



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	熊峰
学号	1190200708
班号	1903008
电子邮件	<a href="mailto:xiong257246@outlook.com">xiong257246@outlook.com</a>
手机号码	13114991114

## 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>	2
3.1.3 Problem 2: Implement Graph <String>	5
3.1.3.1 Implement ConcreteEdgesGraph	5
3.1.3.2 Implement ConcreteVerticesGraph	12
3.1.4 Problem 3: Implement generic Graph<L>	19
3.1.4.1 Make the implementations generic	19
3.1.4.2 Implement Graph.empty()	20
3.1.5 Problem 4: Poetic walks	22
3.1.5.1 Test GraphPoet	22
3.1.5.2 Implement GraphPoet	24
3.1.5.3 Graph poetry slam	25
3.1.6 使用 Eclemma 检查测试的代码覆盖率	25
3.1.7 Before you're done	26
3.2 Re-implement the Social Network in Lab1	28
3.2.1 FriendshipGraph 类	28
3.2.2 Person 类	30
3.2.3 客户端 main()	30
3.2.4 测试用例	31
3.2.5 提交至 Git 仓库	33
4 实验进度记录	36
5 实验过程中遇到的困难与解决途径	36
6 实验过程中收获的经验、教训、感想	36
6.1 实验过程中收获的经验教训	36
6.2 针对以下方面的感受	37

## 1 实验目标概述

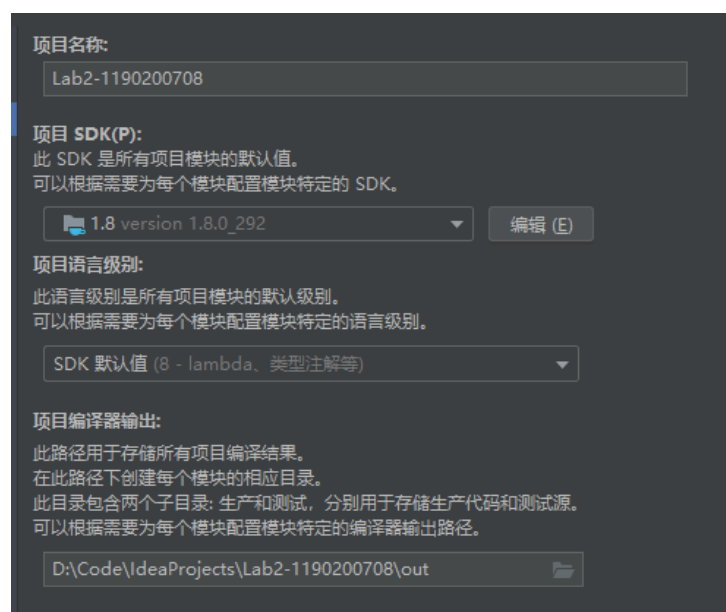
本次实验训练抽象数据类型（ADT）的设计、规约、测试，并使用面向对象编程（OOP）技术实现 ADT。具体来说：

- 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

环境配置：

IntelliJ IDEA 2020.3.1 (JDK 1.8 Junit 4.12(下载到 lib 目录中))



Git:

Git version 2.24.1.windows.2

MINGW64:/c/Users/Alienware/Desktop

```
Alienware@DESKTOP-GD6M11S MINGW64 ~/Desktop
$ git --version
git version 2.24.1.windows.2
```

GitHub Lab2 URL:

<https://github.com/ComputerScienceHIT/HIT-Lab2-1190200708>

### 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

满足所提供接口的规范。在某些情况下，可以加强提供的规范或添加新方法，但在其他情况下，不允许更改它们。

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

使用 `git clone` 命令从远程仓库中获取源代码

`git clone https://github.com/rainywang/Spring2020\_HITCS\_SC\_Lab2/tree/master/P1`

并在本地使用 `git init` 命令建立 `git` 仓库。

通过 `git add .` 命令添加所有文件，在通过 `git commit -m "message"` 的命令，提交本地项目，通过 `git remote add origin <url>` 的命令关联远程仓库，并使用 `git push -u origin master` 的命令，将其推送到远程 `github` 仓库。

我们也可以通过 `git log` 命令，详细查看提交记录，可以查看到不同版本。

总结：我们可以通过 `git status` 查看状态，`git add .` 添加文件，`git commit -m "描述信息"` 生成版本控制，`git log` 查看不同版本。

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

以下各部分，请按照 MIT 页面上相应部分的要求，逐项列出你的设计和实现思路/过程/结果。

1> `public boolean add(L vertex);`

设计：我们此时根据图是否为空，待添加顶点是否在图中作为输入的区分，因此我们应该对这几种情况的笛卡尔积做测试。将此时分为三种情况：

图为空，则 <code>vertex</code> 不在图中
图不为空，且 <code>vertex</code> 不在图中
图不为空，且 <code>vertex</code> 在图中

思路、过程：根据方法上方的注释分析，此时我们需要添加一个顶点到这个图，若该图中没有包含该顶点，则返回 `true`，否则我们应该返回 `false`。因此，对于这种情况，我们应该测试输入作如下分区：图是否为空图，当前顶点是否在图中。并对这两种情况做笛卡尔积，排除掉图为空，且当前顶点在图中的情况。

结果：分别对三种情况做测试，测试通过。

2> `public int set(L source, L target, int weight);`

设计：此时根据图是否为空，当前的边是否在图中，当前的顶点是否在图中，以及 `weight` 是否为 0 分区。对这些分区求笛卡尔积做测试，由于分区的笛卡尔积存在某些不可能出现的情况，故此时分为以下几种情况：

顶点在图中，边不在图中，图不为空， <code>weight=0</code>
顶点不在图中，边不在图中，图为空， <code>weight=0</code>
顶点在图中，边在图中，图不为空， <code>weight=0</code>
顶点在图中，边在图中，图不为空， <code>weight!=0</code>
顶点在图中，边不在图中，图不为空， <code>weight!=0</code>
顶点不在图中，边不在图中，图为空， <code>weight!=0</code>

思路、过程：根据方法上方的注释分析，我们需要添加，改变，或删除一个加权有向边在这个图中，如果 `weight!=0`，则添加一条边，或更新该边的权值。若给定标签的顶点没有出现，则带有给定的标签的顶点将被添加到图中，若 `weight==0`，且存在这条边，则去除这条边，返回之前边的权值。因此我们可以据此，对图是否为空，当前的边是否在图中，当前的顶点是否在图中，以及 `weight` 是否为 0 分区

结果：分别对以上几种情况测试，测试通过。

3> `public boolean remove(L vertex);`

设计：此时可以根据当前顶点是否在图中进行分区，则一共分为两种。

当前顶点在图中
当前顶点不在图中

思路、过程：根据方法上方的注释分析，从图中去掉一个顶点，任何和这个顶点相关的边都删除。

结果：分别对以上几种情况测试，测试通过。

4> `public Set<L> vertices();`

设计：此时可以根据图是否为空，判断当前是否会返回值，因此分为以下两种情况。

图为空
-----

图不为空
------

思路、过程：根据方法上方的注释分析，返回这个图中所有的顶点。

结果：分别对以上两种情况测试，测试通过。

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

设计：此时可以根据顶点是否在图中，顶点是否存在源顶点分区。

顶点在图中，顶点存在源顶点
---------------

顶点不在图中，顶点不存在源顶点
-----------------

顶点在图中，定点不存在源顶点
----------------

顶点不在图中，顶点存在源顶点
----------------

思路、过程：根据方法上方的注释分析，获取带有指向目标顶点的边的源顶点以及这些边的权值。因此我们可以根据当前顶点是否在图中，以及定点是否存在源顶点分区。

结果：分别对以上四种情况测试，测试通过。

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

设计：此时可以根据顶点是否在图中，顶点是否存在目标顶点分区。

顶点在图中，顶点存在目标顶点
----------------

顶点不在图中，顶点存在目标顶点
-----------------

顶点在图中，顶点不存在目标顶点
-----------------

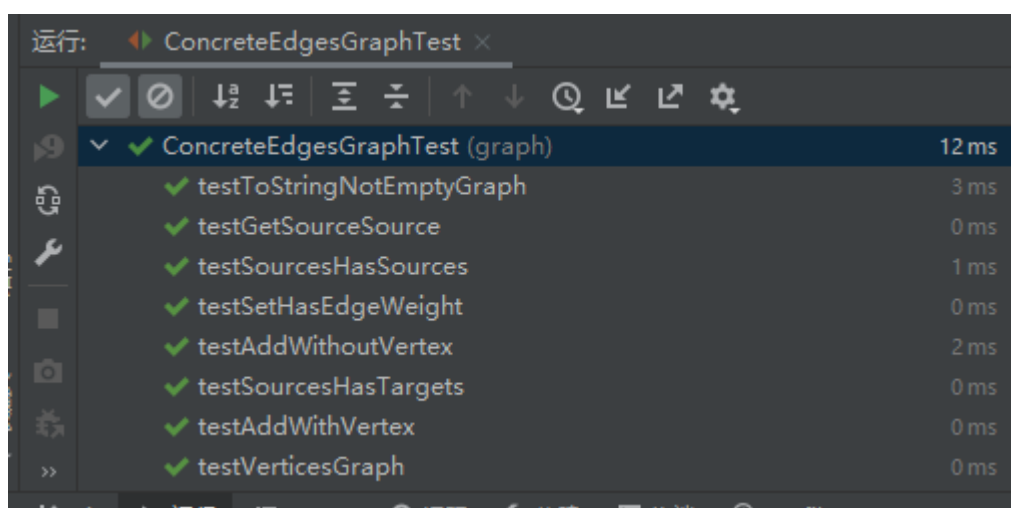
顶点不在图中，顶点不存在目标顶点
------------------

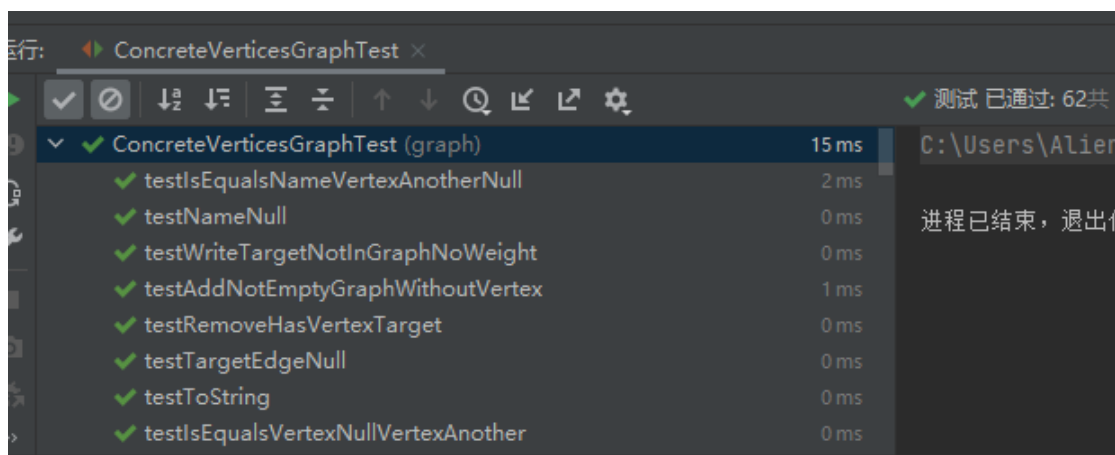
思路、过程：根据方法上方的注释分析，我们需要从源顶点获得带有有向边的目标顶点以及这些边的权值。因此我们可以对当前顶点是否在图中、当前顶点是否存在目标顶点分区。

结果：分别对以上四种情况测试，测试通过。

以下截图为在实现 Problem2 后对 Problem1 进行的测试：

依次通过 `ConcreteEdgesGraph` 与 `ConcreteVerticesGraph` 实现：





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

以下各部分, 请按照 MIT 页面上相应部分的要求, 逐项列出你的设计和实现思路/过程/结果。

#### 3.1.3.1 Implement ConcreteEdgesGraph

1> Edge 类:

Edge 类的私有变量如下, 由于 Edge 类的变量都是用 `private`、`final` 修饰, 且 `String`、`Integer` 均为不可变类型, 因此 Edge 类不可变:

<code>private final String source;</code>	边的起点
<code>private final String target;</code>	边的终点
<code>private final Integer weight;</code>	边的权值

Edge 类的方法如下:

<code>public Edge(final String sourceConstruct,final String targetConstruct,final int weightConstruct)</code>	构造函数, 通过参数中的
<code>public void checkRep()</code>	检查不变量, <code>weight&gt;0</code> , <code>source!=null</code> , <code>target!=null</code>
<code>public String getWeight ()</code>	获取边的 <code>weight</code> 值
<code>public String getSource ()</code>	获取边的起点
<code>public String getTarget()</code>	获取边的终点
<code>public String toString()</code>	返回边的字符串表示

// Abstraction function:

// 从 `source` 到 `target`, 且权值为 `weight` 的有向边

// Representation invariant:

// `weight>0`, 且 `source!=null`, `target!=null`

// Safety from rep exposure:

// 使用 `private` 和 `final` 修饰变量, 且使用变量的类型均为不可变类型。

// 避免从外部直接修改的风险。

CheckRep:

```
private void checkRep()
{
    assert this.weight>=0;
    assert this.target!=null;
    assert this.source!=null;
    //自环
    //assert !this.source.equals(this.target)
}
```

我们此时只需要检查 `this.weight>0`, `this.target!=null`, `this.source!=null`, 即可完成对不变量的检查。

toString:

```
@Override
public String toString()
{
    StringBuilder stringBuilder = new StringBuilder("Source:");
    stringBuilder.append(this.source+" ");
    stringBuilder.append("Target:"+this.target);
    stringBuilder.append(" Weight:"+this.weight);
    return stringBuilder.toString();
}
```

将 toString 函数打印方法设置如上，方便读取。

2> ConcreteEdgesGraph 类:

ConcreteEdgesGraph 有两个私有变量:

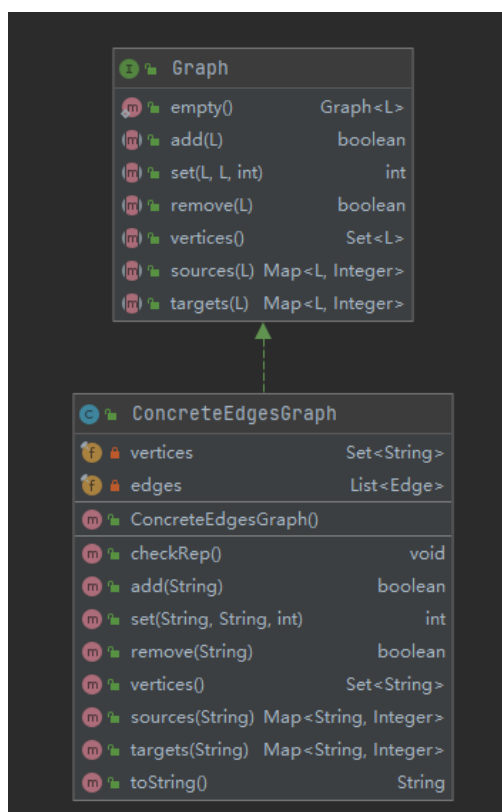
<code>private final Set&lt;String&gt; vertices = new HashSet&lt;&gt;()</code>	图的所有顶点
<code>private final List&lt;Edge&gt; edges = new ArrayList&lt;&gt;()</code>	图的所有边

ConcreteEdgesGraph 的方法如下:

<code>public ConcreteEdgesGraph()</code>	构造函数，创建一个空图
<code>public void checkRep()</code>	检查不变量
<code>public boolean add(String vertex)</code>	向图中添加一个边
<code>public int set(String source, String target, int weight)</code>	添加、该边、或删除一个图中的一个加权有向边
<code>public boolean remove(String vertex)</code>	去掉图中的一个顶点
<code>public Set&lt;String&gt; vertices()</code>	返回图中所有的顶点
<code>public Map&lt;String, Integer&gt; sources(String target)</code>	返回 target 顶点的所有源顶点及权值的映射
<code>public Map&lt;String, Integer&gt; targets(String source)</code>	返回 source 顶点的所有目标点及权值的映射
<code>public String toString()</code>	返回图的字符串表示

继承关系如下所示:





// Abstraction function:

// 由顶点为 vertices、边为 edges 组成的图

// 即从 vertices、edges 到有向图的映射

// Representation invariant:

// edges.getWeight>0,且 edges 中不存在两条相同的边

// vertices 中的顶点不为空

// Safety from rep exposure:

// 使用 private 和 final 修饰变量

// 使用防御性复制

CheckRep:

```

private void checkRep()
{
    for(Edge<L> edge1:edges)
    {
        assert edge1.getWeight()>=0;
        for(Edge<L> edge2:edges)
        {
            if(edge1.getSource().equals(edge2.getSource()))
            {
                if(edge1.getTarget().equals(edge2.getTarget()))
                {
                    assert edges.indexOf(edge1)==edges.indexOf(edge2);
                }
            }
        }
    }
    for(L vertex:vertices)
    {
        assert vertex!=null;
    }
}
  
```

检查 edges 中是否存在相同的边, edges 中每条边的权值是否大于, 以及是否

存在顶点为空的情况。

toString:

```
@Override
public String toString()
{
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.append("Vertices:");
    for(String vertex:this.vertices)
    {
        stringBuilder.append(" ");
        stringBuilder.append(vertex);
    }
    stringBuilder.append("\nEdges:");
    for(Edge edge:this.edges)
    {
        stringBuilder.append(" ");
        stringBuilder.append(edge.getSource());
        stringBuilder.append("->");
        stringBuilder.append(edge.getTarget());
        stringBuilder.append(":");
        stringBuilder.append(edge.getWeight());
    }
    return stringBuilder.toString();
}
```

将 toString 函数打印方法设置如上，方便读取。

### 3> 测试策略

对 ConcreteEdgesGraph 类的测试策略如下：

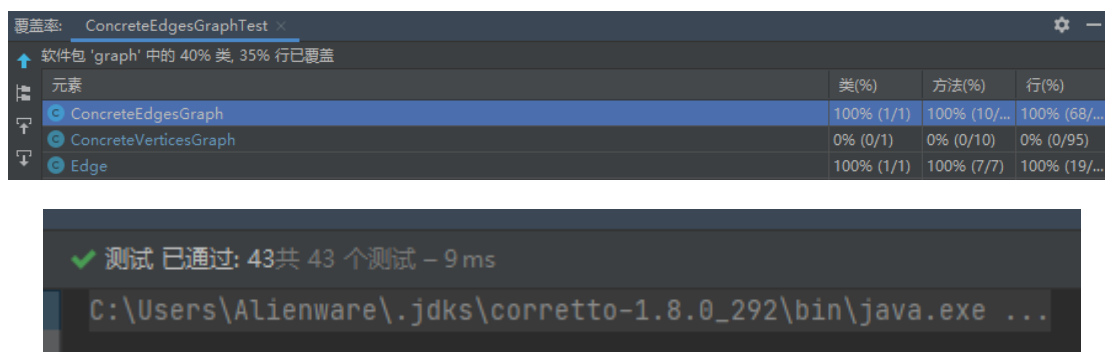
1. public boolean add(String vertex)  
// Testing strategy for ConcreteEdgesGraph.add()  
//  
// Partition the inputs as follows:  
// whether the vertex is in the vertices  
//  
// Exhaustive Cartesian coverage of partitions.
2. public int set(String source, String target, int weight)  
// Testing strategy for ConcreteEdgesGraph.set()  
//  
// Partition the inputs as follows:  
// whether the edge is in the graph  
// weight : >0 =0  
//  
// Exhaustive Cartesian coverage of partitions.
3. public boolean remove(String vertex)  
// Testing strategy for ConcreteEdgesGraph.remove()  
//  
// Partition the inputs as follows:  
// whether the vertex is in the graph  
//

```
// Exhaustive Cartesian coverage of partitions.
4. public Set<String> vertices()
// Testing strategy for ConcreteEdgesGraph.vertices()
//
// Partition the inputs as follows:
// whether the graph is empty
//
// Exhaustive Cartesian coverage of partitions.
5. public Map<String, Integer> sources(String target)
// Testing strategy for ConcreteEdgesGraph.sources()
//
// Partition the inputs as follows:
// whether the target has sources
//
// Exhaustive Cartesian coverage of partitions.
6. public Map<String, Integer> targets(String source)
// Testing strategy for ConcreteEdgesGraph.targets()
//
// Partition the inputs as follows:
// whether the source has targets
//
// Exhaustive Cartesian coverage of partitions.
7. public String toString()
// Testing strategy for ConcreteEdgesGraph.toString()
//
// Partition the inputs as follows:
// whether the graph is empty
//
// Exhaustive Cartesian coverage of partitions.
对 Edge 类的测试策略如下：
1. public String getWeight ()
// Testing strategy for Edge.getWeight()
//
// Partition the inputs as follows:
// Whether the weght is null
//
// Exhaustive Cartesian coverage of partitions.
2. public String getSource ()
// Testing strategy for Edge.getSource()
//
// Partition the inputs as follows:
// Whether the source is null
//
// Exhaustive Cartesian coverage of partitions.
```

```
3. public String getTarget()
// Testing strategy for Edge.getTarget()
//
// Partition the inputs as follows:
// Whether the target is null
//
// Exhaustive Cartesian coverage of partitions.
4. public String toString()
// Testing strategy for Edge.toString()
//
// Partition the inputs as follows:
// Whether the Edge is empty
//
// Exhaustive Cartesian coverage of partitions.
```

4> 测试结果：

可以看到所有测试通过，并且，代码覆盖率为 100%.



ConcreteEdgesGraphTest	
ConcreteEdgesGraphTest (graph)	9 ms
testToStringNotEmptyGraph	1 ms
testGetSourceSource	0 ms
testSourcesHasSources	0 ms
testSetHasEdgeWeight	0 ms
testAddWithoutVertex	0 ms
testSourcesHasTargets	0 ms
testAddWithVertex	0 ms
testVerticesGraph	0 ms
testToStringEmptyEdge	1 ms
testSetHasEdgeNoWeight	0 ms
testGetWeightNoWeight	0 ms
testGetTargetNoTarget	0 ms
testVerticesEmptyGraph	0 ms
testGetTargetTarget	0 ms
testGetSourceNoSource	0 ms
testRemoveHasNotVertex	0 ms
testSetHasnotEdgeNoWeight	0 ms
testGetWeightWeight	0 ms
testToStringEmptyGraph	0 ms
testSetHasnotEdgeWeight	0 ms
testSourcesHasNotSources	0 ms
testToStringEdge	0 ms
testRemoveHasVertex	1 ms
testSourcesHasNotTargets	0 ms
testSourcehasVerticesnoSource	1 ms
testSethasWeighthasEdgehasVertex	4 ms
testSetnoWeighthasEdgehasVertex	0 ms
testTargetsnVertices	0 ms
testAddhasVertex	0 ms
testSethasWeightnoEdgehasVertex	0 ms
testVerticesemptyGraph	0 ms
testAddnoVertex	0 ms
testInitialVerticesEmpty	0 ms
testSourcehasVerticeshasSource	1 ms
testAssertionsEnabled	0 ms
testRemovehasVertex	0 ms
testSourcenoVertices	0 ms
testSetnoWeightnoEdgehasVertex	0 ms
testTargetshasVerticeshasTarget	0 ms
testTargetshasVerticesnoTarget	0 ms
testSethasWeightnoEdgenoVertex	0 ms
testSetnoWeightnoEdgenoVertex	0 ms

### 3.1.3.2 Implement ConcreteVerticesGraph

1> Vertex 类:

Vertex 中含有两个私有变量:

```
private final String vertex;
private final Map<String, Integer> TargetEdge = new HashMap<>();
```

以下为 Vertex 中的方法:

public Vertex(final String VertexName);	构造函数
public void checkRep();	检查不变量
public boolean isEqualVertex(Vertex vertexAnother);	检查顶点的名字是否与当前顶点名字相同
public boolean isEqualName(String vertexAnother);	检查字符串是否与当前顶点名字相同
public void writeTarget(String vertexAnother, Integer weight);	将有向边写入 TargetEdge 中
public String Name();	返回当前顶点的名字
public Map<String, Integer> Target();	返回当前顶点所有目标顶点及对应权值
public Integer weight(String vertexAnother);	返回当前顶点所有源顶点及对应权值
public boolean remove(String vertexTarget);	去除掉当前顶点的 vertexTarget 目标顶点
public String toString();	返回图的字符串表示

// Abstraction function:

// 由顶点的名字及其对应目标顶点的映射所对应的实际顶点表示

// 即由 String、Map 组成的抽象数据类型对应的顶点

// Representation invariant:

// vertex 不为空且 TargetEdge 中的 weight>0

// Safety from rep exposure:

// 使用 private、final 修饰的变量

// 防御性复制

CheckRep:

```
private void checkRep() {
    if (!TargetEdge.isEmpty()) {
        for (L VertexName : TargetEdge.keySet()) {
            assert !this.vertex.equals(VertexName);
        }
        for (Integer VertexWeight:TargetEdge.values())
        {
            assert VertexWeight>0;
        }
    }
}
```

检查 vertex 不为空且 TargetEdge 中的 weight>0。

toString:

将 toString 函数打印方法设置如上，方便读取。

```
@Override
public String toString() {
    StringBuilder string = new StringBuilder();
    string.append(vertex);
    string.append("\n");
    for (Map.Entry<String, Integer> map : this.TargetEdge.entrySet()) {
        string.append("->");
        string.append(map.getKey());
        string.append(" value = ");
        string.append(map.getValue());
        string.append("\n");
    }
    checkRep();
    return string.toString();
}
```

2> ConcreteVerticesGraph 类:

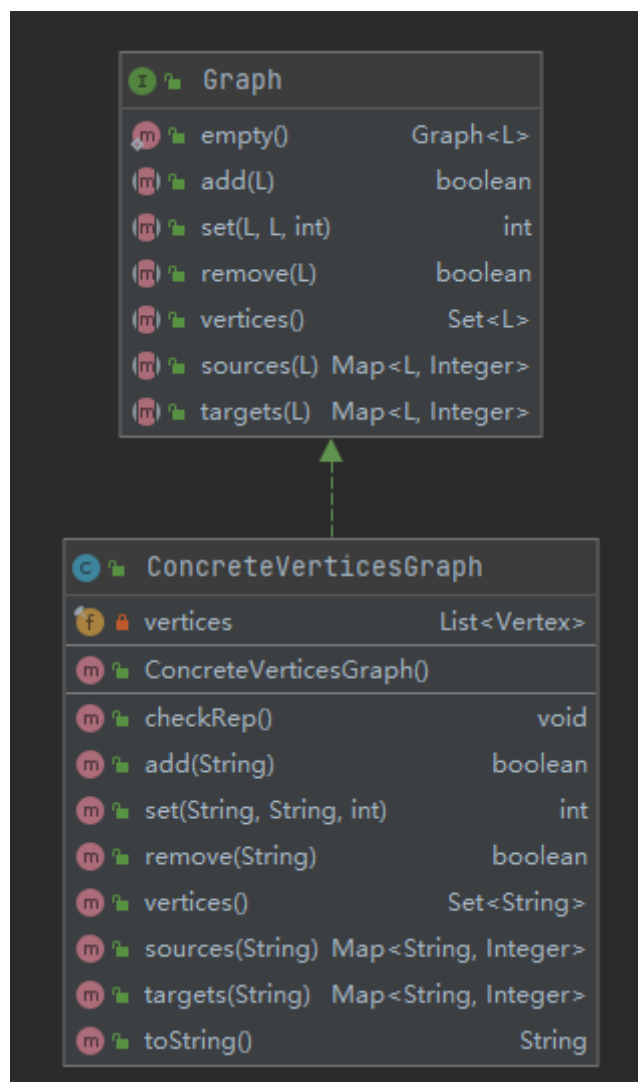
ConcreteVerticesGraph 类有一个私有变量:

<code>private final List&lt;Vertex&gt; vertices = new ArrayList&lt;&gt;()</code>	存储所有顶点
--	--------

以下为 ConcreteVerticesGraph 的方法:

<code>public ConcreteVerticesGraph()</code>	构造函数，创建新的空的图
<code>public void checkRep()</code>	检查不变量
<code>public boolean add(String vertex)</code>	将顶点添加到图中
<code>public int set(String source, String target, int weight)</code>	添加、该边、或删除一个图中的一个加权有向边
<code>public boolean remove(String vertex)</code>	从图中去掉 vertex 顶点
<code>public Set&lt;String&gt; vertices()</code>	返回图中所有顶点的集合
<code>public Map&lt;String, Integer&gt; sources(String target)</code>	返回 target 在图中的所有源顶点
<code>public Map&lt;String, Integer&gt; targets(String source)</code>	返回 source 在图中的所有目标顶点
<code>public String toString()</code>	返回图的字符串表示

继承关系如下:



// Abstraction function:

// 由 vertices 组成的顶点

// 即由 List<Vertex>类型数据到加权有向图的映射

// Representation invariant:

// vertices 中不存在重复的顶点

// Safety from rep exposure:

// 使用 private、final 修饰变量

// 采用防御性复制

CheckRep:

```

private void checkRep() {
    for (Vertex vertex1 : vertices) {
        for (Vertex vertex2 : vertices) {
            if (vertex1.isEqualsVertex(vertex2)) {
                assert (vertices.indexOf(vertex1) == vertices.indexOf(vertex2));
            }
            continue;
        }
    }
}

```



toString:

将 toString 函数打印方法设置如上, 方便读取.

```
@Override
public String toString() {
    StringBuilder str = new StringBuilder("The composition graph is as follows:\n");
    for (Vertex vertex : vertices) {
        str.append(vertex.Name());
        for (Map.Entry<String, Integer> mapVertex : vertex.Target().entrySet()) {
            str.append("  Weight(");
            str.append(vertex.Name());
            str.append(", ");
            str.append(mapVertex.getKey());
            str.append(")=");
            str.append(mapVertex.getValue());
        }
        str.append("\n");
    }
    checkRep();
    return str.toString();
}
```

3> 测试策略:

对 ConcreteVerticesGraph 的测试策略:

1. public boolean add(String vertex)

// Testing strategy for ConcreteVerticesGraph.add()

//

// Partition the inputs as follows:

// 是否为空图: empty graph? yes 、 no

// 当前的顶点是否在图中: whether vertex is in the graph? yes 、 no

//

// Exhaustive Cartesian coverage of partitions.

2. public int set(String source, String target, int weight)

// Testing strategy for ConcreteVerticesGraph.set()

//

// Partition the inputs as follows:

// 当前的顶点是否在图中: whether vertex is in the graph? yes 、 no

// value of weight? =0、 >0

// 顶点是否存在目标顶点: whether vertex has target? yes 、 no

//

// Exhaustive Cartesian coverage of partitions.

3. public boolean remove(String vertex)

// Testing strategy for ConcreteVerticesGraph.remove()

//

// Partition the inputs as follows:

// 是否为空图: empty graph? yes 、 no

// 当前的顶点是否在图中: whether vertex is in the graph? yes 、 no

//

```

// Exhaustive Cartesian coverage of partitions.
4. public Set<String> vertices()
// Testing strategy for ConcreteVerticesGraph.vertices()
//
// Partition the inputs as follows:
// 是否为空图: empty graph? yes 、 no
//
// Exhaustive Cartesian coverage of partitions.
5. public Map<String, Integer> sources(String target)
// Testing strategy for ConcreteVerticesGraph.sources()
//
// Partition the inputs as follows:
// 是否为空图: empty graph? yes 、 no
// 当前的顶点是否在图中: whether vertex is in the graph? yes 、 no
// 顶点是否存在源顶点: whether vertex has source? yes 、 no
//
// Exhaustive Cartesian coverage of partitions.
6. public Map<String, Integer> targets(String source)
// Testing strategy for ConcreteVerticesGraph.targets()
//
// Partition the inputs as follows:
// 是否为空图: empty graph? yes 、 no
// 当前的顶点是否在图中: whether vertex is in the graph? yes 、 no
// 顶点是否存在目标顶点: whether vertex has target? yes 、 no
//
// Exhaustive Cartesian coverage of partitions.
7. public String toString()
// Testing strategy for ConcreteVerticesGraph.toString()
//
// Partition the inputs as follows:
// empty graph: yes 、 no
//
// Exhaustive Cartesian coverage of partitions.

```

对 Vertex 类的测试策略如下:

```

1. public boolean isEqualVertex(Vertex vertexAnother)
// Testing strategy for Vertex.isEqualVertex()
//
// Partition the inputs as follows:
// whether vertexAnother is equal to Vertex
//
// Exhaustive Cartesian coverage of partitions.
2. public boolean isEqualName(String vertexAnother)
// Testing strategy for Vertex.isEqualName()

```

```
//
// Partition the inputs as follows:
// whether vertexAnother's name is equal to Vertex
//
// Exhaustive Cartesian coverage of partitions.
3. public void writeTarget(String vertexAnother, Integer weight)
// Testing strategy for Vertex.writeTarget()
//
// Partition the inputs as follows:
// whether vertexAnother is in the TargetEdge
// weight:>0 =0
//
// Exhaustive Cartesian coverage of partitions.
4. public String Name()
// Testing strategy for Vertex.Name()
//
// whether Vertex is null
//
// Exhaustive Cartesian coverage of partitions.
5. public Map<String, Integer> Target()
// Testing strategy for Vertex.Target()
//
// Partition the inputs as follows:
// wherger TargetEdge is null
//
// Exhaustive Cartesian coverage of partitions.
6. public Integer weight(String vertexAnother)
// Testing strategy for Vertex.weight()
//
// Partition the inputs as follows:
// whether vertexAnother is in the TargetEdge
//
// Exhaustive Cartesian coverage of partitions.
7. public boolean remove(String vertexTarget)
// Testing strategy for Vertex.remove()
//
// Partition the inputs as follows:
// vertexTarget is in the TargetEdge or not
//
// Exhaustive Cartesian coverage of partitions.
8. public String toString()
// Testing strategy for Vertex.toString()
//
// Partition the inputs as follows:
```

```
// vertex is null or not
```

```
//
```

```
// Exhaustive Cartesian coverage of partitions.
```

#### 4> 测试结果

可以看到测试全部通过，且代码覆盖率为 100%:

元素	类(%)	方法(%)	行(%)
ConcreteEdgesGraph	0% (0/1)	0% (0/10)	0% (0/68)
ConcreteVerticesGraph	100% (1/1)	100% (10/...	100% (95/...
Edge	0% (0/1)	0% (0/7)	0% (0/19)
Graph	0% (0/1)	0% (0/1)	0% (0/1)
Vertex	100% (1/1)	100% (11/...	100% (59/...

✓ 测试 已通过: 62 共 62 个测试 - 8 ms

C:\Users\Alienware\.jdk\corretto-1.8.0\_292\bin\java.exe ...

ConcreteVerticesGraphTest	
✓ testWriteTargetNotInGraphNoWeight	0 ms
✓ testAddNotEmptyGraphWithoutVertex	0 ms
✓ testRemoveHasVertexTarget	0 ms
✓ testTargetEdgeNull	0 ms
✓ testToString	0 ms
✓ testIsEqualsVertexNullVertexAnother	0 ms
✓ testWeightNotInTargetEdge	0 ms
✓ testSetWithVertexNoWeightWithoutTarget	0 ms
✓ testAddNotEmptyGraphWithVertex	0 ms
✓ testSetWithoutVertexNoWeightWithTarget	0 ms
✓ testSourceWithVertexHasSource	0 ms
✓ testWriteTargetInGraphNoWeight	0 ms
✓ testSetWithVertexHasWeightWithoutTarget	1 ms
✓ testToStringGraph	0 ms
✓ testRemoveGraphWithoutThisVertex	0 ms
✓ testTargetsEmptyGraph	0 ms
✓ testRemoveEmptyGraphWithoutThisVertex	0 ms
✓ testSetWithoutVertexHasWeightWithoutTarget	0 ms
✓ testWriteTargetInGraphWeight	4 ms
✓ testWeightInTargetEdge	0 ms
✓ testTargetEdgeNotNull	0 ms
✓ testVerticesGraph	0 ms
✓ testSetWithoutVertexHasWeightWithTarget	0 ms
✓ testSetWithVertexNoWeightWithTarget	0 ms
✓ testIsEqualsVertexNotEqualAnother	0 ms
✓ testSetWithVertexHasWeightWithTarget	0 ms
✓ testVerticesEmptyGraph	0 ms
✓ testSetWithoutVertexNoWeightWithoutTarget	0 ms
✓ testIsEqualsVertexEqualAnother	0 ms
✓ testSourceWithVertexHasnotSource	1 ms
✓ testRemoveHasNotVertexTarget	0 ms
✓ testToStringEmptyGraph	0 ms
✓ testIsEqualsVertexVertexAnother	0 ms
✓ testNameNotNull	0 ms
✓ testSourceEmptyGraph	0 ms
✓ testIsEqualsNameNullVertexAnother	0 ms
✓ testRemoveGraphWithThisVertex	0 ms
✓ testTargetsHasVertexHasnotTarget	0 ms
✓ testWriteTargetNotInGraphWeight	0 ms
✓ testAddEmptyGraphWithoutVertex	0 ms
✓ testTargetsHasVertexHasTarget	0 ms
✓ testSourcehasVerticesnoSource	0 ms

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

#### 3.1.4.1 Make the implementations generic

此时具体类的声明如下:

```
public class ConcreteEdgesGraph<L> implements Graph<L> {
```

```
class Edge<L> {
```

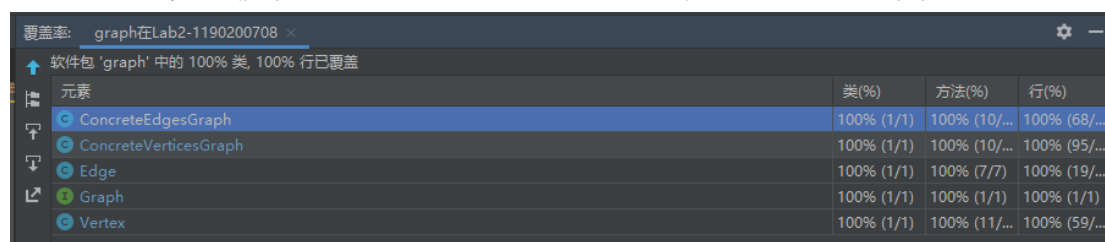
```
public class ConcreteVerticesGraph<L> implements Graph<L> {
```

```
class Vertex <L>{
```

并对 problem2 部分实现的功能修改, 使其支持泛型。

将 ConcreteEdgesGraph 及 ConcreteVerticesGraph 中部分用 String 声明的变量修改为 L, 并且在声明类的时候, 使用泛型 L。

实现后具体类的修改后, 对其测试, 发现测试全部通过。且覆盖率仍为 100%

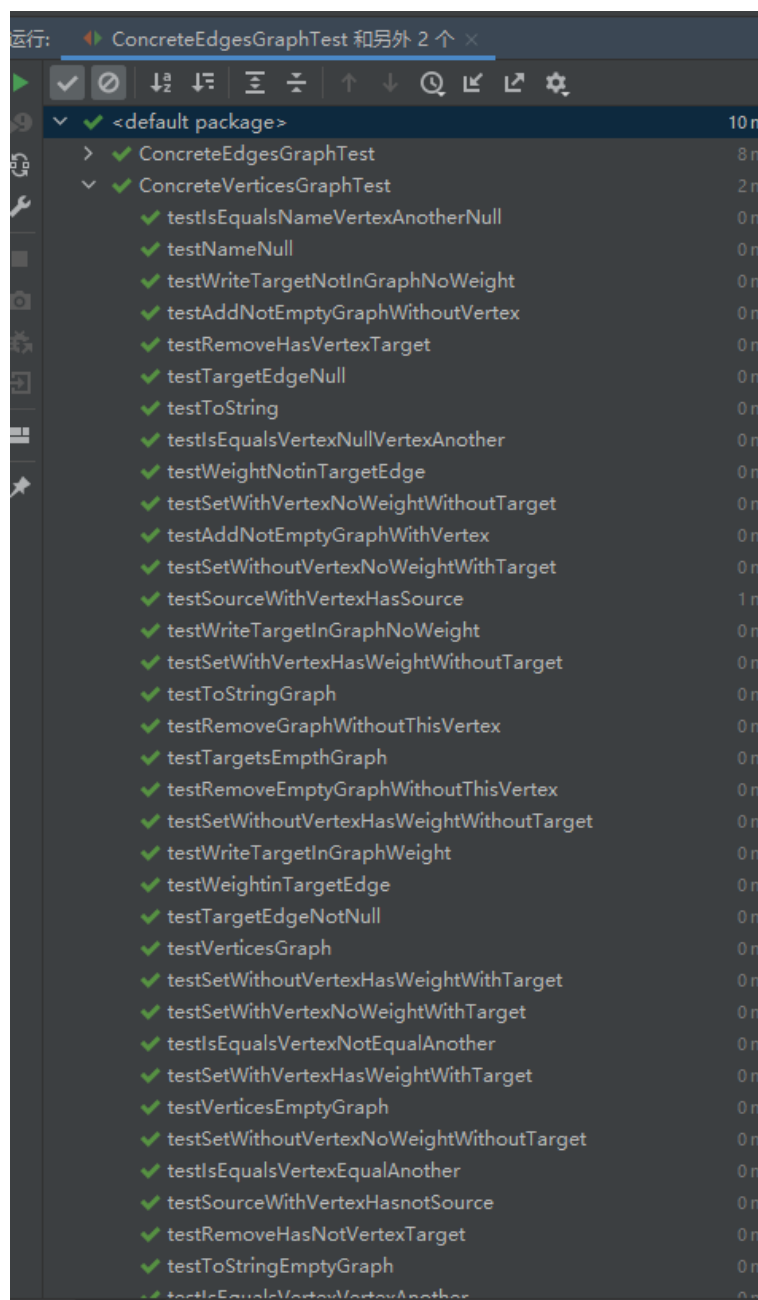


覆盖率: graph在Lab2-1190200708

软件包 'graph' 中的 100% 类, 100% 行已覆盖

元素	类(%)	方法(%)	行(%)
ConcreteEdgesGraph	100% (1/1)	100% (10/...	100% (68/...
ConcreteVerticesGraph	100% (1/1)	100% (10/...	100% (95/...
Edge	100% (1/1)	100% (7/7)	100% (19/...
Graph	100% (1/1)	100% (1/1)	100% (1/1)
Vertex	100% (1/1)	100% (11/...	100% (59/...

```
✓ 测试 已通过: 105共 105 个测试 - 19 ms  
C:\Users\Alienware\.jdk\corretto-1.8.0_292\bin\java.exe .
```



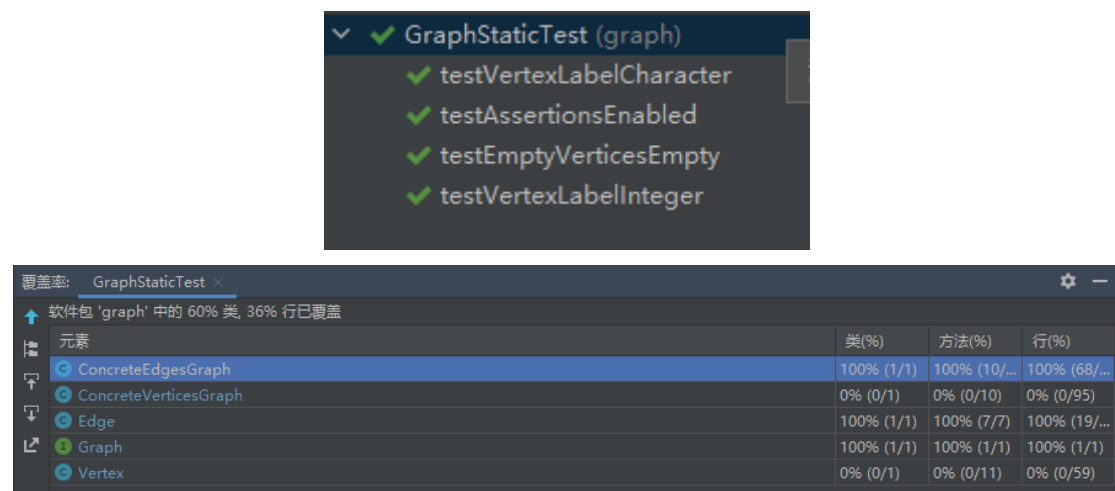
### 3.1.4.2 Implement Graph.empty()

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

此处选择 ConcreteEdgesGraph 或 ConcreteVerticesGraph 中的一个。  
为了确保支持不同的类型，分别测试两种不同的类型 Integer、Charater。  
分别对不同的类型对实现的方法测试：

```
// test vertex label is Integer
@Test
public void testVertexLabelInteger() {
    final Graph<Integer> graph = Graph.empty();
    assertTrue(graph.add(0));
    assertTrue(graph.add(1));
    assertTrue(graph.add(2));
    assertEquals( expected: 0, graph.set( source: 0, target: 1, weight: 1));
    assertEquals( expected: 0, graph.set( source: 1, target: 2, weight: 2));
    assertFalse(graph.remove( vertex: 3));
    Set<Integer> set = new HashSet<Integer>();
    set.add(0);
    set.add(1);
    set.add(2);
    assertTrue(set.equals(graph.vertices()));
    Map<Integer,Integer> mapTarget = new HashMap<>();
    mapTarget.put(1,1);
    assertTrue(mapTarget.equals(graph.targets( source: 0)));
    Map<Integer,Integer> mapSource = new HashMap<>();
    mapSource.put(0,1);
    assertTrue(mapSource.equals(graph.sources( target: 1)));
}
```

```
// test vertex label is Character
@Test
public void testVertexLabelCharacter() {
    final Graph<Character> graph = Graph.empty();
    assertTrue(graph.add('a'));
    assertTrue(graph.add('b'));
    assertTrue(graph.add('c'));
    assertEquals( expected: 0, graph.set( source: 'a', target: 'b', weight: 1));
    assertEquals( expected: 0, graph.set( source: 'b', target: 'c', weight: 2));
    assertFalse(graph.remove( vertex: 'd'));
    Set<Character> set = new HashSet<Character>();
    set.add('a');
    set.add('b');
    set.add('c');
    assertTrue(set.equals(graph.vertices()));
    Map<Character,Integer> mapTarget = new HashMap<>();
    mapTarget.put('b',1);
    assertTrue(mapTarget.equals(graph.targets( source: 'a')));
    Map<Character,Integer> mapSource = new HashMap<>();
    mapSource.put('a',1);
    assertTrue(mapSource.equals(graph.sources( target: 'b')));
}
```



如图，此时测试覆盖率为 100%。

### 3.1.5 Problem 4: Poetic walks

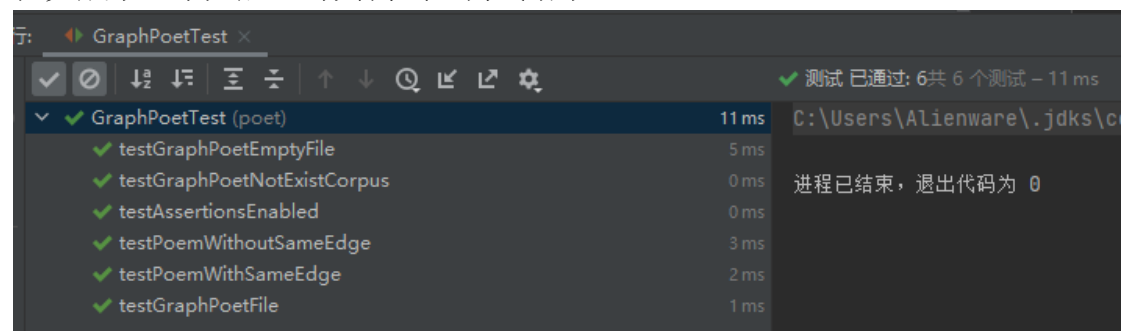
#### 3.1.5.1 Test GraphPoet

对 GraphPoet 类的测试如下：

1. public GraphPoet(File corpus)  
 // Testing strategy for the constructor of GraphPoet  
 // Testing strategy for GraphPoet.toString()  
 // Partition the inputs as follows:  
 // whether the corpus is not exist  
 // whether the corpus is a empty file  
 // Exhaustive Cartesian coverage of partitions.
2. public String poem(String input)  
 // Testing strategy for GraphPoet.poem()  
 // Partition the inputs as follows:  
 // Whether the input has the same edge  
 // Exhaustive Cartesian coverage of partitions.
3. public String toString()  
 // Testing strategy for the constructor of GraphPoet  
 // Testing strategy for GraphPoet.toString()  
 // Partition the inputs as follows:  
 // whether the corpus is not exist  
 // whether the corpus is a empty file



在完成下一单元后，对其测试，测试结果：



可以看到，此时覆盖率为百分值百：

软件包 poet 中的 50% 类, 95% 行已覆盖

元素	类(%)	方法(%)	行(%)
GraphPoet	100% (1/1)	100% (5/5)	100% (56/...
Main	0% (0/1)	0% (0/1)	0% (0/4)

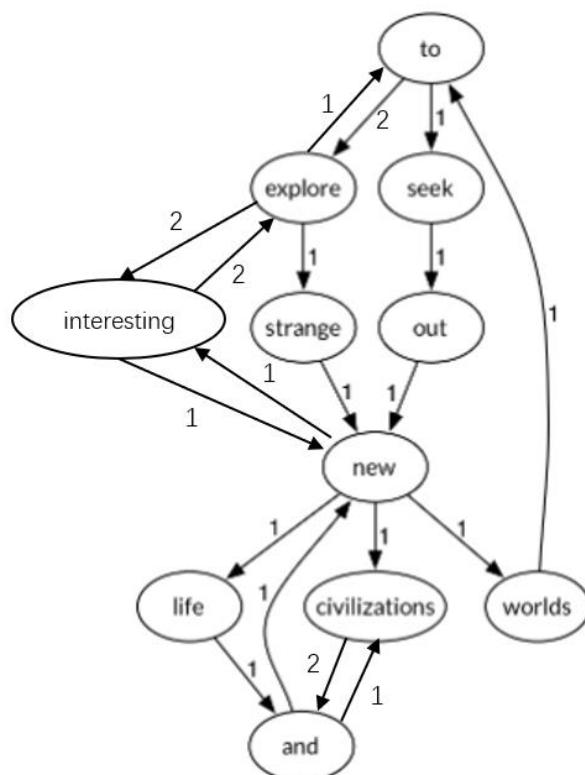
对部分测试策略分析：

当输入文件为下方文件时：

To explore to explore interesting new interesting explore interesting explore  
strange new worlds

To seek out new life and new civilizations and civilizations and

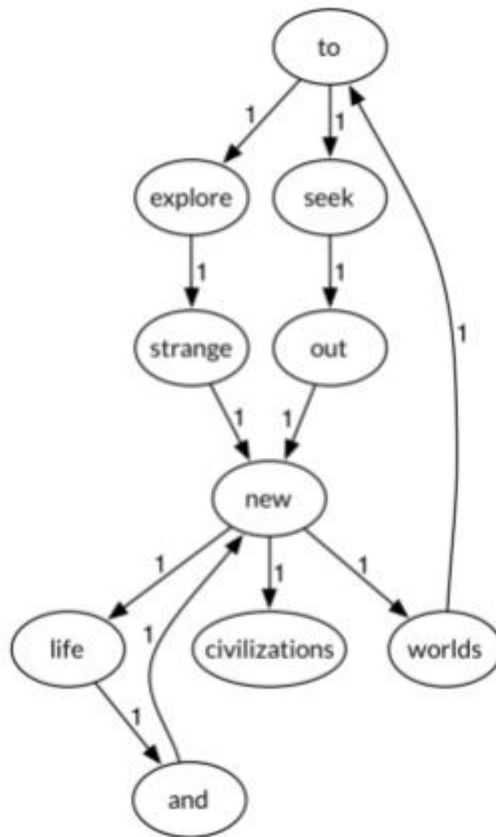
此时 graph 如图所示：



当我们的输入语句为 Seek to explore new and exciting synergies!时，此时对应的语句应该为 Seek to explore interesting new civilizations and exciting synergies!

当此时输入文件为下方字符时：

To explore strange new worlds  
To seek out new life and new civilizations  
此时 graph 如图所示：



当我们的输入语句为 Seek to explore new and exciting synergies!时，此时对应的语句应该为 Seek to explore strange new life and exciting synergies!

### 3.1.5.2 Implement GraphPoet

#### 1. public GraphPoet(File corpus)

首先是文件的读取，采用 `System.getProperty("user.dir")` 的函数读取当前工作区，并加上路径，读取出所需的文件。

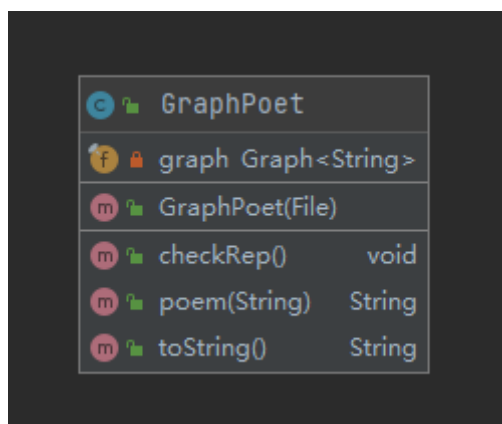
再通过 `BufferedReader`、`FileReader` 将文件读取到一个字符串中，再通过 `split` 函数，将所有空格消除，并读取到 `List` 中，再通过 `List` 中的 `removeAll` 函数，将所有的空字符串删除，此时即可得到文件中所有的单词，并保存到一个列表中。

依次对所有的单词遍历，若前一个节点及后一个节点都在图中，且 `weight++`，否则令 `weight=1`。

#### 2. public String poem(String input)

首先将 `input` 的输入通过 `split` 函数，切分为每个单词，并通过与 `GraphPoet` 相似的方法，获得单词的列表，此时我们用 `List` 中的顺序，每次取出第 `i` 个，及第 `i+1`，如果第 `i` 个单词及第 `i+1` 个单词之间在图中存在别的单词连接，则将这个单词放到 `StringBuilder` 中。直到结束后，将最后一个单词也放进，并加上感叹号。

具体结构如下：



### 3.1.5.3 Graph poetry slam

```
public class Main {  
    /**  
     * Generate example poetry.  
     *  
     * @param args unused  
     * @throws IOException if a poet corpus file cannot be found or read  
     */  
    public static void main(String[] args) throws IOException {  
        final GraphPoet nimoy = new GraphPoet(new File( pathname: "/mugar-omni-theater.txt"));  
        final String input = "Test the system.";  
  
        System.out.println(input + "\n>>>\n" + nimoy.poem(input));  
    }  
}
```

对于如图所示的 main 函数，由于输入的文件为：

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

因此在 test 与 the 中间存在 of 单词，因此输出结果应该为：Test of the system!

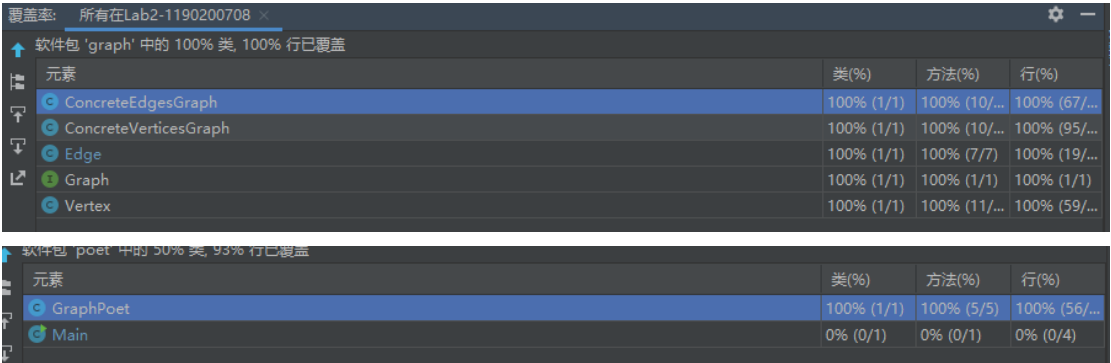
通过测试，所得结果与预期一致，完成预期要求。

```
Main x  
C:\Users\Alienware\.jdk\corret  
Test the system.  
>>>  
Test of the system!
```

### 3.1.6 使用 Eclemma 检查测试的代码覆盖率

由于 IDE 采用 IntelliJ IDEA 2020.3.1，此处使用 IDEA 内置的使用覆盖率测

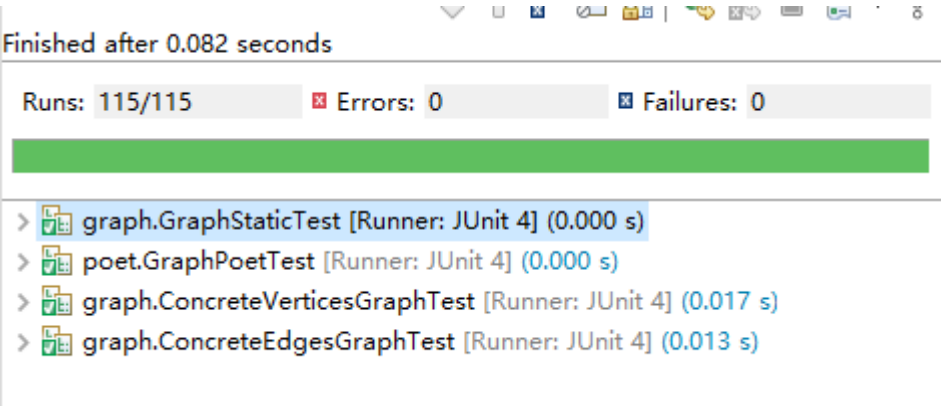
试：



可以看到此时覆盖率为 100%。

src/P1	95.0 %	1,439	75	1,514
graph	96.2 %	1,125	44	1,169
poet	91.0 %	314	31	345

将工作区复制到 Ecilipse 中，用 Eclemma 检测覆盖率，此时覆盖率仍然大约为 95%左右。



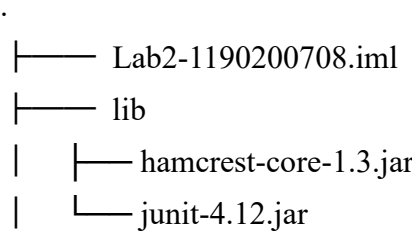
3.1.7 Before you’re done

通过 Git 提交当前版本到 GitHub 上你的 Lab2 仓库。

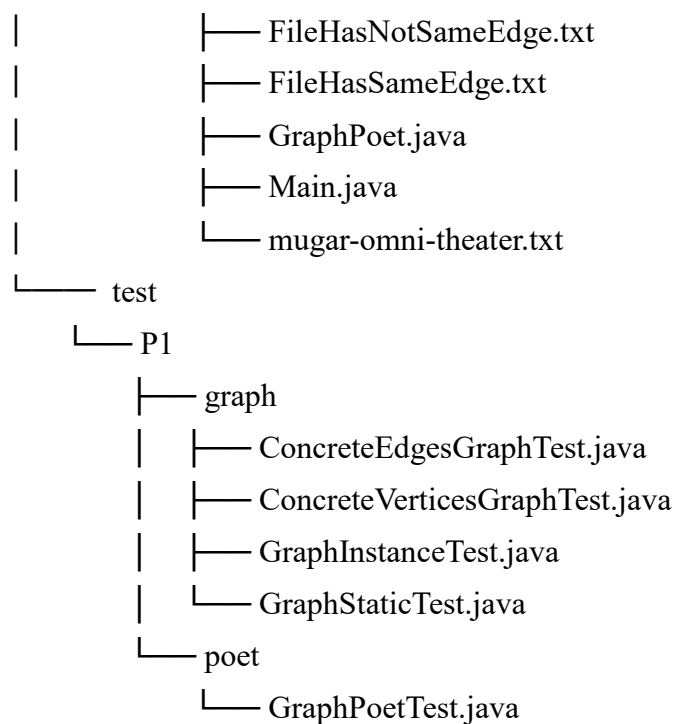
```
git add .
git commit -m "P1"
git push
```

项目的目录结构树状示意图如下方所示：

此处采用 linux 中的 tree 命令：



```
├── out
│   ├── production
│   │   └── Lab2-1190200708
│   │       ├── graph
│   │       │   ├── ConcreteEdgesGraph.class
│   │       │   ├── ConcreteVerticesGraph.class
│   │       │   ├── Edge.class
│   │       │   ├── Graph.class
│   │       │   └── Vertex.class
│   │       └── poet
│   │           ├── EmptyFile.txt
│   │           ├── File1.txt
│   │           ├── FileHasNotSameEdge.txt
│   │           ├── FileHasSameEdge.txt
│   │           ├── GraphPoet.class
│   │           ├── Main.class
│   │           └── mugar-omni-theater.txt
│   └── test
│       └── Lab2-1190200708
│           ├── graph
│           │   ├── ConcreteEdgesGraphTest.class
│           │   ├── ConcreteVerticesGraphTest.class
│           │   ├── GraphInstanceTest.class
│           │   └── GraphStaticTest.class
│           └── poet
│               └── GraphPoetTest.class
├── src
│   └── P1
│       ├── graph
│       │   ├── ConcreteEdgesGraph.java
│       │   ├── ConcreteVerticesGraph.java
│       │   └── Graph.java
│       └── poet
│           ├── EmptyFile.txt
│           └── File1.txt
```



## 3.2 Re-implement the Social Network in Lab1

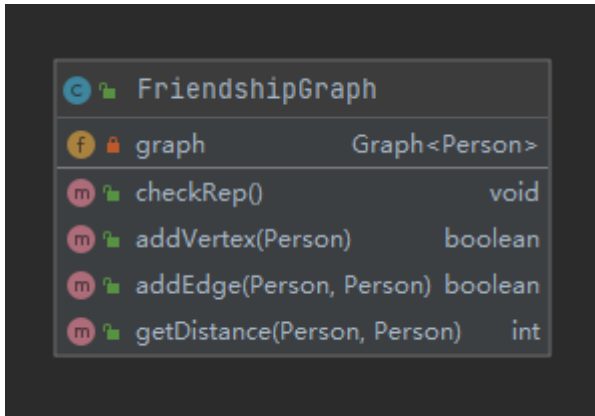
这部分通过以及实现好的 `ConcreteVerticesGraph` 类，来实现新的社交图。

### 3.2.1 FriendshipGraph 类

1. `FriendshipGraph` 类有一个私有变量：  
`private Graph<Person> graph = Graph.empty();`
2. `FriendshipGraph` 类的方法如下：

<code>public void checkRep()</code>	检查不变量
<code>public boolean addVertex(Person person)</code>	添加顶点
<code>public boolean addEdge(Person person1, Person person2)</code>	添加边
<code>public int getDistance(Person person1, Person person2)</code>	返回从 person1 到 person2 的距离

关系图如下：



// Abstraction function:

// 由 Graph<Person> graph 对应的图

// Representation invariant:

// graph 中所有边的 weight=0 或=1，且顶点名字不为空

// Safety from rep exposure:

// 使用 private、final 修饰的变量

// 防御性复制

checkRep 如下：

```
public void checkRep()
{
    for(Person person:graph.vertices())
    {
        assert !person.getName().isEmpty();
        for(Integer weight :graph.targets(person).values())
        {
            assert weight==0 || weight==1;
        }
    }
}
```

getDistance 的实现如下：

1. 将 person1 保存到 HashSet<Person> Search 中；
2. 将 person1 认识的人保存到 HashSet<Person> Search\_Next 中；
3. 将 Search 和 Search\_Next 都保存到新的 HashSet<Person> Search\_assist 中；
4. 初始化 distance = 1；
5. 若 Search\_Next 中含有 person2，返回 distance，即为所求距离。否则将 Search 清空，将 Search\_Next 保存到 Search 中，再将 Search 中所有的人认识的人的 HashSet 中所有的元素添加到 Search\_Next，再将 Search\_Next 中所有在 Search\_assist 中保存的元素消除掉，再将所有 Search\_Next 元素添加到 Search\_assist 中。此时 distance++；
6. 重复步骤 5，若 Search\_Next 最终为空，则不存在距离，返回值为-1；否则，person1 与 person2 之间存在路径，且 person1 与 person2 距离为 distance。

### 3.2.2 Person 类

1. Person 类有一个私有变量：

private final String name;

2. FriendshipGraph 类的方法如下：

public Person(String personName)	构造函数
public void checkRep()	检查不变量
public String getName()	返回当前 Person 的姓名

// Abstraction function:  
// 由 String name 对应的 Person  
// Representation invariant:  
// name 不为空  
// Safety from rep exposure:  
// 使用 private、final 修饰的变量  
// 防御性复制

checkRep 如下：

```
/**
 * 检查不变量
 */
private void checkRep(){
    assert !name.isEmpty();
}
```

### 3.2.3 客户端 main()

此处将重新利用 Lab1 中 main 客户端，由于对 addVertex、addEdge、getDistance 该函数均已完成，因此此处重新利用即可。

运行发现，与预期结果一致。



```
public static void main(String[] args) {
    FriendshipGraph graph = new FriendshipGraph();
    Person rachel = new Person( personName: "Rachel");
    Person ross = new Person( personName: "Ross");
    Person ben = new Person( personName: "Ben");
    Person kramer = new Person( personName: "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
}
}
```

FriendshipGraph x

C:\Users\Alienware\.jdk\corretto-1.8.0\_292\bin\java.exe ...

1  
2  
0  
-1

### 3.2.4 测试用例

对于 FriendshipGraph 类：

#### 1. 测试策略：

- 1.1 public boolean addVertex(Person person)  
// Testing strategy for GraphPoet.addVertex()  
//  
// Partition the inputs as follows:  
// Whether the graph is empty  
//  
// Exhaustive Cartesian coverage of partitions.
- 1.2 public boolean addEdge(Person person1, Person person2)  
// Testing strategy for GraphPoet.addVertex()  
//  
// Partition the inputs as follows:  
// Whether the edge is in the graph  
//  
// Exhaustive Cartesian coverage of partitions.
- 1.3 public int getDistance(Person person1, Person person2)

```
// Testing strategy for GraphPoet.getDistance()  
//  
// Partition the inputs as follows:  
// Is it possible to go from A to B  
//  
// Exhaustive Cartesian coverage of partitions.
```

## 2. 测试结果

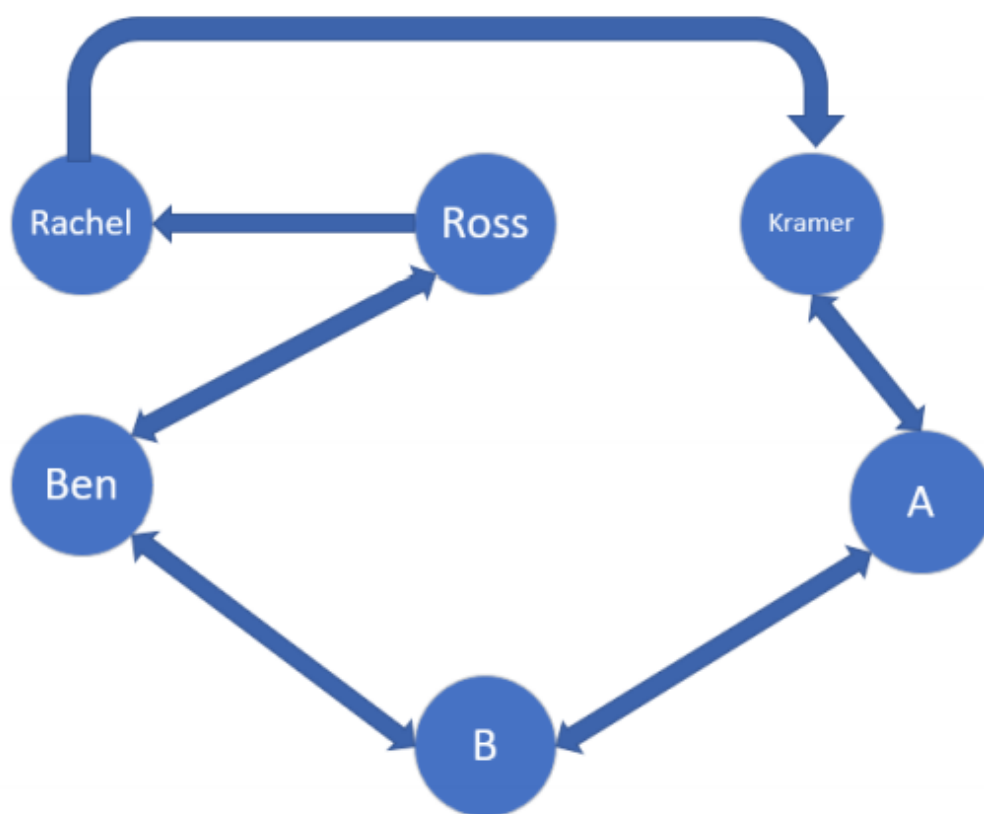
此处主要说明对 `getDistance` 的测试，其余部分详见代码。

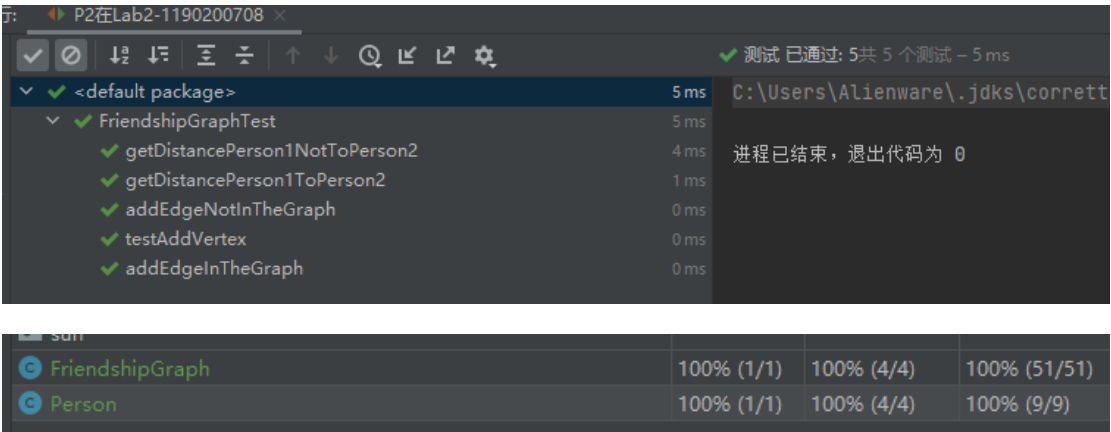
通过以下的图，对 `getDistance` 测试：

由图可得，ben 与 a 的距离为 2，ben 与 kramer 的距离为 3，b 与 kramer 的距离为 2，kramer 与 rachel 的距离为 5。

并进行测试：

```
assertEquals( expected: 2, graph.getDistance(ben, a));  
assertEquals( expected: 3, graph.getDistance(ben, kramer));  
assertEquals( expected: 2, graph.getDistance(b, kramer));  
assertEquals( expected: 5, graph.getDistance(kramer, rachel));
```



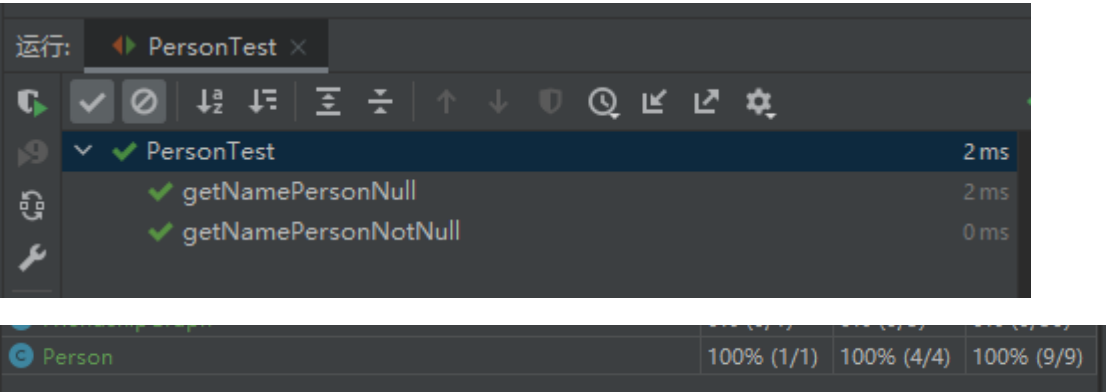


可以发现，此时测试全部通过测试，并且代码覆盖率为 100%。

对于 Person 类：

- 1. 测试策略
  - // Testing strategy for Person.getName
  - //
  - // Partition the inputs as follows:
  - // whether Person is null
  - //
  - // Exhaustive Cartesian coverage of partitions.

- 2. 测试结果
- 测试全部通过，并且代码覆盖率 100%。



### 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。

在这里给出你的项目的目录结构树状示意图。

```
git add .
git commit -m "P2"
git push
```

目录结构树状示意图如下所示：

此处采用 linux 中的 tree 命令：

```
.
├── Lab2-1190200708.iml
├── lib
│   ├── hamcrest-core-1.3.jar
│   └── junit-4.12.jar
├── out
│   ├── production
│   │   └── Lab2-1190200708
│   │       ├── FriendshipGraph.class
│   │       ├── Person.class
│   │       └── graph
│   │           ├── ConcreteEdgesGraph.class
│   │           ├── ConcreteVerticesGraph.class
│   │           ├── Edge.class
│   │           ├── Graph.class
│   │           └── Vertex.class
│   └── poet
│       ├── EmptyFile.txt
│       ├── File1.txt
│       ├── FileHasNotSameEdge.txt
│       ├── FileHasSameEdge.txt
│       ├── GraphPoet.class
│       ├── Main.class
│       └── mugar-omni-theater.txt
└── test
    └── Lab2-1190200708
        ├── FriendshipGraphTest.class
        ├── PersonTest.class
        └── graph
```

```

|           |   |—— ConcreteEdgesGraphTest.class
|           |   |—— ConcreteVerticesGraphTest.class
|           |   |—— GraphInstanceTest.class
|           |   |—— GraphStaticTest.class
|           |   |—— poet
|           |   |—— GraphPoetTest.class
|—— src
|   |—— P1
|   |   |—— graph
|   |   |   |—— ConcreteEdgesGraph.java
|   |   |   |—— ConcreteVerticesGraph.java
|   |   |   |—— Graph.java
|   |   |—— poet
|   |       |—— EmptyFile.txt
|   |       |—— File1.txt
|   |       |—— FileHasNotSameEdge.txt
|   |       |—— FileHasSameEdge.txt
|   |       |—— GraphPoet.java
|   |       |—— Main.java
|   |       |—— mugar-omni-theater.txt
|   |—— P2
|   |   |—— FriendshipGraph.java
|   |   |—— Person.java
|—— test
|   |—— P1
|   |   |—— graph
|   |   |   |—— ConcreteEdgesGraphTest.java
|   |   |   |—— ConcreteVerticesGraphTest.java
|   |   |   |—— GraphInstanceTest.java
|   |   |   |—— GraphStaticTest.java
|   |   |—— poet
|   |       |—— GraphPoetTest.java
|   |—— P2
|   |   |—— FriendshipGraphTest.java
|   |   |—— PersonTest.java

```

## 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
2021.5.27	15:45-17:30	了解实验大致情况并复习相关内容	完成
2021.6.6	13:00-18:00	完成测试部分	完成
2021.6.8	13:00-18:00	完成 problem 2 边部分	完成
2021.6.9	13:00-18:00	完成 problem 2 边部分测试	完成
2021.6.10	13:00-21:00	完成 problem 2 顶点部分	完成
2021.6.11	13:00-21:00	完成 problem 2 点部分测试	完成
2021.6.12	13:00-21:00	完成 P1 及测试	完成
2021.6.12	9:00-17:00	完成 P2 及测试及报告	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
泛型的转换最初理解起来稍 有问难。	通过查阅资料并与同学讨论等途径解决。
对于题目某些部分理解起来 稍有困难。	通过自己耐心探究解决。
测试策略的设计。	通过查阅相关资料并加以体会解决。

## 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

1. 掌握了在测试中，写出测试策略，并根据此设计测试用例的方法
2. 了解了 OOP 实现 ADT，并对 RI、REP、AF 等有了更深的了解
3. 对于整个过程了解更加深刻
4. 掌握了 ADT 设计的基本方法
5. 了解了 ADT 的泛型化
6. 意识到避免表示泄露的重要性，并在实践中，践行了保护的过程

## 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？

在面向 ADT 的编程设计，我们往往要考虑程序的功能、应用场景以及复用性，使编出的程序更加灵活；而面向应用场景编程，适用范围较小，但可以根据具体场景，灵活编程。

(2) 使用泛型和不使用泛型的编程，对你来说有何差异？

使用泛型可以更改数据类型，如 P2 可以以 `Person` 为数据类型，在初次接触时有一些不适应，但很快便了解并掌握。泛型为程序的复用性带来了很大帮助，并且使类型更加安全。

(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？

先完成测试用例的编写能够让我更好的理解规格说明。规格说明也可能存在问题--不正确、不完整、模棱两可、确实边界情况。因此先尝试编写测试用例，可以在我浪费时间实现一个有问题的规格说明之前发现这些问题。因此我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？

避免重复实现具有类似功能的 ADT，将已有代码进行复用，不仅在一定程度上节约了时间，同时还能进一步保证程序的正确性，避免写类似代码时，犯更多的错误。同时还能够锻炼我们的抽象能力，在实现 ADT 时候，充分考虑应用场景。

(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？

实验未要求 P3.

(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？

表示暴露影响程序的安全运行，同时影响到不变性和表示独立性处理表示暴露避免类内部数据被从外部访问。并且在编程时候在抽象类型表示声明后写上对于抽象函数和表示不变量的注解，可以便于读取表示不变量、抽象域的表示、规定合法的表示值如何被解释到抽象域。方便程序员的处理。以后也会坚持这个习惯。

(7) 关于本实验的工作量、难度、`deadline`。

本实验工作量较大，尤其是编写测试时，耗费时间较长，难度可以接受，`deadline` 比较合理。

(8) 《软件构造》课程进展到目前，你对该课程有何体会和建议？

使我对软件构造有了新的感悟，一方面对程序的测试分区等有了新的体会，另一方面，对规范的注释等有了新的了解，另一方面更加体会到了面向对象编程，除此之外，更加了解了如何保证程序的安全运行。

建议，对 PPT 内容扩充，方便同学们复习。