个用法  Isxuan12138
public Person(String name) {
    if(Person.namePool.contains(name)){
        System.out.println("你输入的名字已存在");
        System.exit(status: -1);
    }
    namePool.add(name);
    this.name = name;
}
```

实现相应的 getter 方法

### 3.2.3 客户端 main()

使用使用指导书中的代码。构造这样的图



```
public static void main(String[] args) {  
    FriendshipGraph graph = new FriendshipGraph();  
    Person rachel = new Person( name: "Rachel");  
    Person ross = new Person( name: "Ross");  
    Person ben = new Person( name: "Ben");  
    Person kramer = new Person( name: "Kramer");  
    graph.addVertex(rachel);  
    graph.addVertex(ross);  
    graph.addVertex(ben);  
    graph.addVertex(kramer);  
  
    graph.addEdge(rachel, ross);  
    graph.addEdge(ross, rachel);  
    graph.addEdge(ross, ben);  
    graph.addEdge(ben, ross);  
  
    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

#### 3.2.4.1 测试 testAddVertex

测试正常添加节点的返回值是否正确

测试当节点重复添加时的反应

```
Person kramer = new Person( name: "Kramer");  
assertEquals( expected: true, graph.addVertex(rachel));  
assertEquals( expected: true, graph.addVertex(ross));  
assertEquals( expected: true, graph.addVertex(ben));  
assertEquals( expected: true, graph.addVertex(kramer));  
  
//当节点重复的情况下, 应返回false, 并输出提示信息  
assertEquals( expected: false, graph.addVertex(kramer));  
assertEquals( expected: kramer.getName()+"已存在", bytes.toString().trim());
```

#### 3.2.4.2 测试 testAddEdge

测试正常添加边的返回值是否正确

测试当 p1, p2 相同时, 程序的反应

测试重复添加边时的反应

```
assertEquals( expected: true, graph.addEdge(rachel, ross));
assertEquals( expected: true, graph.addEdge(ross, rachel));
assertEquals( expected: true, graph.addEdge(ross, ben));
assertEquals( expected: true, graph.addEdge(ben, ross));

assertEquals( expected: false, graph.addEdge(rachel, rachel));
assertEquals( expected: "你不能添加自己为好友", bytes.toString().split( regex: "\r\n")[0]);

assertEquals( expected: false, graph.addEdge(rachel, ross));
assertEquals( expected: "你们已经是好友了, 无需再次添加", bytes.toString().split( regex: "\r\n")[1]);

addEdgeException.expect(IllegalArgumentException.class);
graph.addEdge(null, new Person());
```

### 3.2.4.3 测试 testGetDistance

测试当两个点间只有一条有向边的反应

测试两个点间有正常的两条边的结果是否正确

```
//rachel和ross只有单向的关系
graph.addEdge(ross, rachel);
graph.addEdge(ross, ben);
graph.addEdge(ben, ross);

assertEquals( expected: -1, graph.getDistance(rachel, ross));
//should print -1
assertEquals( expected: -1, graph.getDistance(rachel, ben));
//should print -1
assertEquals( expected: 0, graph.getDistance(rachel, rachel));
//should print 0
assertEquals( expected: -1, graph.getDistance(rachel, kramer));
//should print -1

graph.addEdge(rachel, ross);
assertEquals( expected: 1, graph.getDistance(rachel, ross));
//should print 1
assertEquals( expected: 2, graph.getDistance(rachel, ben));
//should print 2
assertEquals( expected: 0, graph.getDistance(rachel, rachel));
//should print 0
assertEquals( expected: -1, graph.getDistance(rachel, kramer));
//should print -1
```

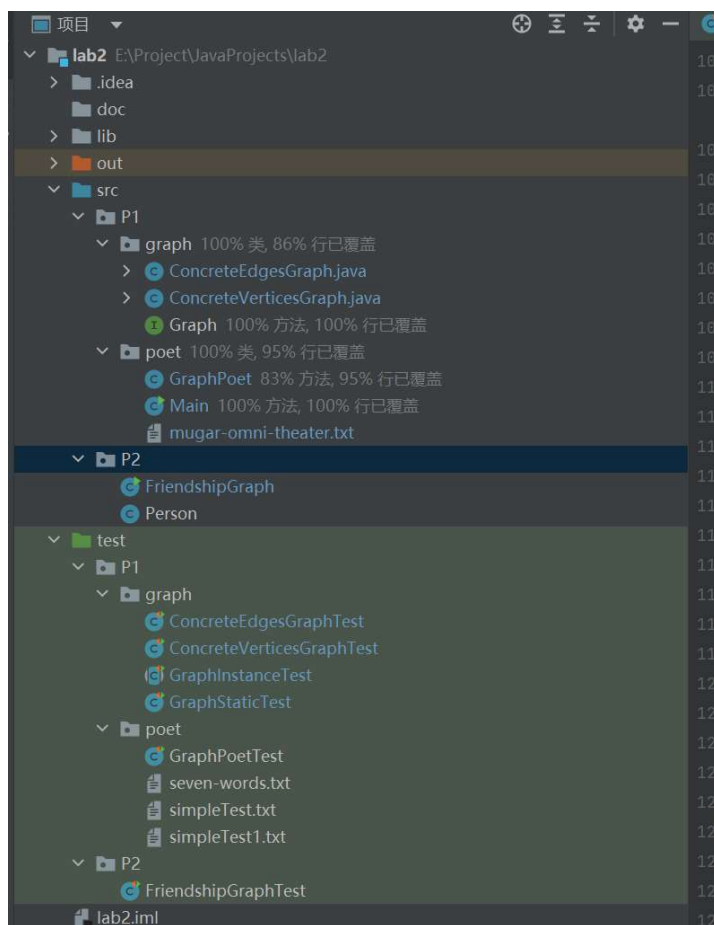
### 3.2.5 提交至 Git 仓库

如何通过 Git 提交当前版本到 GitHub 上你的 Lab2 仓库。

```
Xuan@Matebook14 MINGW64 /e/Project/JavaProjects/lab2 (master)
$
```

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。

日期	时间段	计划任务	实际完成情况
2022-5-11	16:00-17:30	完成 P1 problem1	完成
2022-5-14	18:00-20:30	完成 P1 problem2.1	完成
2022-5-14	20:00-22:30	完成 P1 problem2	完成
2022-5-15	17:30-18:30	修改 P1 problem1&2	完成
2022-5-15	18:30-19:00	完成 P1 problem3.1	完成
2022-5-15	19:00-20:00	完成 P1 problem3	完成
2022-5-15	20:20-21:15	完成 P1 problem4.1	完成
2022-5-15	21:20-22:25	完成 P1 problem4	完成
2022-5-16	15:45-16:15	修改 P1	完成
2022-5-16	16:20-17:05	完成 P2	完成
2022-5-16	17:05-17:30	修改 P1 problem4.2	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 Vertex 类中 rep 的设计	仔细观察 ConcreteVerticesGraph 的 rep 后, 自行解决 因为 ConcreteVerticesGraph 中只有一个 Vertex 的 list 那么显然 Vertex 中应该同时有顶点与边的信息 我们使用 String vertex 来表示节点名称, 用 Map<String,Integer> edges 来表示以 vertex 为起点的所有边的终点和对应的权值
GraphPoet 的构造方法中, 因为以行为单元来处理, 所以上一行的最后一个单词和下一行第一个单词的连接, 一开始被忽略	重新设计方法 若我们处理的是第一行, 那么直接进行处理 先设置 set(line[i].toLowerCase(), line[i + 1].toLowerCase(), 1); 若返回的权值不为 0, 则重新设置 set(line[i].toLowerCase(), line[i + 1].toLowerCase(), beforeWeight + 1); 若处理的不是第一行 对上一行的最后一个单词, 和这一行最后一个词, 进行相同处理

## 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教训 (必答)

初步学会设计 ADT 规约 (pre-condition、post-condition)

了解了 ADT 的泛型化;

学会使用 idea 评估测试的覆盖度;

体会到 ADT 的复用带来的好处;

在测试代码中, 写 testing strategy 并据此设计测试用例。

### 6.2 针对以下方面的感受 (必答)

(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, 对于 ADT 的设计者而言, 是一个相当困难的过程, 我们要设计一整套对应的操作接口, 并进行实现, 在这个过程中, 还有考虑到各种各样的异常情况

对应 ADT 的使用者而言, 这种方式会非常的便捷, 我们可以选用第三方开发的类库来为自己的程序做支撑, 而不必对所有的操作都自己编写



- (2) 使用泛型和不使用泛型的编程, 对你来说有何差异?

差别不大

使用泛型在编程过程中只能使用一些通用的方法, 除此之外, 其他的功能通常都需要自己实现

不使用泛型, 针对一些类型, 也许会有一些更加便捷的操作

- (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?

使用这种方式, 我们对 ADT 的测试会更加全面且天马行空, 不会因为实现 ADT 时的某些想法造成先入为主

目前还算适应

- (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?

这种复用可以大大节省程序员开发需要 ADT 支撑的功能时的时间, 不必将大量的时间花费在太细节的具体实现上

- (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?

这些工作可以在编程过程中为我们提供指导和约束, 在设计好后, 我们的所有工作都必须在这个前提下进行, 可以减少思考的可能性, 也减少出错的可能

- (6) 关于本实验的工作量、难度、deadline。

工作量比较大

难度适中

Deadline 比较宽松

- (7) 《软件构造》课程进展到目前, 你对该课程有何收获和建议?

学习到现在, 对 Java 或者说对面向对象编程有了更深的认识

学到了一些很通用也非常优秀的编程思想