目录

[1. 实验目标概述 1](#_Toc167574118)

[2. 实验环境配置 1](#_Toc167574119)

[3. 实验过程（Poetic Walks） 1](#_Toc167574120)

[3.1 Problem 1: Test Graph <String> 1](#_Toc167574121)

[3. 2 Problem 2: Implement Graph <String> 3](#_Toc167574122)

[3.2.1 Implement ConcreteEdgesGraph 3](#_Toc167574123)

[3.2.2 Implement ConcreteVerticesGraph 5](#_Toc167574124)

[3.3 Problem 3: Implement generic Graph<L> 7](#_Toc167574125)

[3.3.1 Make the implementations generic 7](#_Toc167574126)

[3.3.2 Implement Graph.empty() 7](#_Toc167574127)

[3.4 Problem 4: Poetic walks 7](#_Toc167574128)

[3.4.1 Test GraphPoet 7](#_Toc167574129)

[3.4.2 Implement GraphPoet 8](#_Toc167574130)

[3.4.3 Graph poetry slam 9](#_Toc167574131)

1. 实验目标概述

使用TDD(Test-Driven-Deployment)“测试驱动的开发”实现基于边/点的两种边带权有向图这一抽象数据类型。并以此为基础实现graphPoet并生成poem，也就是基于输入的文件内容生成一张图，再依据此图与给定字符串输入新的字符串。

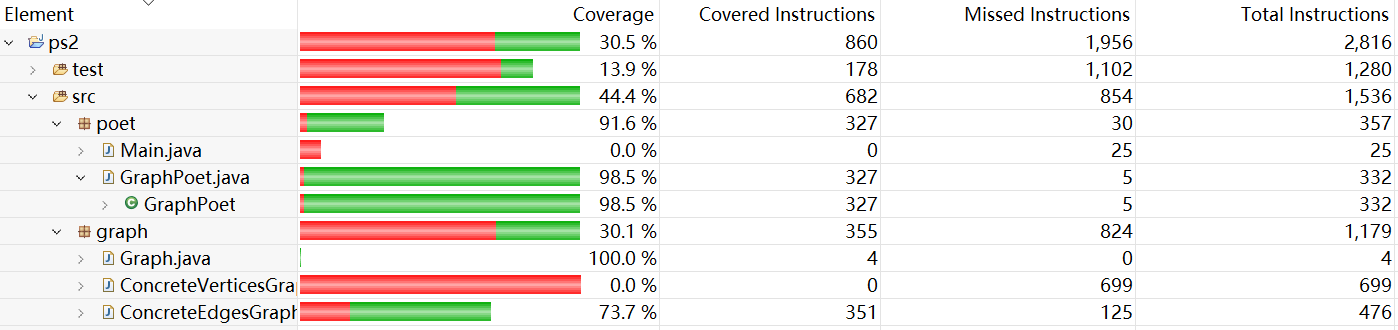
1. 实验环境配置

参与JAVA程序设计课程时安装了JDK17和eclipseIDE

参与软件过程与项目管理课程时安装了Git

所以不需要进行新配置，可以直接写

从Eclipse自带的插件market里安装了插件EclEmma测试代码覆盖率，效果图如下：



1. 实验过程（Poetic Walks）

## 3.1 Problem 1: Test Graph <String>

## 考虑对六个方法逐个分解。

## 设置Situation1和Situation2 两列是部分方法分层分解情况所需。

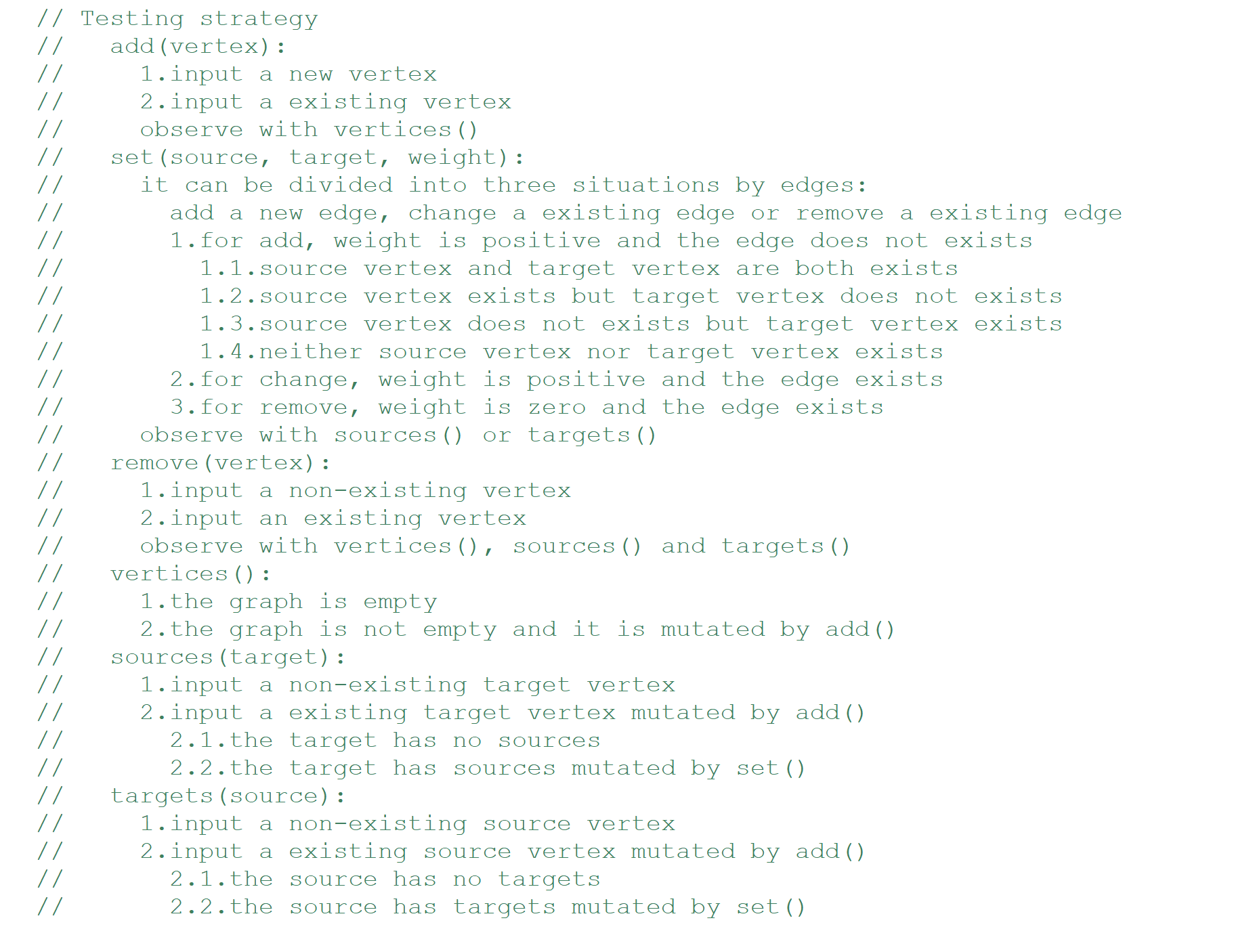
## 对于toString方法：

## 考虑空图与非空图两种情况，分别打印对应的String观察是否符合要求即可。

## 测试策略表:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Method | Input | Situation1 | Operation | Situation2 | Description | Return |
| add | 待插入顶点的标签  vertex | 输入顶点不存在 | Insert | 无 | 插入该顶点 | True |
| 输入顶点已存在 | Nothing | 无 | 不进行任何操作 | False |
| set | 起点  source  终点  target  从起点到终点的边的权值  weight | 输入权值为正并且边不存在 | Add | 起点和终点都存在 | 如果起点或终点不存在，调用add方法添加之  新增一条从起点到终点，权重为给定的weight的边 | 0 |
| 起点存在，终点不存在 |
| 终点存在，起点不存在 |
| 起点和终点都不存在 |
| 输入权值为正并且边已存在 | Change | 无 | 修改此边的权值 | Change之前该边的权值 |
| 输入权值为0 | Remove | 无 | 删除此边 | Remove之前此边的权值 |
| remove | 待删除的顶点标签vertex | 输入顶点已存在 | Remove | 无 | 删除该顶点 | True |
| 输入顶点不存在 | Nothing | 无 | 不进行任何操作 | False |
| vertices | Nothing | 此图是一张空图 | Nothing | 无 | 读取此图的顶点集合，应为一个空集合 | Empty set |
| 此图不是空图 | Nothing | 无 | 读取此图的顶点集合，应为一个非空集合 | Non empty set with all vertices in this graph |
| sources | 终点  target | 输入终点不存在 | Nothing | 无 | 读取此终点的前驱，存入一个映射，此映射应为空 | Empty map |
| 输入终点存在 | Nothing | 有以该点为终点的边 | 读取此终点的前驱，存入一个映射，此映射应为非空 | Non empty map with all predecessors for target in this graph |
| 没有以该点为终点的边 | 读取此终点的前驱，存入一个映射，此映射应为空 | Empty map |
| targets | 起点  source | 输入起点不存在 | Nothing | 无 | 读取此终点的后继，存入一个映射，此映射应为空 | Empty map |
| 输入起点存在 | Nothing | 有从此点为起点的边 | 读取此终点的后继，存入一个映射，此映射应为非空 | Non Empty map with all successors for source in this graph |
| 没有从此点为起点的边 | 读取此终点的后继，存入一个映射，此映射应为空 | Empty map |

代码中书写的测试策略：



据此实现这六个方法

值得注意的是：

set要先调用add添加尚不存在的顶点

遍历列表等结构时，若需要增加/删除其元素，使用迭代器Iterator否则遍历次序会变混乱

不要在多个线程中同时试图修改其结构，否则会抛出异常

## 3. 2 Problem 2: Implement Graph <String>

### 3.2.1 Implement ConcreteEdgesGraph

思路：

设计Edge<String>的测试类 -> 实现Edge<String> -> 实现Graph的方法

字段表如下：

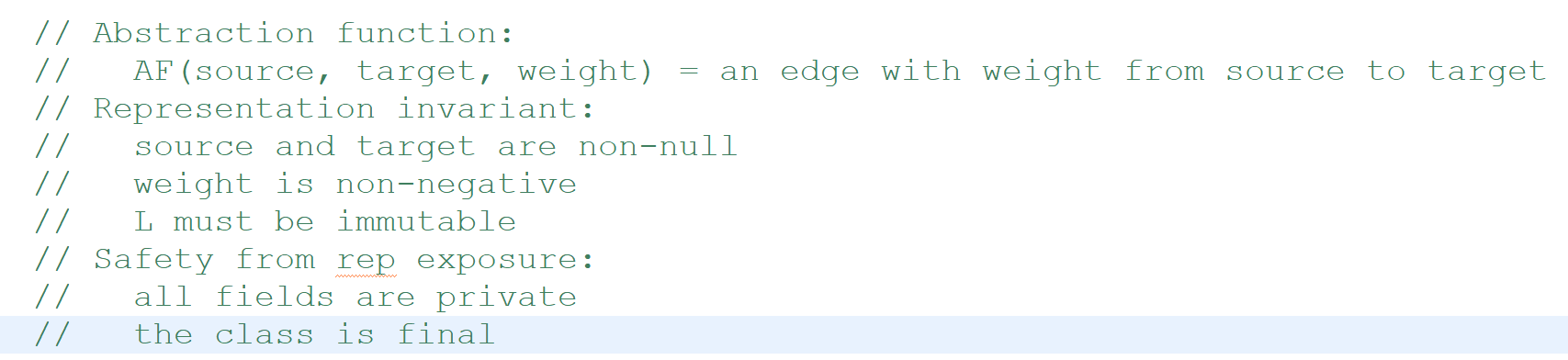
|  |  |  |  |
| --- | --- | --- | --- |
| Modifier | Type | Field | Description |
| private and final | Set<String> | vertices | 顶点集 |
| private and final | List<Edge<String>> | edges | 边集 |

对于Edge<String>

抽象函数AF、表示不变量RI和表示泄露的安全处理如下：

|  |  |
| --- | --- |
|  | Description |
| Abstraction function | 表示空间：起点source，终点target，权值weight  抽象空间：一条权值为weight的从source到target的边 |
| Representation invariant | 起点source和终点target都非空， 权值weight非负 |
| Safety from rep exposure | 所有字段都设定为私有private且不可重新赋值final， Edge类设定不可重新赋值final |

代码中的描述：



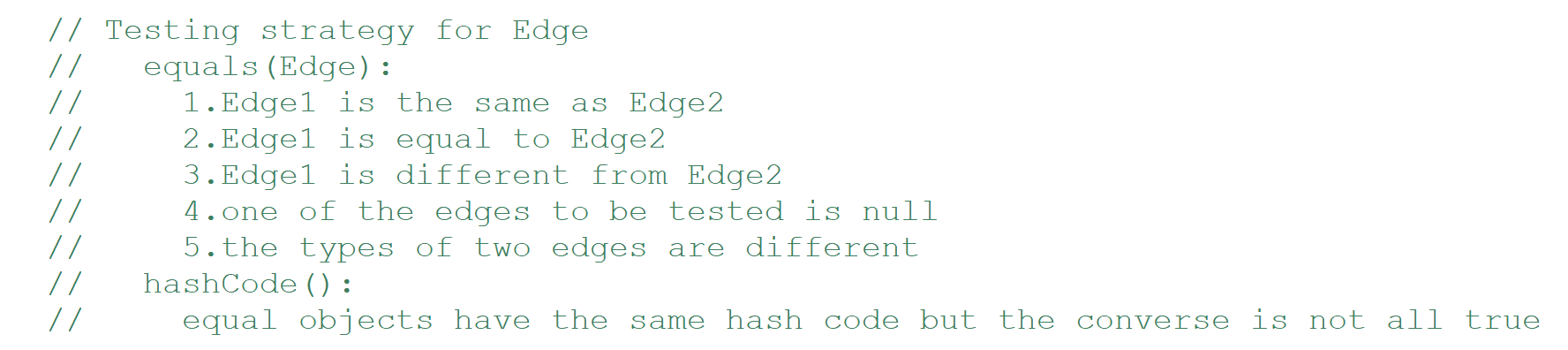
实现checkRep方法检查RI始终成立即可

Edge<String>测试

仅两个方法：equals(Edge<String>)和hashCode()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Input | Situation | Description | Return |
| equals | 待比较对象Object obj | this==obj | 是否指向同一块内存 | True |
| obj==null | 是否为空 | False |
| getClass()!=obj.getClass() | object是否为Edge<String> | False |
| other | 调用Object的equals方法 | 返回  object.equals()的返回值 |
| hashCode | Nothing | 判断两个equals为true的对象的哈希码是否相等 | 相同的对象应有相同的哈希码 | 哈希码的值 |

代码中的描述：



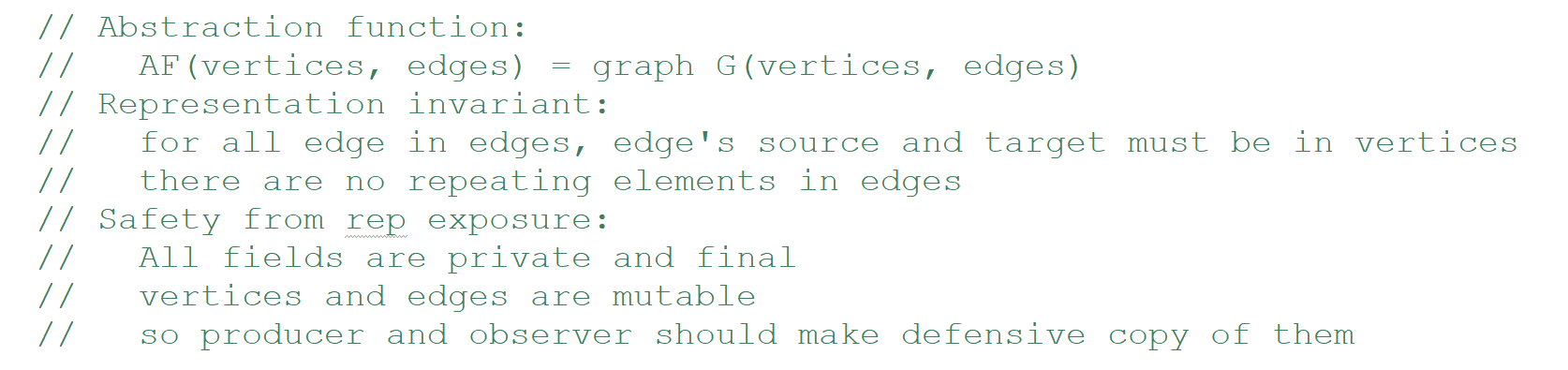
实现这两个方法与带参构造函数即可

对于ConcreteEdgesGraph<String>

抽象函数AF、表示不变量RI和表示泄露的安全处理如下：

|  |  |
| --- | --- |
|  | Description |
| Abstraction function | 表示空间：顶点集vertices,边集edges  抽象空间：一张图G(V,E)，其中V=vertices， E=edges |
| Representation invariant | 边集中任意元素的起点和终点均为此图G的顶点  边集中没有重复元素（集合理应无重复元素，但实际是用List实现，因此有必要进行此检查） |
| Safety from rep exposure | 所有字段都设定为私有private且不可重新赋值final  Observer方法调用可变字段vertices和edges时一律使用防御式拷贝 |

代码中的描述：



据此实现checkRep方法即可

### 3.2.2 Implement ConcreteVerticesGraph

思路：

设计Vertex<String>的测试类 -> 实现Vertex<String> -> 实现Graph的方法

字段表如下：

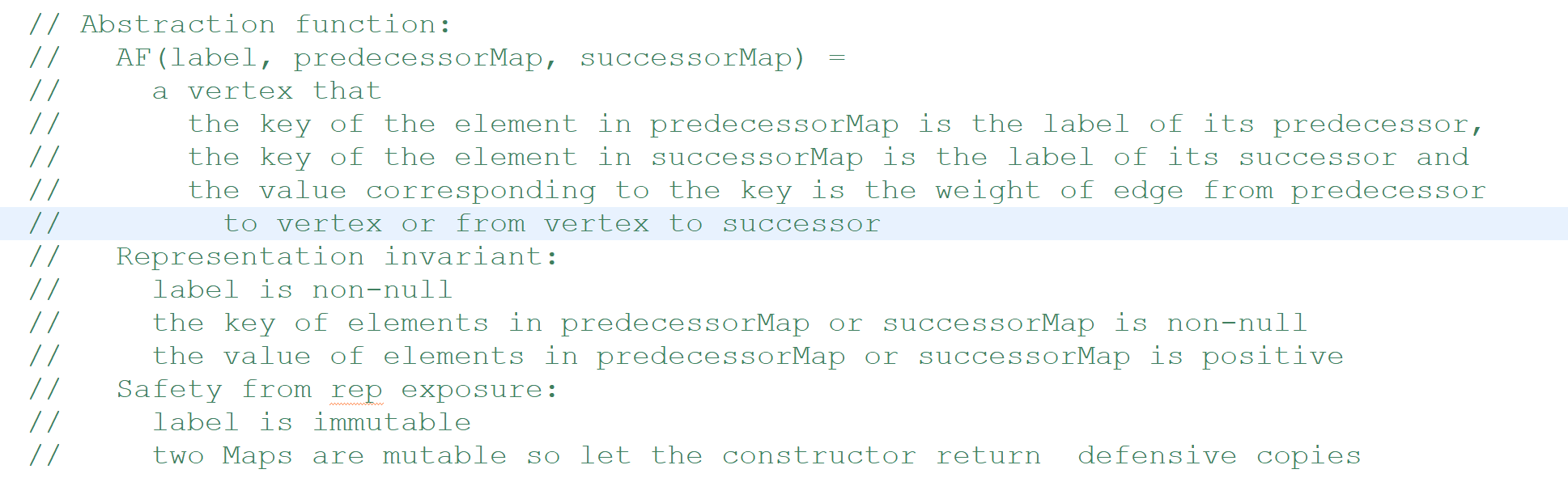
|  |  |  |  |
| --- | --- | --- | --- |
| Modifier | Type | Field | Description |
| private and final | List<Vertex<String>> | vertices | 包含前驱与后继的顶点集 |

对于Vertex<String>

抽象函数AF、表示不变量RI和表示泄露的安全处理如下：

|  |  |
| --- | --- |
|  | Description |
| Abstraction function | 表示空间：标签label，前驱映射predecessorMap，后继映射successorMap  抽象空间：一个顶点，前驱映射里的所有key对应该顶点的所有入边的起点，这条入边的权重是key对应的value，后继映射同理 |
| Representation invariant | 顶点的标签label非空  两个映射的所有key都非空  所有key对应的value都为正 |
| Safety from rep exposure | 所有字段都设定为私有private  构造方法Constructor中对Map采用防御式拷贝 |

代码中的描述：

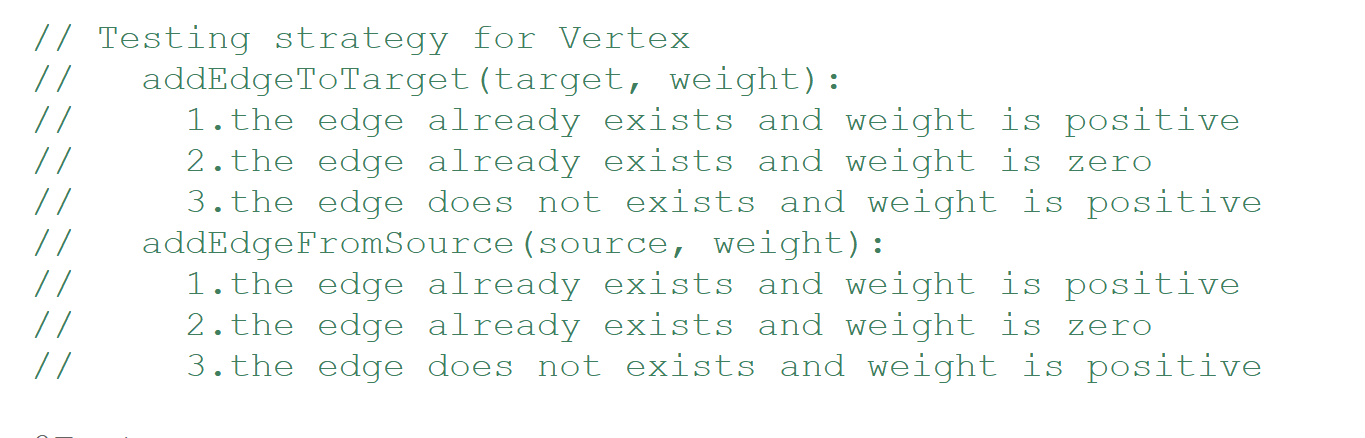


据此实现checkRep方法即可

Vertex<String>测试

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Method | Input | Situation1 | Operation | Situation2 | Description | Return |
| addEdge  From  Source | 起点  source  权值  weight | 从source  到该顶点的边已存在 | change | 输入权值  weight  为正 | 修改从  source  到该顶点的  边的权值 | 修改前  此边的  权值 |
| remove | 输入权值  weight  为0 | 删除  从source  到该顶点的边 | 删除前  此边的  权值 |
| 从source  到该顶点的边不存在 | add | 输入权值  weight  为正 | 新增  从source  到该顶点的边 | 0 |
| addEdgeToTarget | 终点  target  权值  weight | 从该顶点到target的边已存在 | change | 输入权值  weight  为正 | 修改  从该顶点到  target的边的权值 | 修改前  此边的  权值 |
| remove | 输入权值  weight  为0 | 删除  从此顶点  到target的  边 | 删除前  此边的  权值 |
| 从该顶点到target的边不存在 | add | 输入权值  weight  为正 | 新增  从该顶点  到target的  边 | 0 |

代码中的描述：



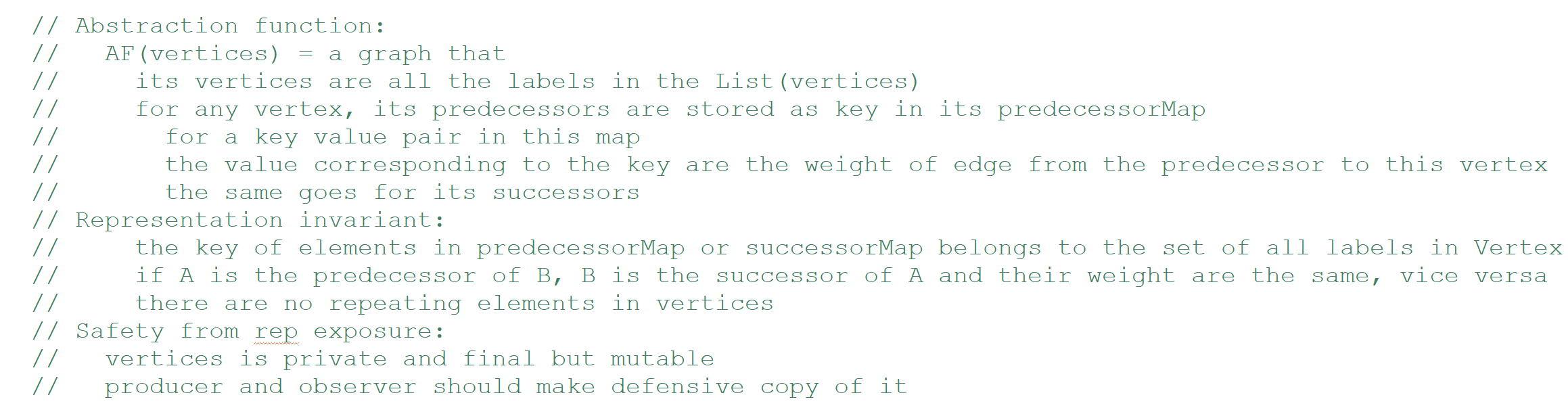
据此实现addEdgeFromSource和addEdgeToTraget方法即可

对于ConcreteVerticesGraph<String>

抽象函数AF、表示不变量RI和表示泄露的安全处理如下：

|  |  |
| --- | --- |
|  | Description |
| Abstraction function | 表示空间：顶点集vertices  抽象空间：一张图G，顶点由vertices中的label构造，  顶点的入边存放在predecessorMap中；  顶点的出边存放在successorMap中。 |
| Representation invariant | 边集中任意元素的起点和终点均为此图G的顶点  边集中没有重复元素（集合理应无重复元素，但实际是用List实现，因此有必要进行此检查）  如果顶点A的successorMap的keySet中含有B，则B的predecessorMap中含有A，反之亦然 |
| Safety from rep exposure | 字段设定为私有private且不可重新赋值final  Observer方法调用可变字段vertices时使用防御式拷贝  Collections.unmodifiableMap |

代码中的描述：



据此实现checkRep方法即可

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

### 3.3.1 Make the implementations generic

将代码中的String替换为泛型L(E，K之类也行，保持使用的符号一致即可)即可

注意考虑适当使用泛型通配符以消除警告，例如：



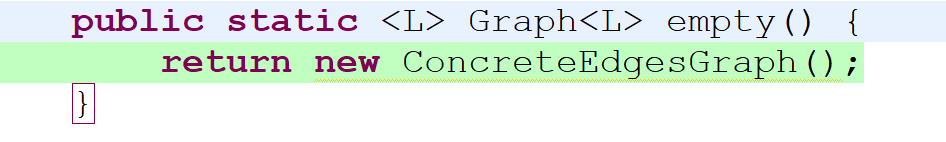
其中？是泛型通配符，可以消除编译器警告

### 3.3.2 Implement Graph.empty()

任选ConcreteVerticesGraph与ConcreteEdgesGraph中的一个

new一个并返回即可

此处选用了ConcreteEdgesGraph：



## 3.4 Problem 4: Poetic walks

### 3.4.1 Test GraphPoet

期望结果：

以任意长度的空白字符分割语料库文本成为许多个word并存入图中

图中顶点均为小写

边A -> B的权值代表A之后跟着几个连续的B

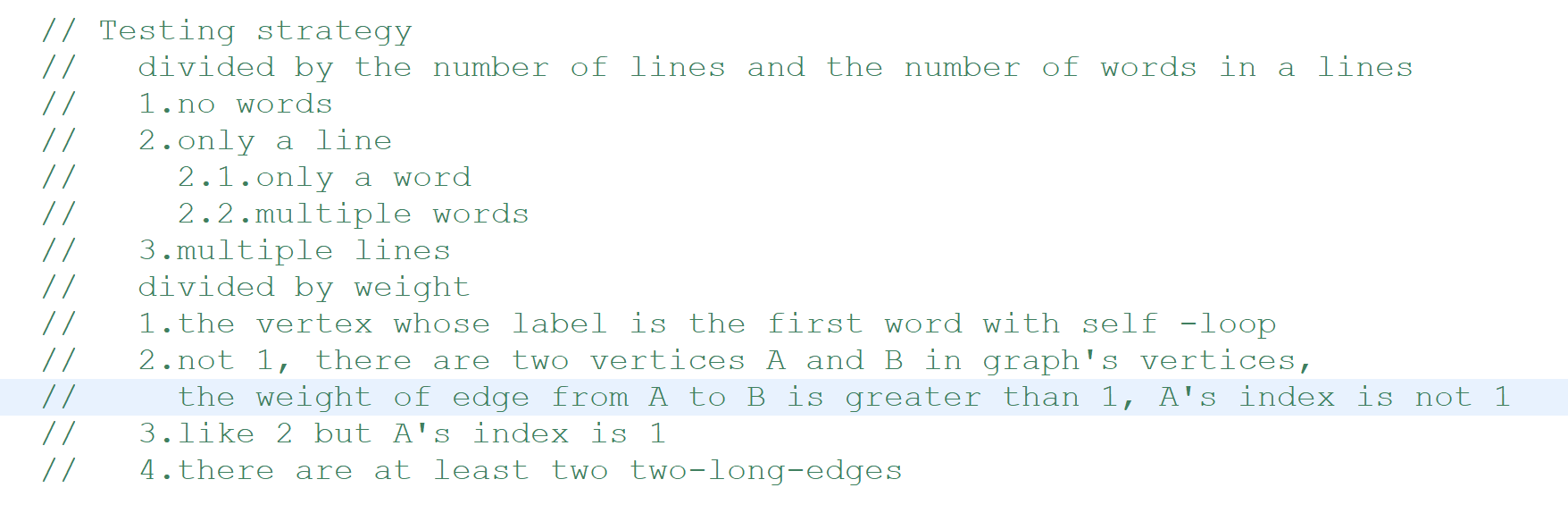
若输入字符串的两个word A和B在图中有中间顶点，即存在顶点W，有两条边组成的路径A -> W -> B，则添加W至A到B中间，若有许多条这样的边，取权最大者

W称为桥接字符且W必为全小写

测试策略表：

|  |  |
| --- | --- |
| Index | Stituation |
| 1 | 语料库中没有单词 |
| 2 | 语料库中仅有一行、有一个单词 |
| 3 | 语料库中仅有一行且一行有多个单词 |
| 4 | 语料库中有许多行 |
| 5 | 语料库中至少最前两个单词相同 |
| 6 | 语料库中的第一个单词和第二个单词不同但第二个单词至少和第三个单词相同 |
| 7 | 语料库中的第一个单词和第二个单词不同且第二个单词和第三个单词不同，但构成的图中仍有权值大于1的边 |
| 8 | 语料库形成的图中至少存在这样一堆顶点A与B，使得至少有两个顶点X和Y满足A -> X -> B和A -> Y -> B |

代码中的描述：



据此实现图的生成算法

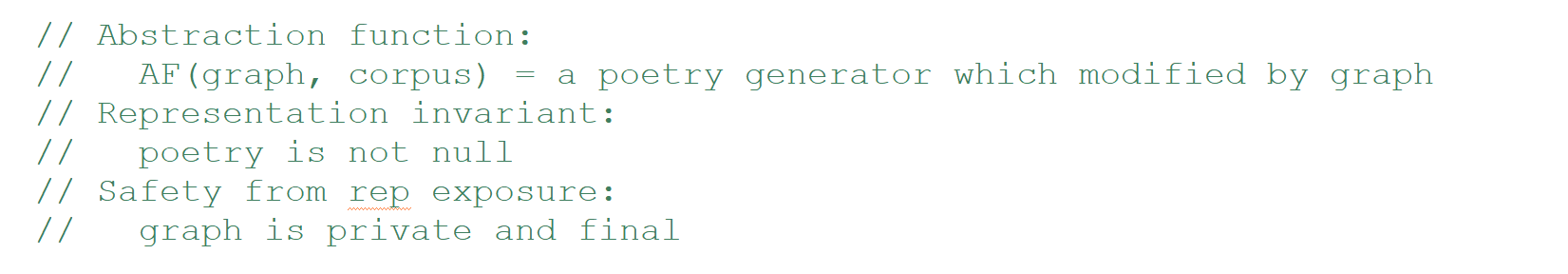
再依据期望结果实现poem方法

### 3.4.2 Implement GraphPoet

抽象函数AF、表示不变量RI和表示泄露的安全处理如下：

|  |  |
| --- | --- |
|  | Description |
| Abstraction function | 表示空间：图graph和语料库文件corpus  抽象空间：一个graphPoetry，其中包含了一张被构造方法修改过的图G |
| Representation invariant | graphPoetry不能为空  其它必要的检查在图的checkRep中已经完善过 |
| Safety from rep exposure | 字段设定为私有private且不可重新赋值final |

代码中的描述：



Poem的具体实现：

1.将输入字符串按照空白拆分为字符数组

2.遍历字符数组

通过当前字符的后继Map.keySet()与下一个字符的前驱Map.keySet()取交集的方式搜索到所有可能的桥接字符集合

遍历桥接字符集合，搜索对应权和最大的桥接字符并将之插入

3.输出

### 3.4.3 Graph poetry slam

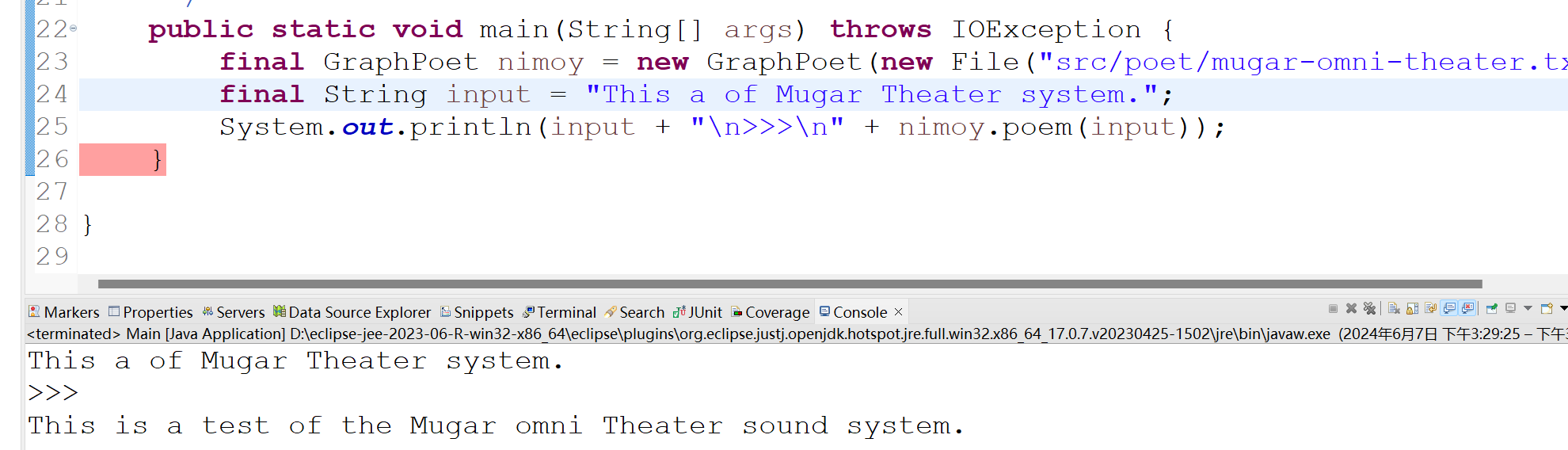
我没有完全理解本任务的要求，没有修改他给出的.txt文件，内容仍是：

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

完全输出这个句子的最小输入应该是

This a of Mugar Theater system.

进行测试：



符合预期