

**2018年春季学期  
计算机学院大二软件构造课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | Arcwoc |
| 学号 | xxxxx |
| 班号 | xxxxxx |
| 电子邮件 | xxxxx |
| 手机号码 | xxxxx |

**目录**

[1 实验目标概述 1](#_Toc507927440)

[2 实验环境配置 1](#_Toc507927441)

[3 实验过程 2](#_Toc507927442)

[3.1 Poetic Walks 2](#_Toc507927443)

[3.1.1 Get the code and prepare Git repository 2](#_Toc507927444)

[3.1.2 Problem 1: Test Graph <String> 3](#_Toc507927445)

[3.1.3 Problem 2: Implement Graph <String> 6](#_Toc507927446)

[3.1.3.1 Implement ConcreteEdgesGraph 6](#_Toc507927447)

[3.1.3.2 Implement ConcreteVerticesGraph 7](#_Toc507927448)

[3.1.4 Problem 3: Implement generic Graph<L> 8](#_Toc507927449)

[3.1.4.1 Make the implementations generic 9](#_Toc507927450)

[3.1.4.2 Implement Graph.empty() 9](#_Toc507927451)

[3.1.5 Problem 4: Poetic walks 10](#_Toc507927452)

[3.1.5.1 Test GraphPoet 10](#_Toc507927453)

[3.1.5.2 Implement GraphPoet 11](#_Toc507927454)

[3.1.5.3 Graph poetry slam 12](#_Toc507927455)

[3.1.6 Before you’re done 12](#_Toc507927456)

[3.2 Re-implement the Social Network in Lab1 13](#_Toc507927457)

[3.2.1 FriendshipGraph类 13](#_Toc507927458)

[3.2.2 Person类 14](#_Toc507927459)

[3.2.3 客户端main() 15](#_Toc507927460)

[3.2.4 测试用例 16](#_Toc507927461)

[3.2.5 提交至Git仓库 17](#_Toc507927462)

[3.3 The Transit Route Planner（选做，额外给分） 17](#_Toc507927463)

[4 实验进度记录 21](#_Toc507927464)

[5 实验过程中遇到的困难与解决途径 21](#_Toc507927465)

[6 实验过程中收获的经验、教训、感想 22](#_Toc507927466)

# 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;

④ADT 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(abstraction function)

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外泄(rep exposure);

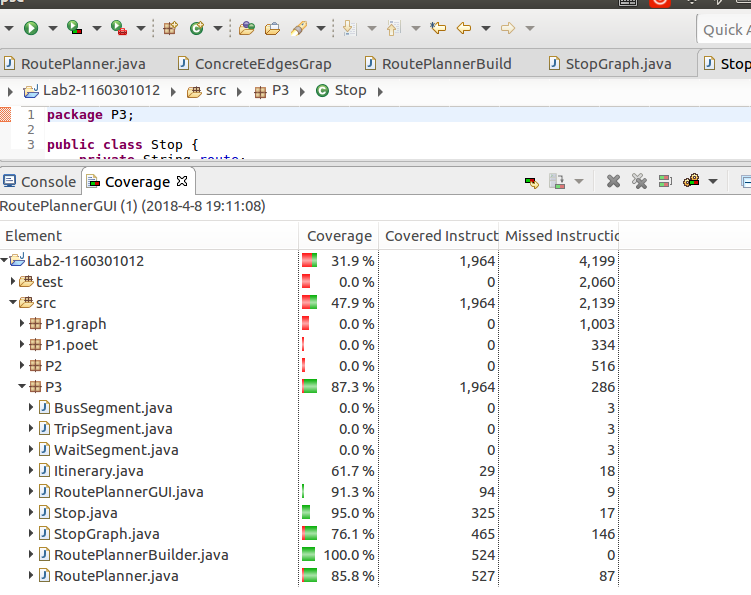
⑦测试 ADT 的实现并评估测试的覆盖度;

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。

# 2 实验环境配置

安装并配置和使用EclEmma，截图如下：



https://github.com/ComputerScienceHIT/Lab2-1160301012

# 实验过程

## Poetic Walks

问题1-3：实现具有标记顶点的Graph<L>，可变加权有向图的抽象数据类型并测试。然后使用图形数据类型，实现一个GraphPoet，一个使用单词的图生成诗歌的类。

### Get the code and prepare Git repository

git clone https://github.com/rainywang/Spring2018\_HITCS\_SC\_Lab2/t

ree/master/P1

### Problem 1: Test Graph <String>

所有的实例方法的测试在GraphInstanceTest.java中，在这些测试中，用emptyInstance()方法来获取新的空图，然后在GraphInstanceTest中对set，add，remove,sources,targets等方法进行测试。

测试策略：

等价类划分法：给图中添加的点：未存在，已存在

给图中添加的边的权重：大于零，等于零

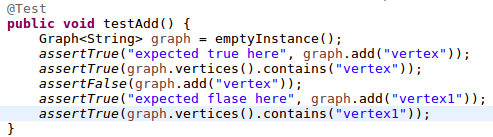
然后在每个类中进行排列组合，根据每个类的需求再设计相应的测试用例。

**①testAdd()**

**add()方法实现要求**：add函数要求向图中加一个新的顶点，如果加入成功则返回true，否则返回false，当该顶点已经存在时返回false，图不改变。

**测试策略**：加入一个之前没有存在的顶点，判断是否返回true，返回true则测试成功，否则失败。然后再加入第一次加入的点，判断是否返回false，返回false则测试成功，否则失败。然后判断之前加入的顶点是否以及存在于顶点集合，如果存在，表明加入成功，否则失败。

**测试代码：**

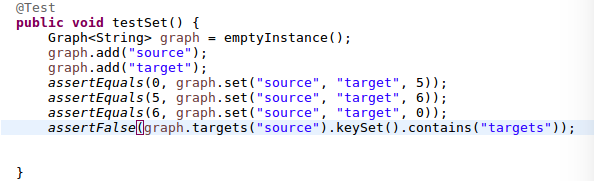
****

**②testSet()**

**set()方法实现要求**：向图中加边，或者改变，删除一个带权有向边。如果权重非零且此边之前不存在，则加入一条新的边。如果权重非零且此边已经存在，则更新旧有的边的权重。如果权重为0并且该边已经存在，则删除该边，否则图不改变。

**测试策略**：向图中加入一条之前不存在的边，测试返回值是否为0，为0则测试成功，否则失败。然后向图中加入第一次加入的边，改变权重，查看返回值是否 是第一次加入的边的权重，相符合则测试成功，否则测试失败。再改变权重为0重新加入该边，判断源点到目标点之间是否还存在边，存在则测试失败，否则测试成功。

**测试代码：**

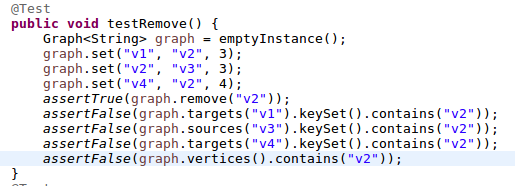


③testRemove()

**remove()实现要求：**从图中删除一个顶点，任何与该顶点相关联的边都应该被删除。如果该边存在于图中，删除成功，返回true，否则返回false，不改变图。

**测试策略：**向图中加入三条边，每条边都包含有同一个顶点，有出边有入边，然后删除公共顶点。判断之前的边是否都还存在。如果存在则测试失败，否则测试成功。判断被删除的顶点是否还存在，存在则测试失败，否则测试成功。

**测试代码：**

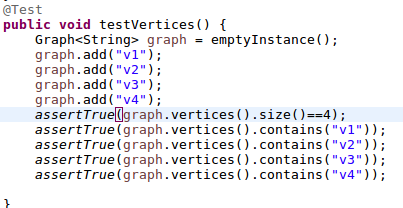
****

**④vertices()**

**vertices()实现要求：**返回包含图中所有顶点的集合。

**测试策略：**向图中加入顶点，判断返回集合的大小和内容是否正确。

**测试代码：**

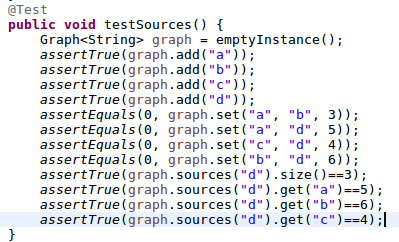
****

**⑤testSources()**

**sources()实现要求：**返回与参数顶点直接相连的源点以及它们之间的边权重

**测试策略：**向图中加入顶点和边，判断某目标点的源点集合的大小和内容是否符合期望，符合则测试成功，否则测试失败。

**测试代码：**

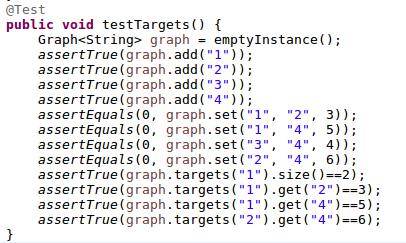
****

**⑥testTargets()**

**targets()实现要求：**返回与参数顶点直接相连的目标点以及它们之间的边权重

**测试策略：**向图中加入顶点和边，判断某源点的目标点集合的大小和内容是否符合期望，符合则测试成功，否则测试失败。

**测试代码：**

****

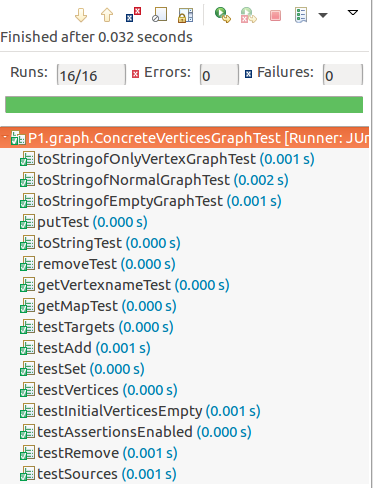
### Problem 2: Implement Graph <String>

#### Implement ConcreteEdgesGraph

**Edge<String>类的设计：**用source表示源点，target表示目标点，weight表示权重，都是private final域，防止属性被改变。类中没有改变属性的方法。有getSource,getTarget和getWeight返回三个属性。checkRep检查source和target是否为空，权重是否非负。

**ConcreteEdgesGraph类的设计：**在类中，用Set<String>vertices表示顶点集合，用List<Edge<String>>edges表示边集合。这两个域都是private final的。checkRep检查顶点是否有重复。toString方法先判断顶点是否为空，如果顶点为空，则表明是空图。再检查边集合是否为空，如果为空，则打印相应提示信息。如果有顶点有边，则打印所有的边的信息。在返回顶点集合时，进行了防御性拷贝，防止顶点集合被随意改变。sources()和targets()方法也进行了相同的处理，防止rep exposure.

**测试结果：**

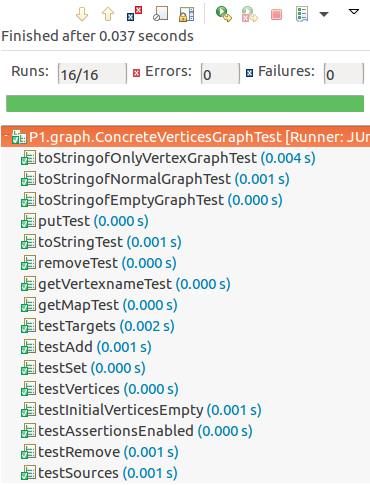
****

#### Implement ConcreteVerticesGraph

**Vertex<String>类的设计：**vertex类是可变类，用vertexname代表顶点名字，map内存储与该顶点相连接的目标顶点的名字以及边权重，所有的属性都是private的。checkRep检查vertexname不为空，并且没有从该顶点到自己的顶点。所有的类型都是private防止rep exposure. toString()函数先判断map是否为空，如果为空则只打印该顶点的相关信息，不为空则同时打印相关边的信息。put()方法和remove()方法可以向map内加入边和删除边。

**ConcretVerticesGraph类的设计：**用vertices存储所有的顶点，该属性是private final的。checkRep检查顶点是否有重复。toString()方法先判断顶点集合是否为空，若为空则返回相应提示信息。如果不为空返回包含所有顶点和边的信息的字符串。

**测试结果：**

****

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

用前面实现的Graph<L>,实现通用的Graph<L>

#### Make the implementations generic

将具体类的声明更改为：

public class ConcreteEdgesGraph<L> implements Graph<L> { ... }

class Edge<L> { ... }

和：

public class ConcreteVerticesGraph<L> implements Graph<L> { ... }

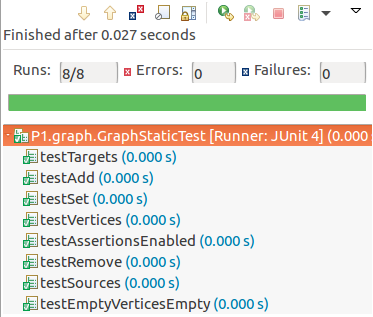
class Vertex<L> { ... }

然后更新两个实现以支持任何类型的顶点标签，找到所有的String并替换为L，完成转换后，测试调整。

#### Implement Graph.empty()

选择一个实现来使用和实现Graph.empty()类，这里使用的类是ConcreteEdgesGraph<L>类。

用Integer重新测试结果如下：



### Problem 4: Poetic walks

通过语料库内的单词之间的关系建立图，每个顶点是一个单词，不区分大小写。用语料库初始化后的关系图来在给定输入的情况下，将一个句子扩充。

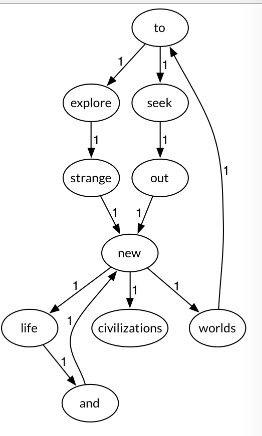
比如给定语料库：

To explore strange new worlds.

To seek out new life and new civilizations.

根据任意相邻的两个单词一起按顺序的出现的次数作为两个单词所代表的顶点之间的边的权重。

生成图如下：



程序通过在每对输入词之间插入单词来通过图生成诗歌。如果在输入的句子的相邻两个单词之间能在图中找到有相应的单词将它们连接，那么可以将这个单词加入句子中输出。如果有多个这样的单词，寻找权重最大的。

#### Test GraphPoet

可以向构造函数传入参数，参数是一个文本文件，里面的内容如下：

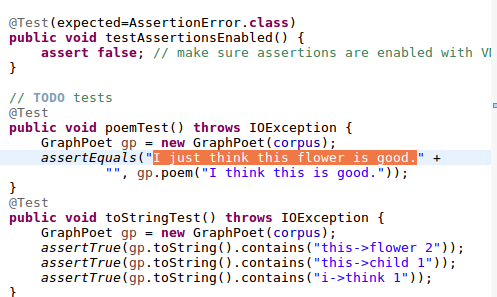
i just think this flower is wonderful,i think this flower is beautiful,i just think this child is smart!

测试句子：

I think this is good.

上面的文本考虑到了有大于1的权重的情况，在多个有效单词之间选择权重最大的情况。上面的句子中有两个从this到flower再到is的边，一个从this到child再到is，所以在输入句子中的this到is之间添加的单词是flower.并且在i和think之间填充了单词just.

测试代码：



#### Implement GraphPoet

**public** GraphPoet(File corpus) **throws** IOException

解析文本，并且建立与语料库相对应的单词图。先将文本中的标点符号全部替换为”标点符号\n”，然后在分割的时候根据换行符和空格分割，注意在分割之前要把内容全部转换成小写字母。这样可以恰当的处理标点符号。然后每次在加边的时候先获得之前边的权重，加1之后更新边的权重。

**public** String poem(String input)

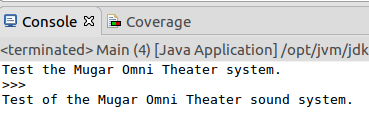
输入的句子按照相同的方法处理分割。然后每次检查两个单词之间在图中是否有连接，如果有连接，则选择权重最大的单词加入生成的答案句子中，如果没有则不作处理。最后返回处理后的字符串。

测试结果：



#### Graph poetry slam

Main.java 运行结果如下：



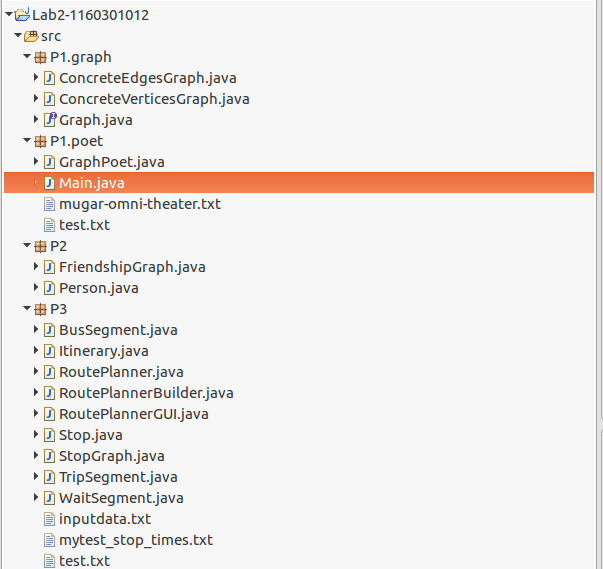
### Before you’re done

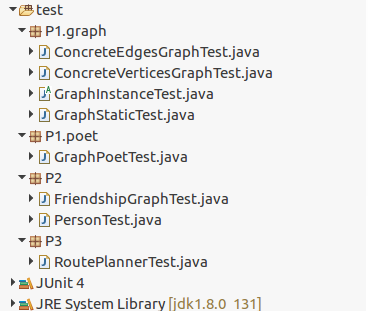
检查是否为所有的实现记录了AF和RI，以及如何防止rep exposure。并且实现了checkRep()，实现了toString方法。

git add .

git commit -m “second”

git push origin master





## Re-implement the Social Network in Lab1

利用前面实现的Graph<L>，重新实现Lab1中的FriendshipGraphPoet类，用Person替换L。

### FriendshipGraph类

用graph代表用户之间的关系，用vertexCount为顶点计数编号，两个属性都是private的。checkRep检查vertexCount是否等于vertices的元素数目。

**addVertex()**向graph中添加顶点，如果添加成功则返回true，否则返回false，如果返回true则vertexCount加1并且为顶点设置编号。

**addEdge()**向graph中添加边，最后返回与第一个人有关系的所有好友列表。

**getDistance()**利用广度优先搜索，寻找最短路径并返回最短路径长度，如果没有连接，则返回-1

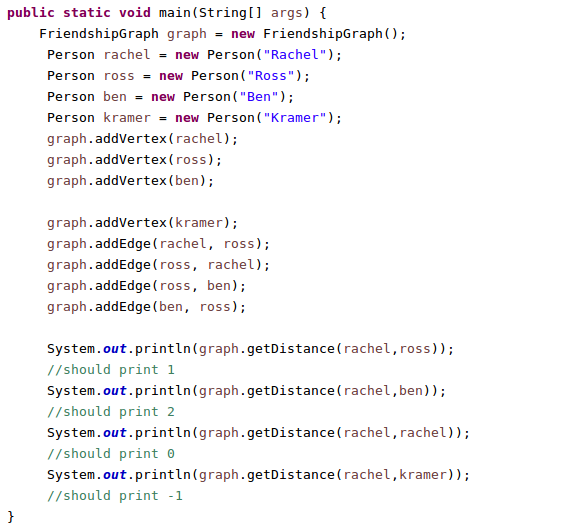
### Person类

Vertexname是顶点的名字，dis是记录从源点到该点的最短距离，在求两个顶点之间最短路径长度的时候使用，num是顶点的编号，map内存的是与该顶点相连接的顶点以及之间的权重，list内存的是从该顶点出发相连接的所有顶点。

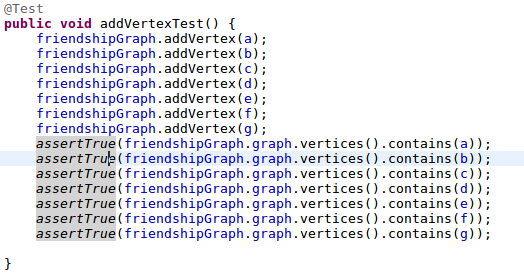
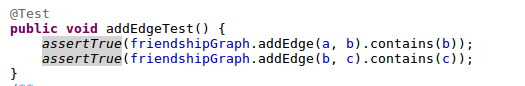
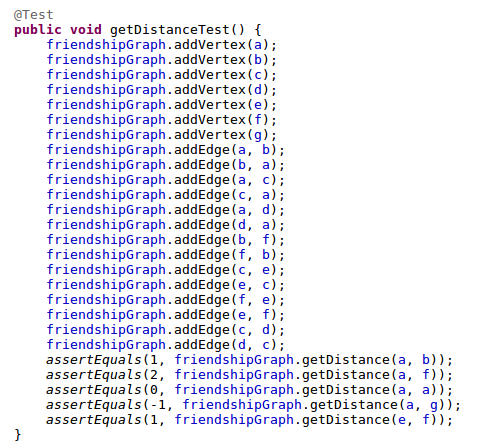
这些属性都是private的。

checkRep检查在list中是否有重复元素，vertexname是否为空

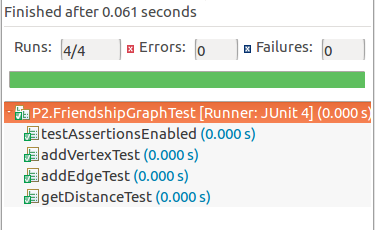
### 客户端main()



### 测试用例



测试结果：



### 提交至Git仓库

git add .

git commit -m “message”

git push origin master

在这里给出你的项目的目录结构树状示意图。

## The Transit Route Planner（选做，额外给分）

根据输入建立公交路线图，并为给定的起点站和终点站设计最快的路线。

**Stop类的设计：**

route属性是stop所在的路线名，name是站名，latitude和longitude是站的经纬度，time是该站代表的固定地点固定路线固定车辆经过的时间，visited是在dijkstra遍历图时是否被访问过(是否被确定最短),isDivided表示顶点是否被分裂过(因为换乘的原因),id用来区分分裂后的顶点的两个顶点，son1和son2是为了将分裂前的顶点和分裂后的顶点联系起来，便于操作，parent是在dijkstra遍历时记录路径用。dist是到源点的最短路径。similarto方法是判断两个站的位置(名字和经纬度)是否相同。另外为了区分不同的stop,重写了equals和hashcode方法。

**StopGraph类的设计：**

stopGraph里的大多数代码是用ConcreteVerticesGraph类里的内容写的，修改了set方法:之前的set方法当权重为0时等于删除该边，在这里权重为零的边是允许存在的，所以修改了这个地方。另外添加了previous和next类，因为在建图初始(还没有分裂顶点并且加换乘边的时候)，每个顶点一定是最多一个出边和一个入边的，所以这两个函数的作用就是找到参数顶点的源点和目标点，以便在分裂顶点时重新建立边点关系。

**RoutePlannerBuilder类的设计：**

**Build方法：**先解析传入的文件，向图中加入顶点和边，建立交通网，将有不止一条线路经过的顶点拆分成两个顶点，以解决多次换乘导致等待时间超过20分钟的情况。



然后判断可换乘的顶点要等待的时间是否超过最大等待时间，如果没有超过，就在该站的分裂点的第一个连接到相同位置满足等待时间关系的另一个顶点的分裂点的第二个。最后，RoutePlanner对象，用一个建好的图作为参数。

**RoutePlanner类的设计：**

**findStopsBySubstring()方法:**遍历图中所有顶点，如果包含search子字符串，则将该站加入list，否则不加入。

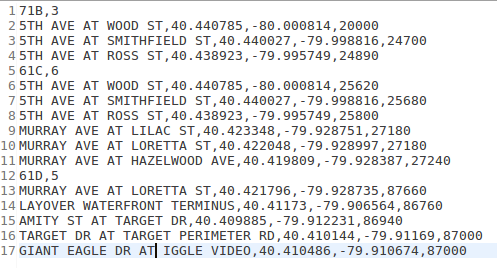
**computeRoute()方法：**先根据time参数，将图中在time之前的顶点和相关联的边去掉，减轻搜索负担。然后将起点和终点与图建立联系，起点与所有和起点在同一经纬度，名字相同的stop连边，边的权重为站的时间减去出发时间的大小。将所有与目标站的名字和经纬度相同的站和目标站建立联系，边的权重为0。最后删除那些能够被分裂（可换乘）的站点被分裂之前的顶点，最后选择起点站为起点，目标站为重点，利用dijkstra算法搜索并打印最短路径。

**RoutePlannerGui()类的设计：**

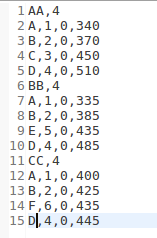
根据用户输入的起始站和目标站以及准备出发时间，查找最短路径并打印。

**测试：**

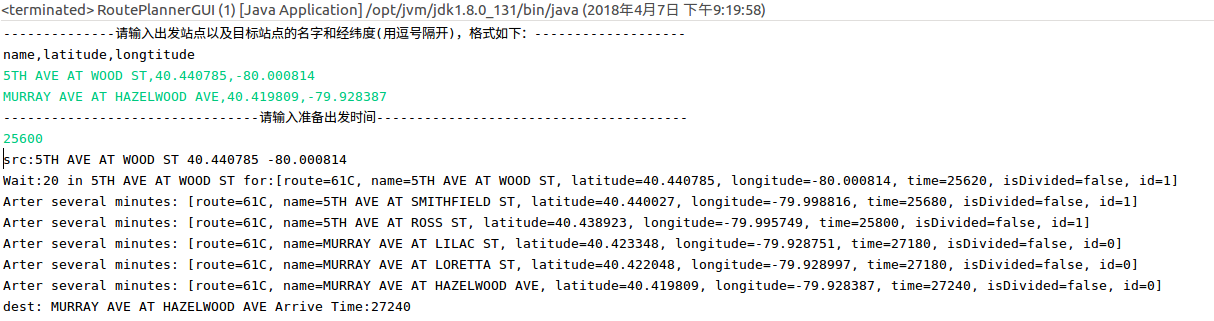
测试文本(1)：



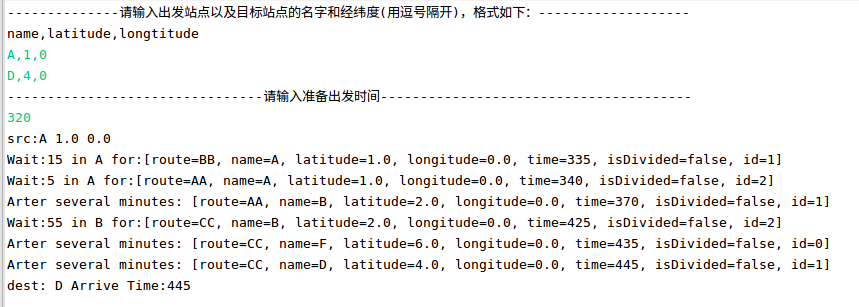
测试文本(2):



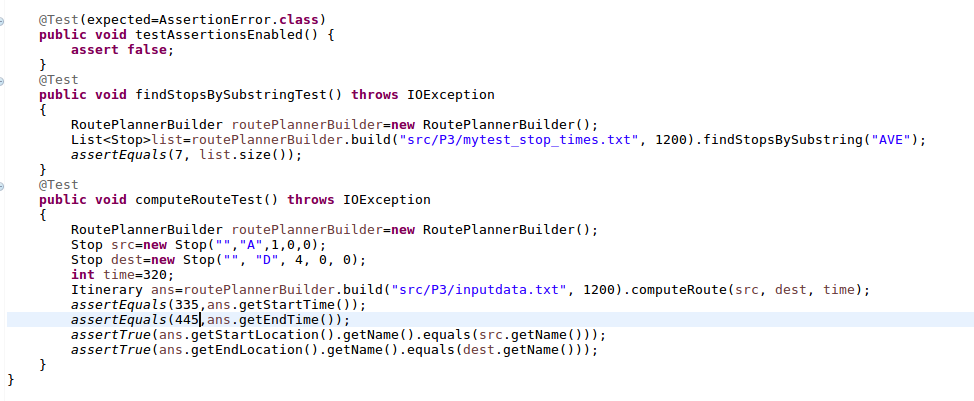
测试结果(1):



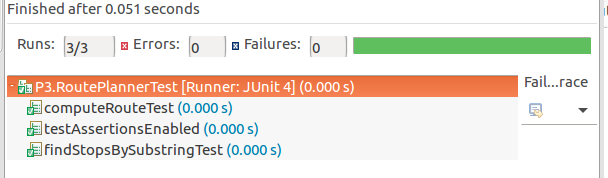
测试结果(2):



Test测试：



测试结果：



# 实验进度记录

请尽可能详细的记录你的进度情况。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 3.24-3.25 | 13：00-15:00 | P1 | 完成 |
| 3.31 | 13：00-15:00 | P2 | 完成 |
| 4.1-4.5 | 13：00-15:00 | P3 | 完成 |

# 实验过程中遇到的困难与解决途径

关于mutable和immutable类的概念不是很清晰，通过查阅相关书籍和资料明白。对AF，RI的理解不是很透彻，通过去mit课程官网阅读相关阅读材料搞清楚。另外对于exposure的处理还有一些疑问，和同学讨论找到了解决方法。

# 实验过程中收获的经验、教训、感想

①面向ADT的编程和直接面向应用场景编程的差异在于：面向ADT编程会有更多的限制，但是做好之后代码复用很方便，更安全。面向应用场景编程比较自由，但是容易出问题。②使用泛型的编程，有利于提高代码的可重用性，类型安全和效率，而不使用泛型会导致代码的冗杂。③在给出ADT的规约之后就开始编写测试用例，能够更早的发现错误，越早发现错误，改正错误越简单。能更深入更早的了解要做的题目的需求，确保测试代码的有效性。编写测试用例首先在编写代码之前就暴露出需求问题，发现一些需要特别注意的地方。确保每个功能的测试被实现，确保了程序是为可测试性编写。我觉得这种测试方式刚开始让人感觉非常不适，但是我也确实因为这个提高了效率，没有考虑全面的可能性降低，从而使得代码bug更少，debug更简单。④这种复用使得代码可读性更强，代码更简洁，工作效率更高。⑤编写RI，AF，specification, invariant,rep exposure可能很烦，但是我还是愿意去做的，因为这确实提高了代码的安全性和可读性，。⑥前两个实验工作量适当，第三个实验由于实验要求不完全，实验难度略大等原因，我感觉还是有点紧张的。