

**2023年春季学期  
计算学部《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 |  |
| 学号 | 2021110577 |
| 班号 | 2137102 |
| 电子邮件 | 934692896@qq.com |
| 手机号码 | 15750682634 |

**目录**

[1 实验目标概述 1](#_Toc72247573)

[2 实验环境配置 1](#_Toc72247574)

[3 实验过程 1](#_Toc72247575)

[3.1 Poetic Walks 2](#_Toc72247576)

[3.1.1 Get the code and prepare Git repository 2](#_Toc72247577)

[3.1.2 Problem 1: Test Graph <String> 2](#_Toc72247578)

[3.1.3 Problem 2: Implement Graph <String> 4](#_Toc72247579)

[3.1.3.1 Implement ConcreteEdgesGraph 4](#_Toc72247580)

[3.1.3.2 Implement ConcreteVerticesGraph 5](#_Toc72247581)

[3.1.4 Problem 3: Implement generic Graph<L> 6](#_Toc72247582)

[3.1.4.1 Make the implementations generic 6](#_Toc72247583)

[3.1.4.2 Implement Graph.empty() 7](#_Toc72247584)

[3.1.5 Problem 4: Poetic walks 7](#_Toc72247585)

[3.1.5.1 Test GraphPoet 7](#_Toc72247586)

[3.1.5.2 Implement GraphPoet 8](#_Toc72247587)

[3.1.5.3 Graph poetry slam 8](#_Toc72247588)

[3.1.6 Before you’re done 9](#_Toc72247589)

[3.2 Re-implement the Social Network in Lab1 9](#_Toc72247590)

[3.2.1 FriendshipGraph类 10](#_Toc72247591)

[3.2.2 Person类 11](#_Toc72247592)

[3.2.3 客户端main() 11](#_Toc72247593)

[3.2.4 测试用例 12](#_Toc72247594)

[3.2.5 提交至Git仓库 12](#_Toc72247595)

[4 实验进度记录 12](#_Toc72247596)

[5 实验过程中遇到的困难与解决途径 12](#_Toc72247597)

[6 实验过程中收获的经验、教训、感想 13](#_Toc72247598)

[6.1 实验过程中收获的经验和教训 13](#_Toc72247599)

[6.2 针对以下方面的感受 13](#_Toc72247600)

# 实验目标概述

* 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。

# 实验环境配置

本次实验沿用了lab1搭建好的所有相关配置，没有引入新软件，所以省略了安装与配置的过程。

编译与调试均使用JetBrain Idea 2022.3.3

测试覆盖度使用Idea内置相关插件

提交与推送使用Git

我的github-lab2的URL地址是：

https://github.com/ComputerScienceHIT/HIT-Lab2-2021110577

# 实验过程

请仔细对照实验手册，针对两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！）。

## Poetic Walks

该问题内容主要分为两部分，前半部tr分重在阐述与演示ADT相关概念的性质与实现，后半部分重在阐述类间方法间关系与调用。

### Get the code and prepare Git repository

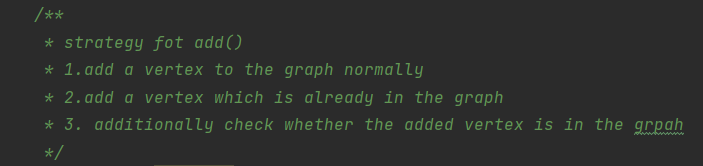
文件解压形式

### Problem 1: Test Graph <String>

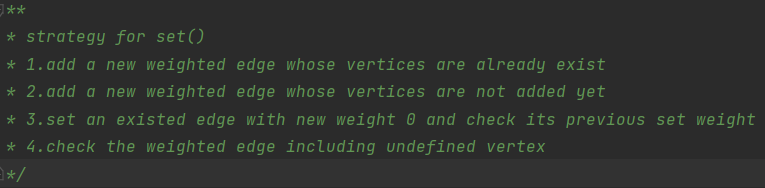
逐项阅读Graph.java中ADT接口Graph里每个方法的功能，形参和返回值，假设内部已经被完美实现，从而根据其特性设计可以全面测试功能的策略和代码。

*测试策略：*

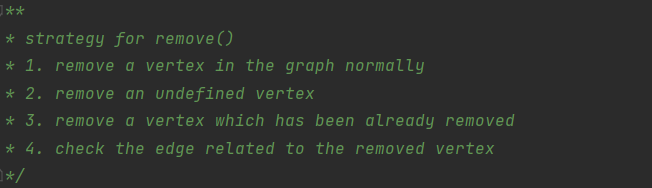
* Test\_Add



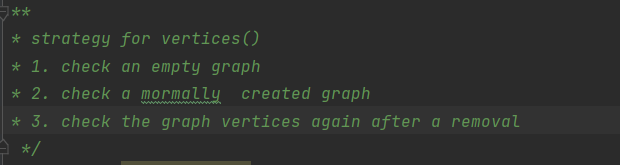
* Test\_Set



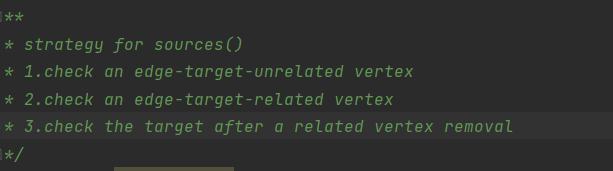
* Test\_Remove

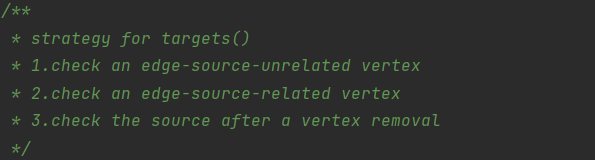


* Test\_Vertices

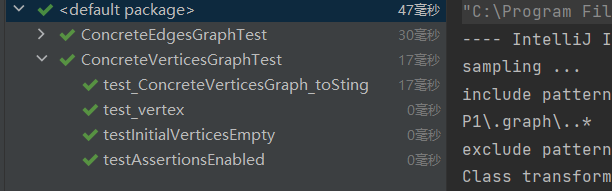


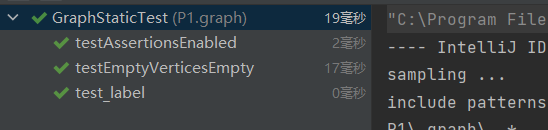
* Test\_Sources & Test\_Targets（二者测试思路相同）

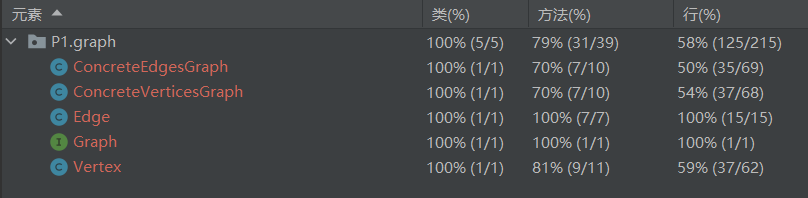




以下是实现完功能后的测试结果以及覆盖度:







### Problem 2: Implement Graph <String>

#### Implement ConcreteEdgesGraph

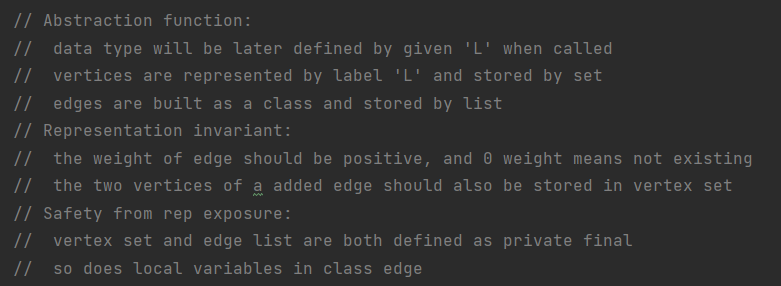
首先实现Edge类：

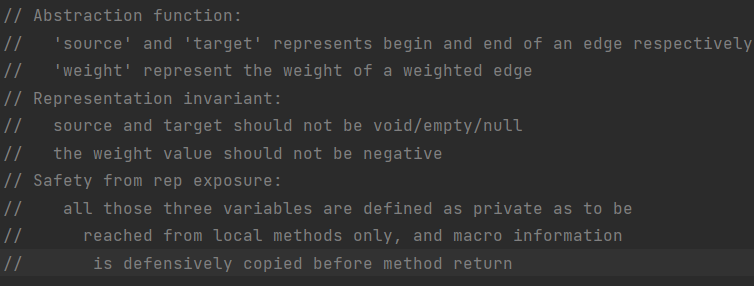
1. 定义私有局部变量source，target和weight，并将变量初始化写入构造函数.
2. ChecRep检查三个变量的值是否合法，要求定点非空、权值为正。
3. 设计访问内部信息的get函数，分别返回需要的局部变量，防止泄露。
4. ToString函数，用一种安全的方式打印边的信息。

继而实现EdgeGraph类：

1. 定义私有变量，用Set存点，List存边，并将初始化写入构造函数。
2. CheckRep检查变量是否合法，每个边的权值都为争，并且每个边的顶点都存在于点集内，得益于Set的性质无需检验是否有重复点。
3. Add：如果点集内存在目标点则返回false，否则将点加入返回true。
4. Set：检查权值是否非负，不满足则抛出异常。先将目标边顶点加入点集，检查边集内如果存在该边则设置新权值返回就权值，这里因为边集设置的为private final，所以只能将旧边移除再将新边加入，如果不存在该边则直接加入并返回0。
5. Remove：检查是否存在该点，存在则遍历边集合，移除所有和该点有关的边，最后将其移出点集。
6. Source/Target：初始化点到权值的Map，遍历边集，将指定点作为入点/出点的边相应信息存入Map返回。

* 使用边类构建图关系，单独再开一个点集存贮，降低复杂度
* 所有对内部信息进行修改的方法都应该及时调用CheckRep检查
* 跟随编写合适且准确的Abstraction function、Representation invariant以及safety from rep exposure





#### 1.Implement ConcreteVerticesGraph

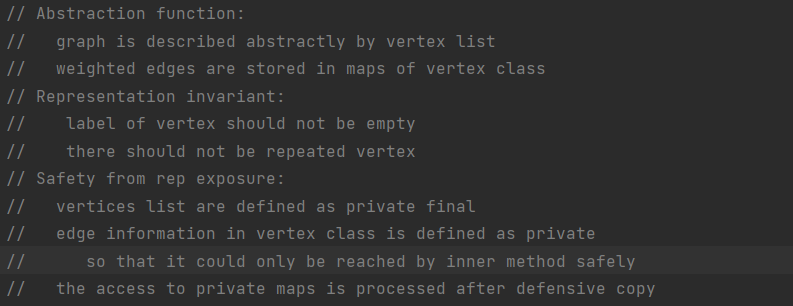
首先实现Vertex类：

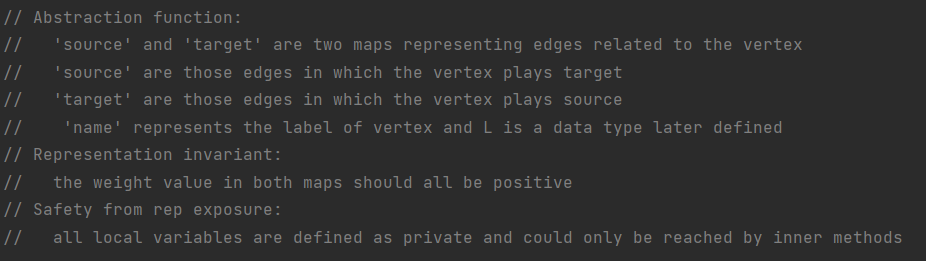
1. 定义局部变量标签name以及分别象征着顶点作为边起始点和边终点的、由边另侧顶点到边权值的Map，并在构造函数中将其初始化。
2. CheckRep检查每个边的权值均为正且顶点标签不为空。
3. Remove Source / Remove Target： 条件判断抛出异常防止空输入，检查该点的邻接边起始点集/邻接边终点集是否包含目标点，若存在则将其移除。
4. Add source / Add target：检查新邻接点是否为空以及权值是否为负，如果该点的目标点集中已经存在新临界点，则根据权值更新替换目标点集Map中的键值，若不存在则在Map中加入新的邻接边信息。
5. 设计Get方法访问内部信息，分别返回点标签、起始点Map与终点Map。
6. 用toString函数以安全的方式返回处理过的顶点信息

继而实现VerticesGraph类：

1. 定义私有变量列表存储点集，并将初始化写入构造函数。
2. CheckRep检查每个顶点的标签不为空，以及列表中不存在重复标签的点。
3. Add：检查目标点是否存在于点集中，若不存在则将其加入。
4. Set：检查权值非负以及顶点非空，将点加入点集，遍历点集搜寻每个点的初始点Map和终点Map，找到两个目标顶点，分别将他们加入到对应Map中。
5. Remove：搜寻点集检查是否存在目标点，在搜寻该点是否存在与其它点的Map中，将找到的映射全部remove掉，最后在点集中删除该点。
6. Vertices：为vertices点集做一份防御式拷贝后将副本返回。
7. Sources / Targets：给目标点的初始点映射/终点映射做一份防御式拷贝后返回副本。

* 当使用点集构建边时，Map类集合映射可以较好地构建有向与带权的对应关系
* 所有对内部信息进行修改的方法都应该及时调用CheckRep检查
* 跟随编写合适且准确的Abstraction function、Representation invariant以及safety from rep exposure



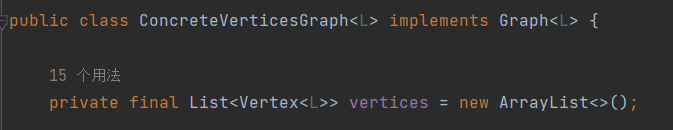


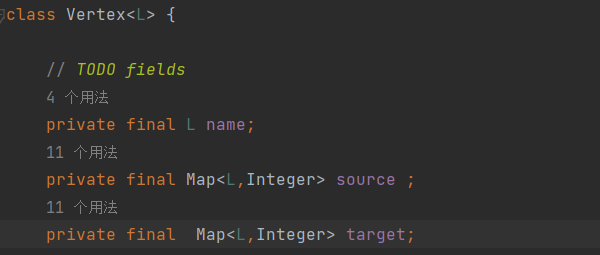
### Problem 3: Implement generic Graph<L>

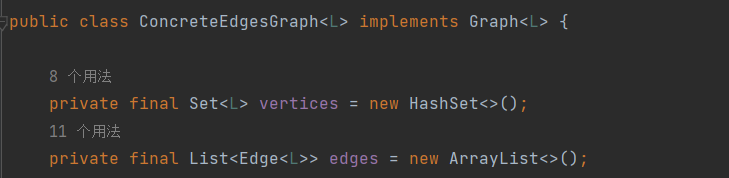
#### Make the implementations generic

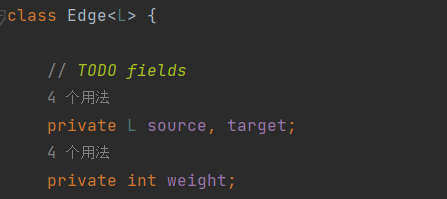
比较简单，只需在类名与变量初始化构建时将指定的<String>类型全部改为泛型<L>

应该注意在代码编写时候不能有，适用于String而不适用于其他类型的方法，本实验在完成过程中保守地尽量使用了通用方法如.equals/.contains。



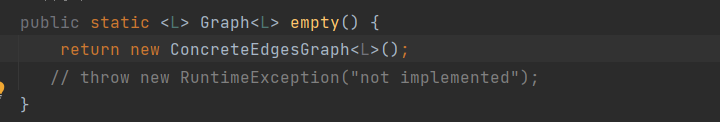






#### Implement Graph.empty()

直接初始化一个新的graph就行了，因为edge类的构建比较简单，遇到问题也好调试修改，所以这里采用ConcreteEdgeGraph来实现。

****

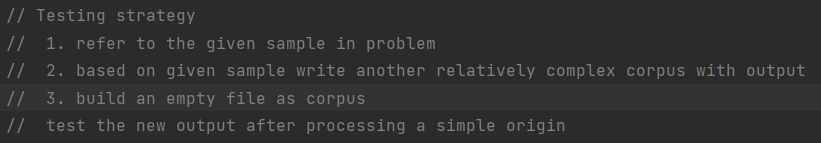
### Problem 4: Poetic walks

#### Test GraphPoet

测试策略：1.题干给定处理组

2.根据算法理解自己构造一个较复杂的处理组

3.空corpus处理组



以下为功能实现后的测试结果：





#### Implement GraphPoet

算法理解：

大致内容为，通过读取给定文本作为corpus构建一个有向图，文本中每对相邻的单词都会构建出一条由前一个单词指向后一个单词的有向边，并且根据出现次数赋予其权重，当我们输入input后，会逐次搜寻这个构建好的图，如果输入的句子中，存在相邻两个词，他们在corpus图中，可以由前一个词经由一个第三词再到后一个词（只能经由一个且有向），那么就在输出中将第三词插入到这对词的中间，依次扩写整个输入文本。

实现思路：

* GraphPoet

将corpus从文件读入文本并由此构建一个有向带权图，要点在预处理，逐行读入后要通过split将读入数组根据空格，句号问好等常见标点进行文本分割，为了防止匹配过程的问题，将所有单词小写化，将其处理成纯净的无歧义单个单词存入字符串数组，再依次读取字符串数组构建图。

* CheckRep

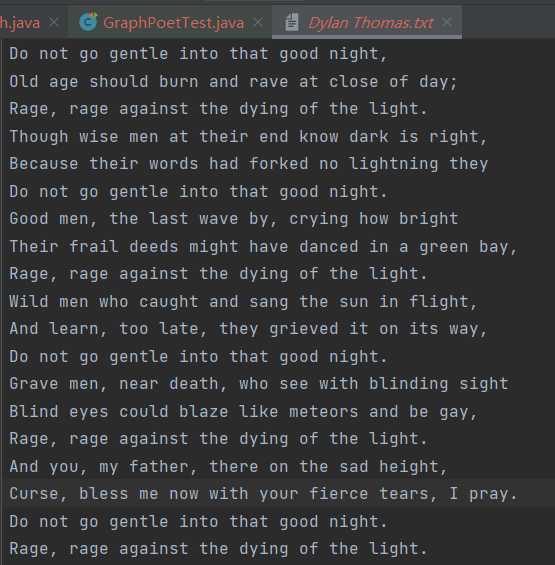
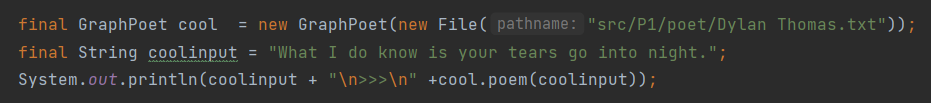
检查图中节点不存在空格，空串以及换行符等非法字符。

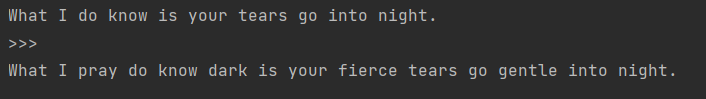
* Poem

读取输入字符串直接按空格拆分存入字符串数组，这里只通过空格拆分是为了再输出句子的时候省去还原标点的复杂操作，我们直接将标点视为单词的一部分，因为英文语句书写中标点紧跟前一个单词再空格，所以还原句子合理，进行搜索时，如果前一个单词的终点集中存在和后一个单词的初始点集一样的单词，则将其暂存，选取所有暂存词中权值最大的一个存入输出字符串并附上一个空格，依次扩写。

#### Graph poetry slam

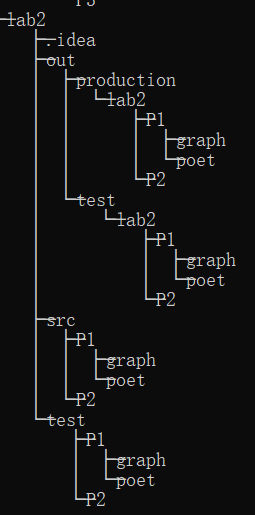
根据MIT课件中对主函数添加输出样例中的所谓“cool sample“的要求，我动容了，于是选用了个人比较中意的迪兰托马斯的一首比较有名的诗歌《do not go gentle into that good night》当作corpus，收获了令人满意的扩写结果。





### Before you’re done

* Git add \*.\*
* Git commit
* Git push



## Re-implement the Social Network in Lab1

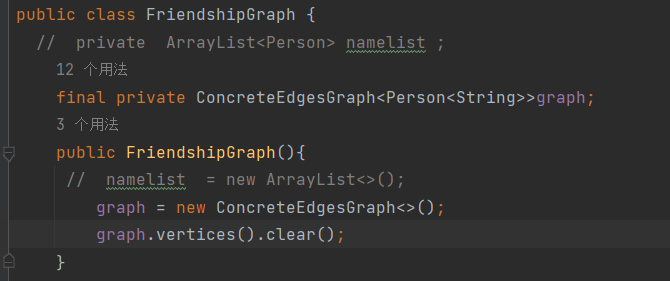
通过重写项目，切身感受规范的ADT相较于朴素无序编写的程序在理解、移植以及调试性能上的不同，以及学会对于提升程序安全性需要做出的修改。

利用上文中完成的Graph类方法重新实现Friendship类方法，其中图的构建概念与Person节点不变，只是重构实现方法。

### FriendshipGraph类

* 构造方法

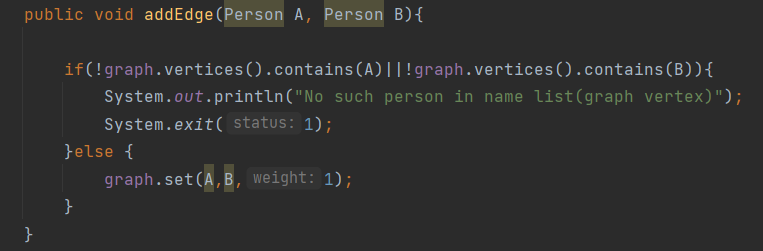
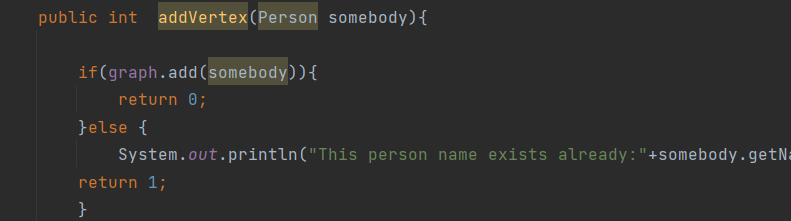
将原方法中的列表存储名称节点，Person类中列表存储相关边的图实现方式改为直接新建一个Graph类的图（这里选用Edge类实现，因为他的构造比较简单出了问题好调试）。



* Addvertex & Addedge

二者思路相同，检测原图中是否已经存在目标边/点，如果存在则打印报错并异常返回，如果不存在则通过Graph类的add与set方法直接将其添加，可以看到转换后的方法实现十分简洁易懂。

因为这个方法中并不涉及到边的权重，所以在添加时只需默认权值为1即可。

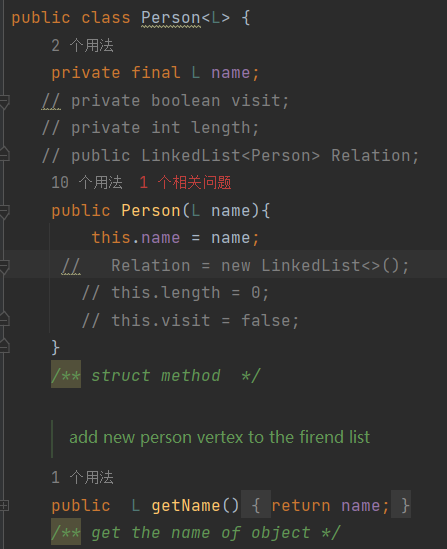


* Getdistance

大致算法思路没有出入，选择了非递归版本的广度优先搜索，只是原方法在标记visited与distance时通过在Person内部实现将其绑定在一起，通过Person类的accessor访问以及mutator更新，而这么做在本次实验中是有悖主题的，改用了在方法中分别定义两个从Person到Visited和从Person到distance的映射来防止对内部信息的访问修改，其他部分代码几乎没有任何需要修改的地方。

### Person类

根据ADT与OOP的相关设计原则，需要增强原写法的权限管理和防止表示泄露，因此要移除其中所有的mutator方法，只留下accessor，并将内部变量声明为private final类型，从而提高了安全性，那么这时候Person类中只剩下变量name和一个构造方法、一个accessor了，最后再把name改成泛型L，就完成了Person的规范化重写。

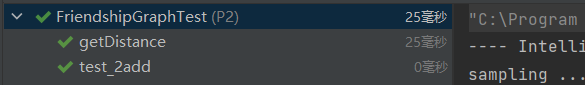


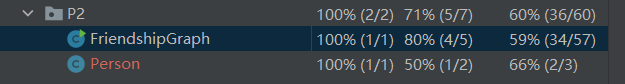
### 客户端main()

用于调试检验方法功能，只进行泛型的声明改动，其他不改写。

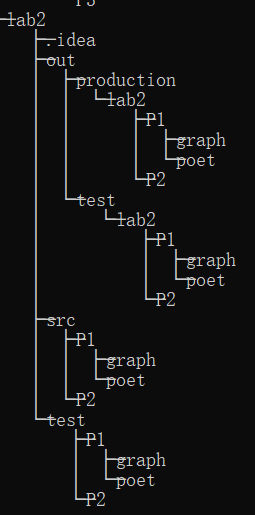
### 测试用例

沿用原测试策略，除写法有调整外，整体测试算法不改写。





### 提交至Git仓库



# 实验进度记录

请使用表格方式记录你的进度情况，以超过半小时的连续编程时间为一行。

每次结束编程时，请向该表格中增加一行。不要事后胡乱填写。

不要嫌烦，该表格可帮助你汇总你在每个任务上付出的时间和精力，发现自己不擅长的任务，后续有意识的弥补。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 4.1 | 16:30-20:30 | 复习ADT相关课程内容 | 如期进行 |
| 4.2 | 13:00-15:30 | 阅读理解全部实验任务 | 不太顺利，没搞懂具体实验目的 |
| 4.6 | 13:30-20:30 | 完成Graph的两种实现 | 如期进行 |
| 4.7 | 10:30-13:30  16:30-22:30 | 收尾Graph的调试并且完成poem部分 | 在处理标点问题上遇阻，poem测试用例出现很多bug |
| 4.8 | 13:30-16:30 | 完成P2 | 进行顺利但没写完 |
| 4.9 | 10.30-1:30 | 收尾改写FriendshipGraph | 如期进行 |
| 4.9 | 15.30-23:30 | 撰写实验报告 | 主要部分完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 读不明白具体的实验要求，尤其是刚开始在实现功能之前直接让写测试，下不去手 | 一方面复习课程内容，加强理解了ADT、OOT及相关规范设计思路，一方面上网找实验指导以及问同学 |
| Poem部分在处理标点时遇到很多问题 | 起初预处理想统一纯净化单词并存储，这就需要去除所有除小写字母以外的东西，第一个念头是全部Split，并且用正则表达式检验，但是这导致在生成输出时无法还原标点，最后还是选择牺牲了扩写效果，直接将合法标点视作单词的一部分，尽管这会导致生成句莫名其妙的断句，但其实也符合题意要求。 |
| 改写过程中思路错误 | 没能完全理解设计思想和规范内涵，导致Person改写不充分，getDistance依旧存在有风险的访问方法，多亏了同学的指点与讲解，深入理解Rep Exposure，完善了改写。 |

# 实验过程中收获的经验、教训、感想

## 实验过程中收获的经验和教训

* 遇见不会的直接问，不丢人
* IDEA自带的对程序语言规化简洁化的改写插件十分有用，可以让人学到更好的写代码习惯以及更合适的写作范风格
* 泛型改写时候不能一味盲目的直接加<L>，可能会违规
* Junit的测试报错在错误定位上可信度不是很高，还是得具备手动Debug的能力
* 这部分课程内容学的不够精导致做实验时候需要重新学习应该反思

## 针对以下方面的感受

1. 面向ADT的编程和直接面向应用场景编程，你体会到二者有何差异？

面向ADT有更好的规范性，初期学习时看似用处不大的AF，RI其实在实际编程中有极高作用，可以实现测试用例于方法实现的分开编写，提高效率也减少测试盲区，并且时刻保持参数于结构合法，省区可能出现的报错，提高编程效率，从安全层面上防止表示泄露能让人养成更好的编程习惯；更多地从重写Lab1任务中体会到了规范的ADT具有极好的可移植性，在以后大工程搭建时可以节省很多精力。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

泛型有着极好的容错性和可移植性，可以在打项目中节省精力。

1. 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？

扫除测试盲区，如果先写代码再写测试，会陷入思维定势，根据自己的编程思路设计对口的测试样例，意义不大，很可能没有有效测出bug所在，而先写测试案例纯粹从功能角度出发，可以实现更有效的测试，我会逐渐适应的。

1. 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？

大幅度提升编程效率，减少反复编写相似功能代码，而且成熟的可移植代码还可以防止可能出现的报错和bug，省区了潜在的调试风险。

1. 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？

增强本人以及其他协同编程的工作组对于每个类每个方法的理解，省去阅读代码的复杂，更加体现了面向对象编程的抽象与封装概念，我以后会坚持这么做。

1. 关于本实验的工作量、难度、deadline。

工作量适中，难度适中，deadline太早了。

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

恳求老师讲得再慢点。