

**2020年春季学期  
计算机学院《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 郭茁宁 |
| 学号 | 1183710109 |
| 班号 | 1837101 |
| 电子邮件 | gzn00417@foxmail.com |
| 手机号码 | 13905082373 |

# 目录

[0 目录 2](#_Toc35684250)

[1 实验目标概述 1](#_Toc35684251)

[2 实验环境配置 2](#_Toc35684252)

[2.1 安装EclEmma 2](#_Toc35684253)

[2.2 GitHub Lab2仓库的URL地址 3](#_Toc35684254)

[3 实验过程 4](#_Toc35684255)

[3.1 Poetic Walks 4](#_Toc35684256)

[3.1.1 Get the code and prepare Git repository 4](#_Toc35684257)

[3.1.2 Problem 1: Test Graph <String> 4](#_Toc35684258)

[3.1.3 Problem 2: Implement Graph <String> 4](#_Toc35684259)

[3.1.3.1 Implement ConcreteEdgesGraph 5](#_Toc35684260)

[3.1.3.2 Implement ConcreteVerticesGraph 6](#_Toc35684261)

[3.1.4 Problem 3: Implement generic Graph<L> 7](#_Toc35684262)

[3.1.4.1 Make the implementations generic 7](#_Toc35684263)

[3.1.4.2 Implement Graph.empty() 7](#_Toc35684264)

[3.1.5 Problem 4: Poetic walks 7](#_Toc35684265)

[3.1.5.1 Test GraphPoet 7](#_Toc35684266)

[3.1.5.2 Implement GraphPoet 7](#_Toc35684267)

[3.1.5.3 Graph poetry slam 8](#_Toc35684268)

[3.1.6 Before you’re done 8](#_Toc35684269)

[3.2 Re-implement the Social Network in Lab1 9](#_Toc35684270)

[3.2.1 FriendshipGraph类 10](#_Toc35684271)

[3.2.2 Person类 11](#_Toc35684272)

[3.2.3 客户端main() 11](#_Toc35684273)

[3.2.4 测试用例 12](#_Toc35684274)

[3.2.4.1 简单图测试 12](#_Toc35684275)

[3.2.4.2 复杂图测试 13](#_Toc35684276)

[3.2.4.3 Junit测试结果 13](#_Toc35684277)

[3.2.5 提交至Git仓库 14](#_Toc35684278)

[3.3 Playing Chess 14](#_Toc35684279)

[3.3.1 ADT设计/实现方案 14](#_Toc35684280)

[3.3.2 主程序MyChessAndGoGame设计/实现方案 14](#_Toc35684281)

[3.3.3 ADT和主程序的测试方案 15](#_Toc35684282)

[4 实验进度记录 16](#_Toc35684283)

[5 实验过程中遇到的困难与解决途径 17](#_Toc35684284)

[6 实验过程中收获的经验、教训、感想 18](#_Toc35684285)

[6.1 实验过程中收获的经验和教训 18](#_Toc35684286)

[6.2 针对以下方面的感受 18](#_Toc35684287)

# 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：

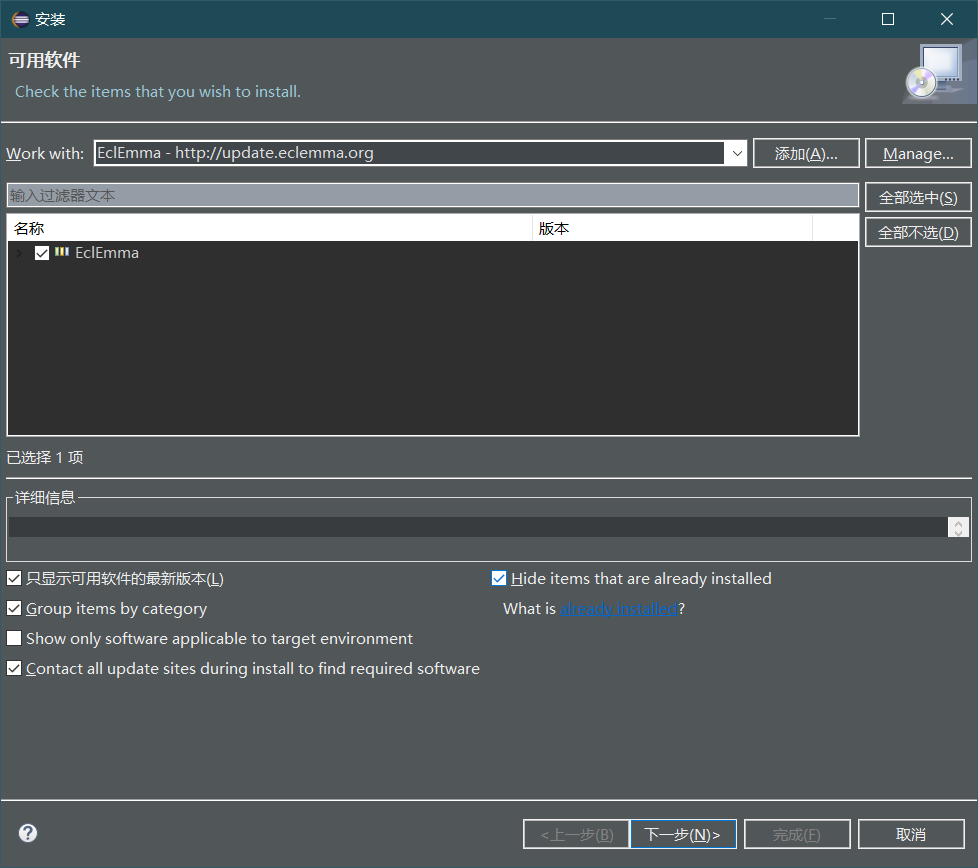
* 针对给定的应用问题，从问题描述中识别所需的ADT；
* 设计ADT规约（pre-condition、post-condition）并评估规约的质量；
* 根据ADT的规约设计测试用例；
* ADT的泛型化；
* 根据规约设计ADT的多种不同的实现；针对每种实现，设计其表示（representation）、表示不变性（rep invariant）、抽象过程（abstraction function）
* 使用OOP实现ADT，并判定表示不变性是否违反、各实现是否存在表示泄露（rep exposure）；
* 测试ADT的实现并评估测试的覆盖度；
* 使用ADT及其实现，为应用问题开发程序；
* 在测试代码中，能够写出testing strategy并据此设计测试用例。

# 实验环境配置

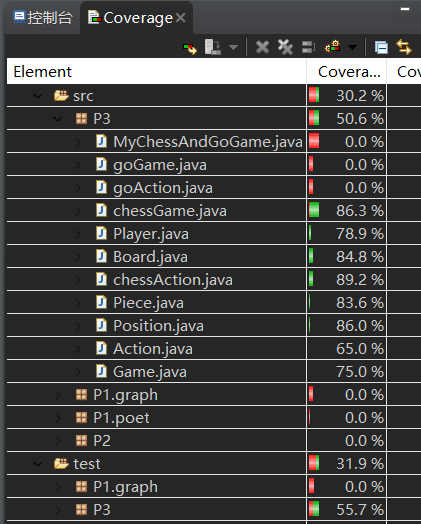
## 安装EclEmma

依据<https://www.eclemma.org/installation.html>内容，从更新站点进行安装。

* 从Eclipse菜单中选择帮助 → 安装新软件；
* 在“安装”对话框中，在“ 工作日期”字段中输入http://update.eclemma.org/；



* 检查最新的EclEmma版本，然后按“下一步”；
* 重启eclipse，即可在java的透视图工具栏中找到coverage启动器，表示安装成功。
* 使用效果



## GitHub Lab2仓库的URL地址

<https://github.com/ComputerScienceHIT/Lab2-1183710109>

# 实验过程

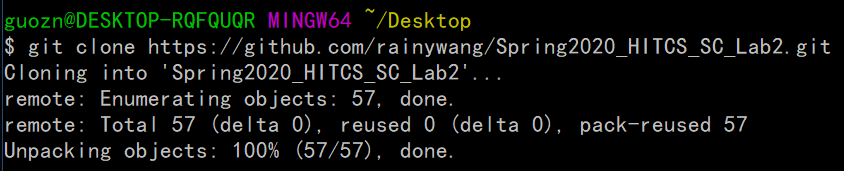
## Poetic Walks

该任务主要是实验一个图的模块，并基于此使用。

* 完善Graph接口类，并运用泛型的思想，将String拓展为泛型L类；
* 实现Graph类的方法：add、set、remove、vertices、sources、targets；
* 利用实现的Graph类，应用图的思想，实现GraphPoet类，如果输入的文本的两个单词之间存在桥接词，则插入该桥接词；若存在多个单一桥接词，则选取边权重较大者。

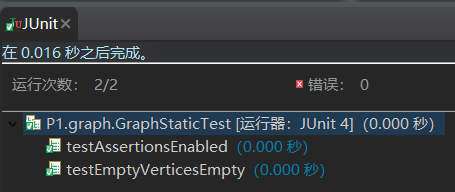
### Get the code and prepare Git repository

git clone https://github.com/rainywang/Spring2020\_HITCS\_SC\_Lab2.git



### Problem 1: Test Graph <String>

测试静态方法生成String类型的Graph。



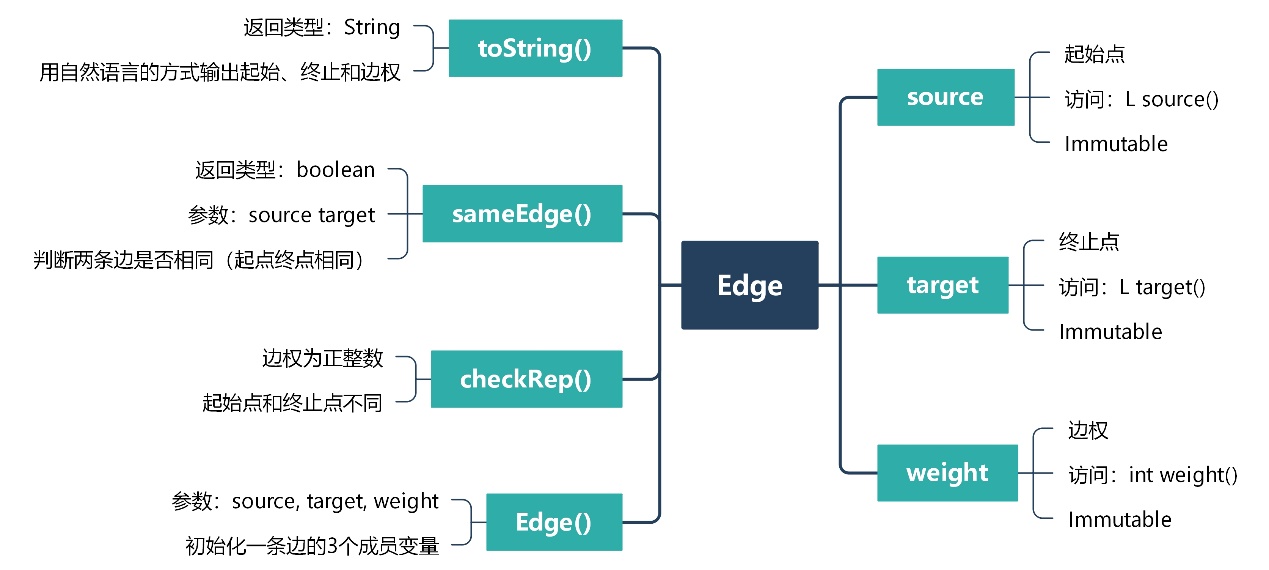
### Problem 2: Implement Graph <String>

该部分要求重写Graph里的方法，分别以点为基础的图和以边为基础的图。

#### Implement ConcreteEdgesGraph

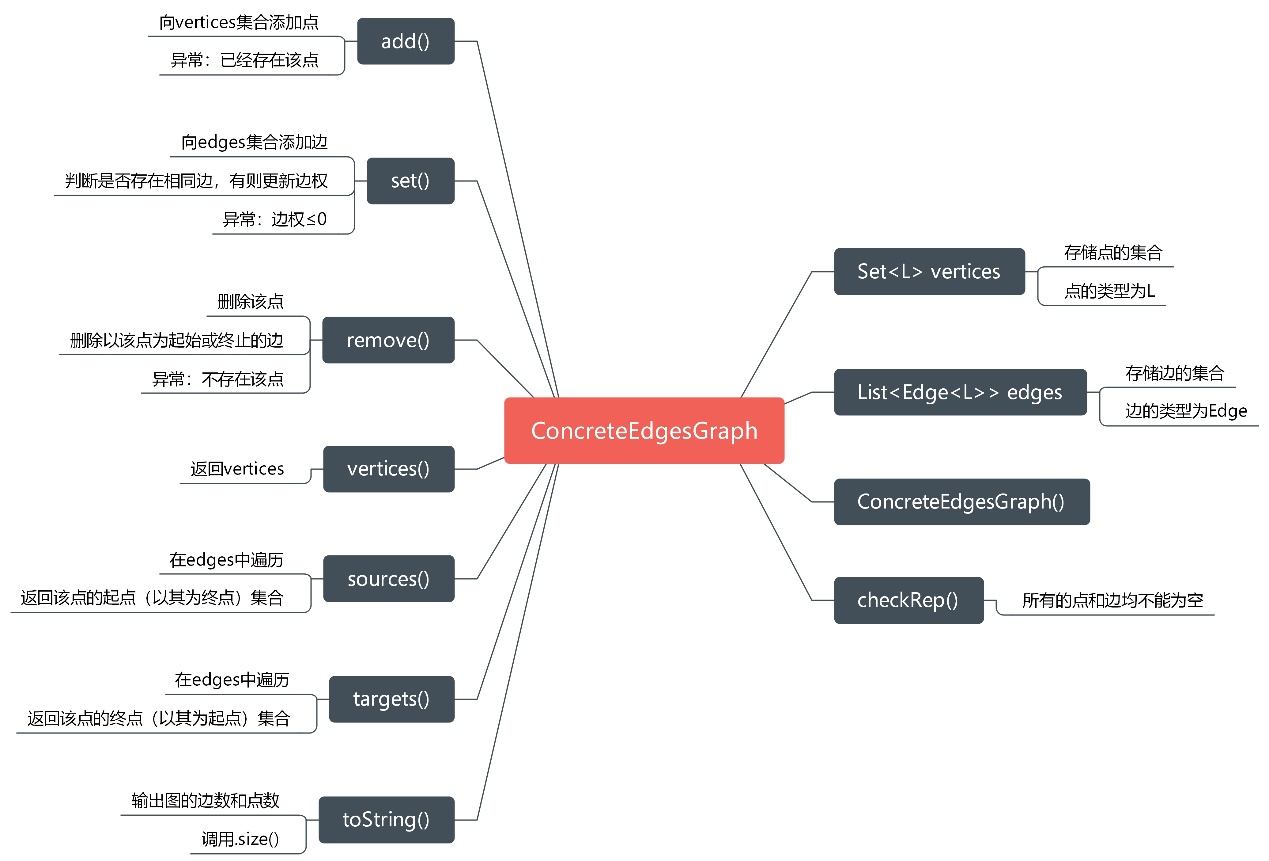
Edge实现

Edge的功能主要为存储边的3个信息。此外，为了Graph实现方便，增加了判断两条边是否相等的方法。



ConcreteEdgesGraph实现

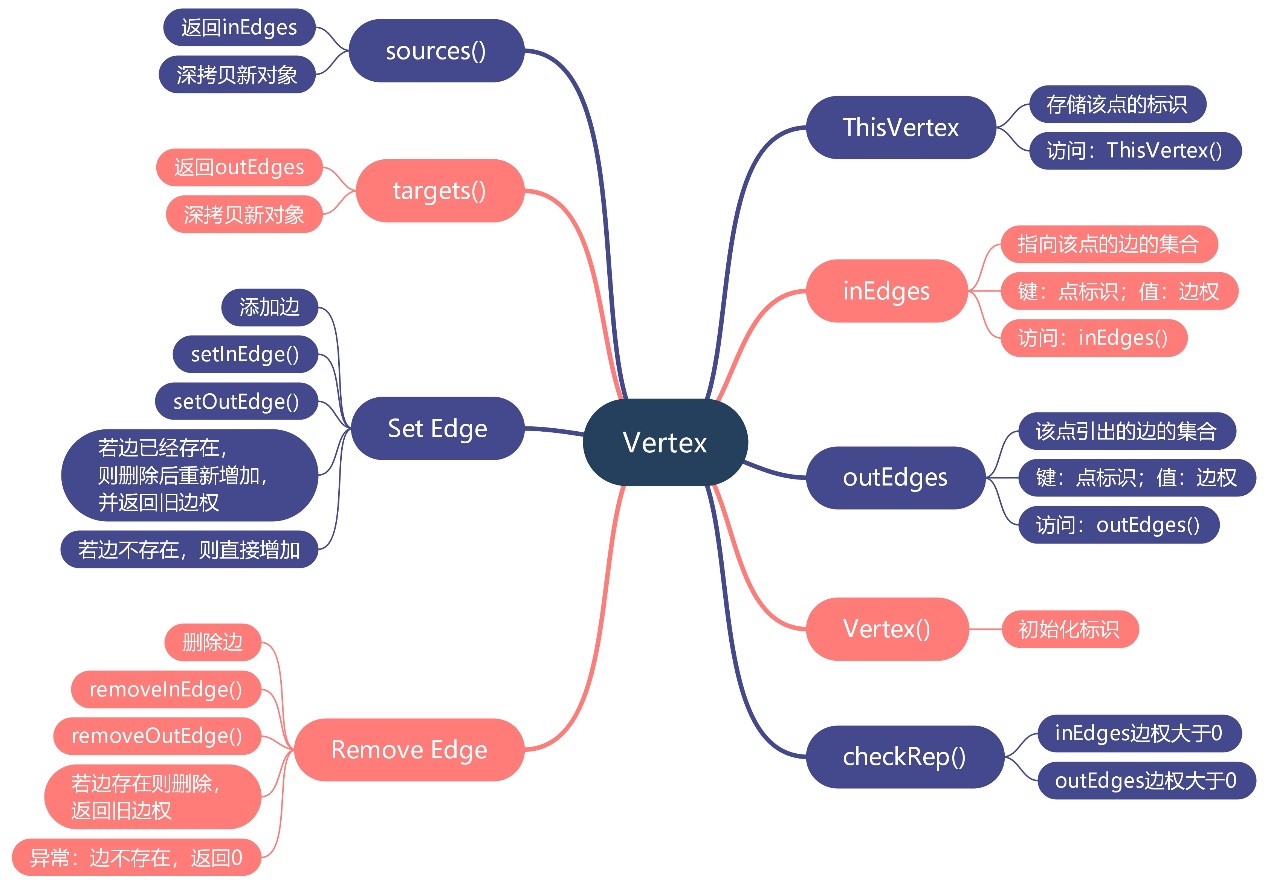
该类以Edge为基础重写Graph<L>，用集合来存储点和边（Edge），每有Edge的增加就会影响到集合的更改，而点的删除也需要在集合中查询匹配。



#### Implement ConcreteVerticesGraph

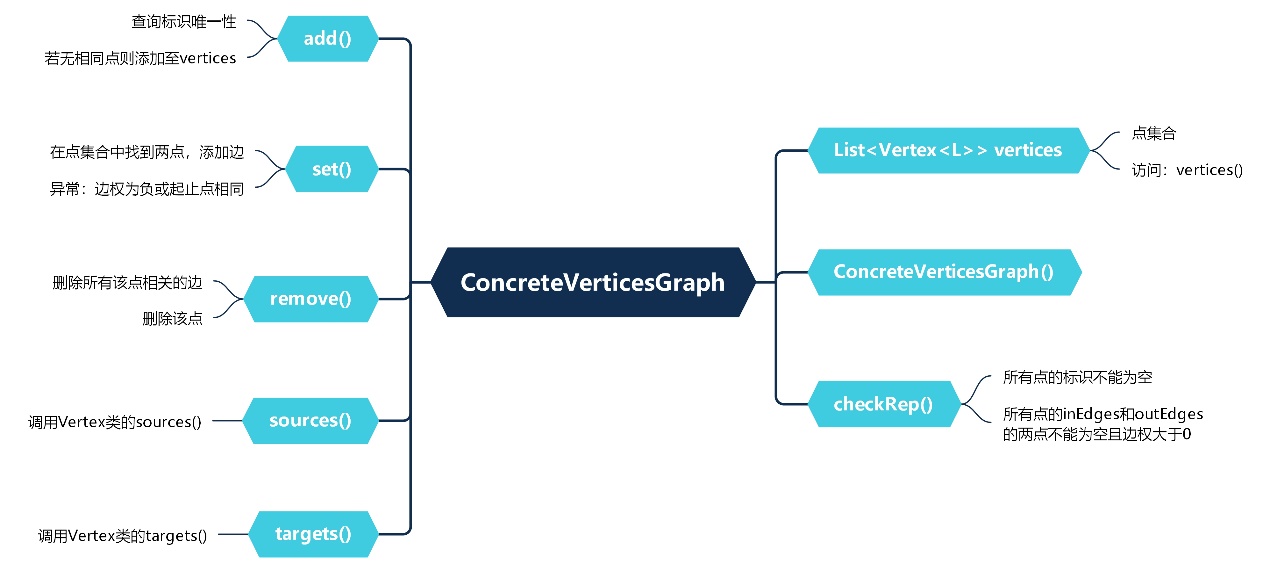
Vertex实现

Vertex是点的抽象类，包含3个信息：点的标识、指向该点的边、由该点引出的边。Vertex需要能访问这3个信息，以及增加/删除进边/出边。



ConcreteVerticesGraph实现

ConcreteVerticesGraph是以点为基础的图，每个点通过唯一的标识进行区分，set和remove都依赖与Vertex类中的添加和删除操作，sources和targets也调用了Vertex类的方法。



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

#### Make the implementations generic

在程序中选择“重构”或选择“String”并选择更改所有匹配项（要注意toString），即可实现泛化类型。

#### Implement Graph.empty()

使Graph.empty()能返回一个新的空实例。代码如下：

    public static Graph<String> empty() {

        return new ConcreteEdgesGraph();

    }

### Problem 4: Poetic walks

问题简述：

给定一个语料库corpus，根据corpus中的文本生成一个单词图，然后给定一条语句输入，在图中搜索词之间的关系，自动补全语句中可能可以完善的部分。

图的构建规则是，在corpus中，对每一个不一样的单词看作一个顶点，相邻的单词之间，建立一条有向边，相邻单词对出现的次数，作为这条有向边的权值。在输入信息补全时，对相邻单词A和B做检查，如果存在一个单词C，在图中可以由前一个单词A通过这个单词C到达单词B，那么就在A和B之间补全C，补全的优先级按照权值越大者优先。

#### Test GraphPoet

在基于预设的测试用例基础上，增加等价类划分的多种情况。

等价类划分：两个单词之间不存在连接词，两个单词之间只有一个连接词，两个单词之间有多个连接词。

此外还要注意句末的句号，测试当一个句子最后一个词是“桥”的一端。

#### Implement GraphPoet

1. 表示不变量和检查不变量

该应用中的不变量是所有的点都不为空。

1. 构造函数

用文件输入单词，String.split()分割为数组，通过String.toLowerCase()小写化。

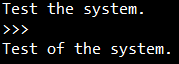
接下来构建图，相邻的单词加边。首先要在加边前通过Graph.add()加点，加边时要判断是否存在：由于Graph.set()能返回之前加的边的值，以此来判断是否存在，存在则在之前的值加一（之前的边的值保存为lastEdgeWeight）。

1. Poem(String input)

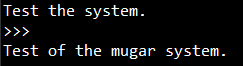
当相邻两个单词任意一个不在之前创建的图里，则将后者单词加入即可（再加个空格）当存在时，由于Bridge长度只能为2，所以：分别求两个单词的sources和targets，将该Map转换为Set求交集；若交集为空，则无桥，若交集不空，则在交集中找最短的桥（可以在Map的value中查询weight）。

#### Graph poetry slam

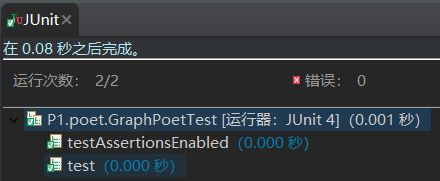
样例“This is a test of the Mugar Omni Theater sound system.”进行测试，测试成功。



修改样例为“This is a the Mugar system Omni Theater sound system test of the.”，测试成功。该样例用于测试极端情况。



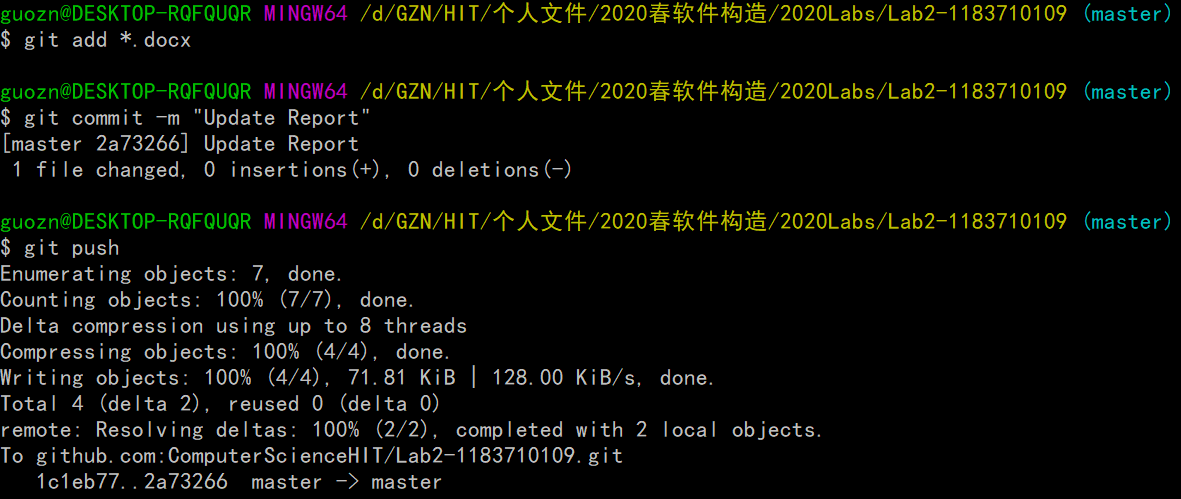
Junit测试



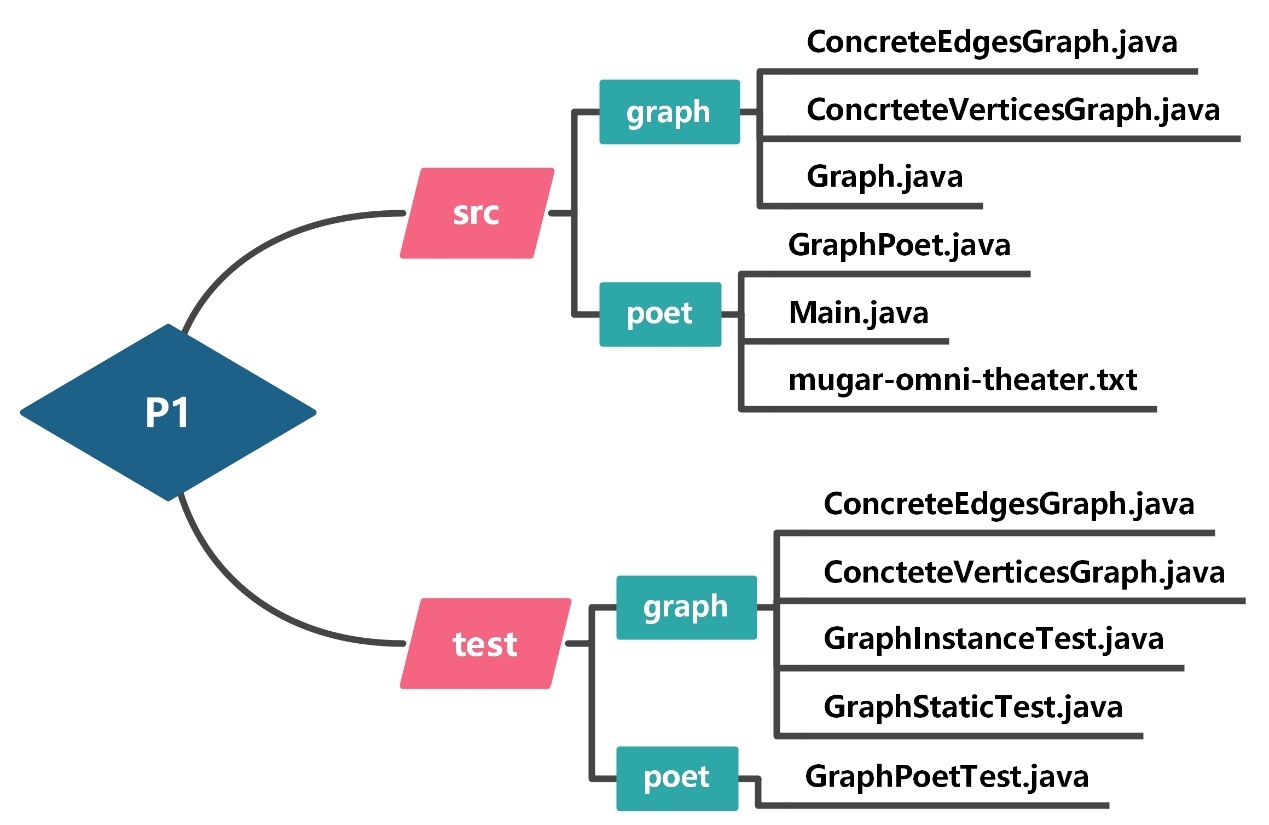
### Before you’re done

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/" \l "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。



在这里给出你的项目的目录结构树状示意图。



## Re-implement the Social Network in Lab1

这部分任务就是用我们在3.1中写的ADT，把第一次实验中的FriendshipGraph重新实现一遍，图中的节点仍然是Person类型，所以泛型L一律为Person. 而对于已经写好的FriendshipGraph中的方法，要用3.1中的Graph ADT中的方法来实现它们。

### FriendshipGraph类

Graph<Person> graph：

直接调用Graph的静态方法.empty()生成一个空的图。

boolean addVertex()：

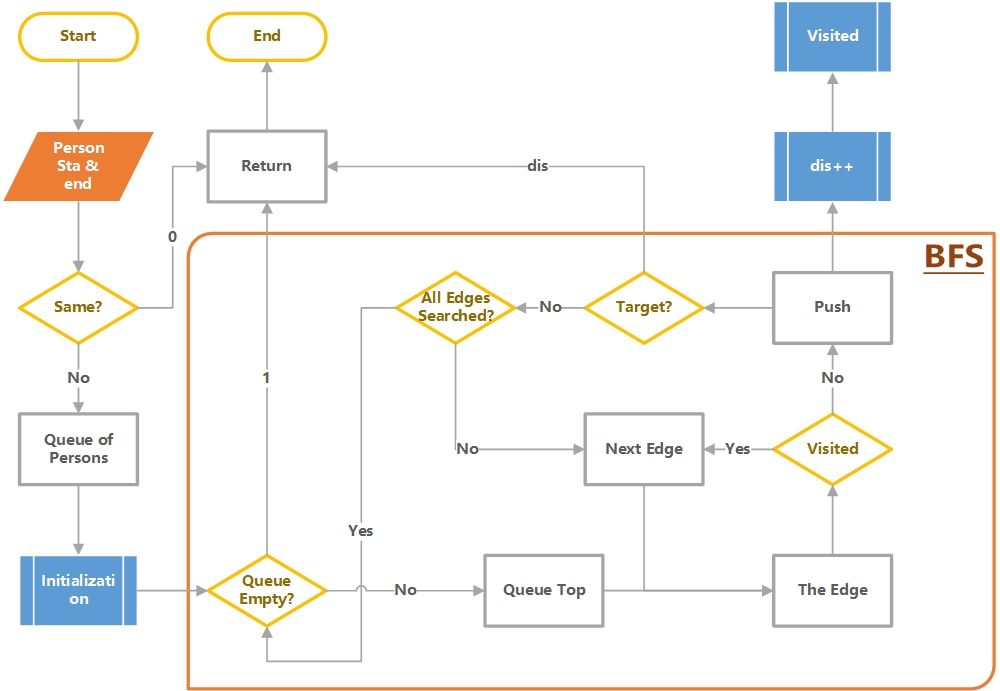
直接调用graph.add()添加点。

int addEdge()：

调用graph.set()两次，添加双向边，默认权值为1，并记录可能存在的旧边的权值。

int getDistance()：

首先判断起止点是否相等。再新建Map<Person, Integer> dis表示从起始点开始到该Person的距离，以及Map<Person, Boolean> vis表示该Person是否访问过。将两个Map初始化后，把起点标记为已经访问（所有涉及这两个Map的操作均需要remove后再put，后文不再阐述）。然后开始BFS搜索，找到终点为止。



### Person类

该类的目标是将每一个人对应到一个Person对象，并存储名字的信息。为了防止泄露，我将String Name设置为私有且不可变的。在构造函数中将Name初始化。

### 客户端main()

public class FriendshipGraphTest {

    /\*\*

     \* Basic Network Test

     \*/

    @Test

    public void Test1() {

        final FriendshipGraph graph = new FriendshipGraph();

        final Person rachel = new Person("Rachel");

        final Person ross = new Person("Ross");

        final Person ben = new Person("Ben");

        final Person kramer = new Person("Kramer");

        assertEquals(true, graph.addVertex(rachel));

        assertEquals(true, graph.addVertex(ross));

        assertEquals(true, graph.addVertex(ben));

        assertEquals(true, graph.addVertex(kramer));

        assertEquals(0, graph.addEdge(rachel, ross));

        assertEquals(1, graph.addEdge(ross, rachel));

        assertEquals(0, graph.addEdge(ross, ben));

        assertEquals(1, graph.addEdge(ben, ross));

        assertEquals(1, graph.getDistance(rachel, ross));

        assertEquals(2, graph.getDistance(rachel, ben));

        assertEquals(0, graph.getDistance(rachel, rachel));

        assertEquals(-1, graph.getDistance(rachel, kramer));

    }

    /\*\*

     \* Further Test

     \*/

    @Test

    public void Test2() {

        final FriendshipGraph graph = new FriendshipGraph();

        final Person a = new Person("A");

        final Person b = new Person("B");

        final Person c = new Person("C");

        final Person d = new Person("D");

        final Person e = new Person("E");

        final Person f = new Person("F");

        final Person g = new Person("G");

        final Person h = new Person("H");

        final Person i = new Person("I");

        final Person j = new Person("J");

        assertEquals(true, graph.addVertex(a));

        assertEquals(true, graph.addVertex(b));

        assertEquals(true, graph.addVertex(c));

        assertEquals(true, graph.addVertex(d));

        assertEquals(true, graph.addVertex(e));

        assertEquals(true, graph.addVertex(f));

        assertEquals(true, graph.addVertex(g));

        assertEquals(true, graph.addVertex(h));

        assertEquals(true, graph.addVertex(i));

        assertEquals(true, graph.addVertex(j));

        assertEquals(0, graph.addEdge(a, b));

        assertEquals(0, graph.addEdge(a, d));

        assertEquals(0, graph.addEdge(b, d));

        assertEquals(0, graph.addEdge(c, d));

        assertEquals(0, graph.addEdge(d, e));

        assertEquals(0, graph.addEdge(c, f));

        assertEquals(0, graph.addEdge(e, g));

        assertEquals(0, graph.addEdge(f, g));

        assertEquals(0, graph.addEdge(h, i));

        assertEquals(0, graph.addEdge(i, j));

        assertEquals(2, graph.getDistance(a, e));

        assertEquals(1, graph.getDistance(a, d));

        assertEquals(3, graph.getDistance(a, g));

        assertEquals(3, graph.getDistance(b, f));

        assertEquals(2, graph.getDistance(d, f));

        assertEquals(2, graph.getDistance(h, j));

        assertEquals(0, graph.getDistance(i, i));

        assertEquals(-1, graph.getDistance(d, j));

        assertEquals(-1, graph.getDistance(c, i));

        assertEquals(-1, graph.getDistance(f, h));

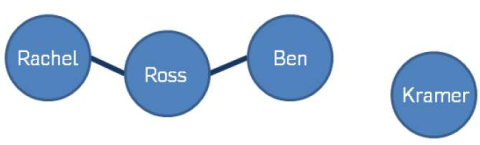
    }

}

### 测试用例

#### 简单图测试

根据题目中的社交网络图：

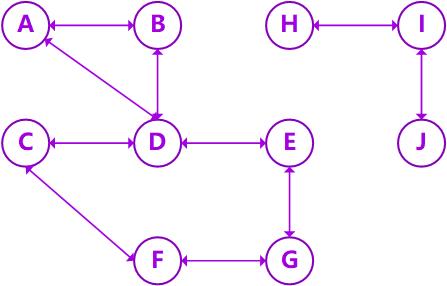


分别测试：

1. Rachel和Ross距离是1，Rachel和Ben距离是2
2. Rachel和Rachel距离是0
3. Rachel和Kramer距离是-1

#### 复杂图测试

设计10个点、10条边的社交网络图：

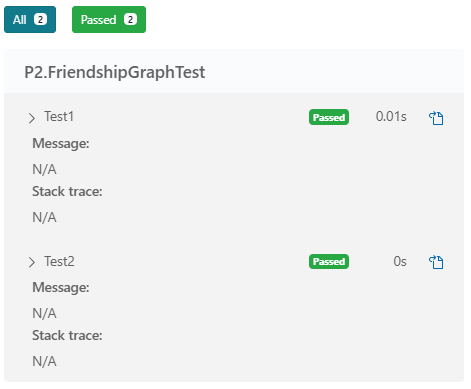


分别测试：

1. AE距离2，AD距离1，AG距离3，BF距离3，DF距离2，HJ距离2
2. II距离0
3. DJ距离-1，CI距离-1，FH距离-1

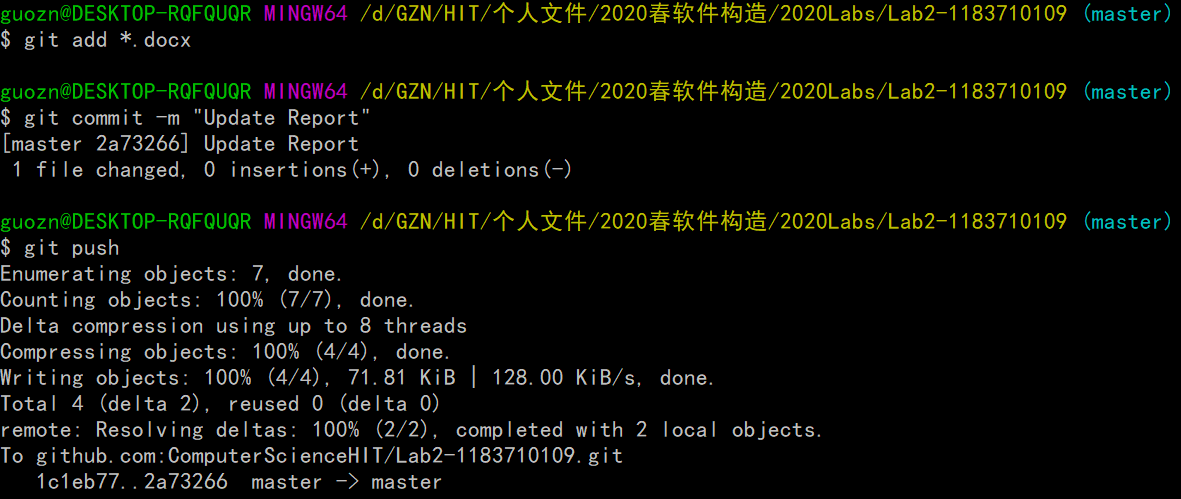
#### Junit测试结果

全部正确。

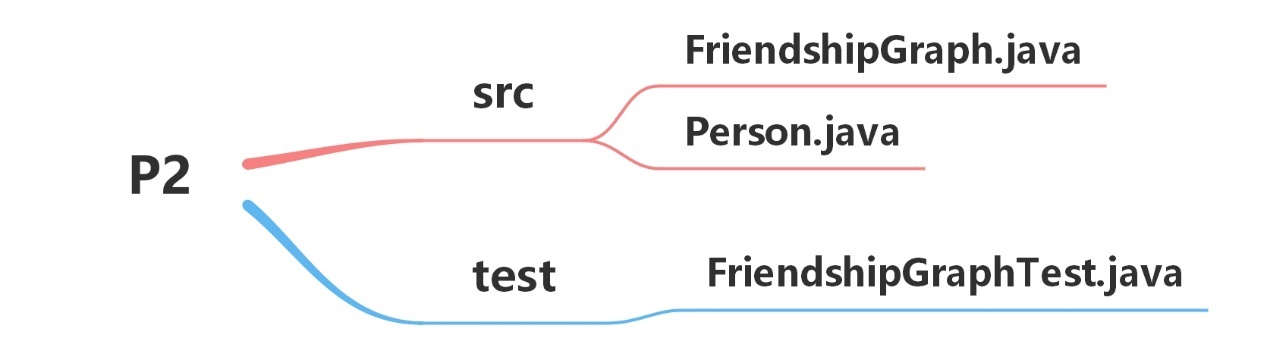


### 提交至Git仓库

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。



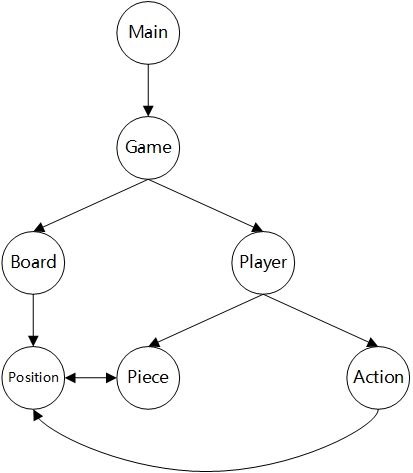
## Playing Chess

问题简述：

设计一款棋类游戏，同时支持国际象棋（Chess）和围棋（Go）。实现功能：

1. 选择游戏类型：创建Game、Board
2. 输入玩家名字：创建Player、Piece，其中Piece属于Player
3. 开始游戏，轮流选择功能
4. 放棋：给定player、piece、x、y
5. 移动棋（chess）：给定player、piece、x1、y1、x2、y2
6. 提子（go）：给定player、x、y
7. 吃子（chess）：给定player、x1、y1、x2、y2
8. 查询某个位置占用情况：给定x、y
9. 计算两个玩家分别的棋子总数
10. 跳过
11. 结束：输入“end”

整体架构



### ADT设计/实现方案

设计了哪些ADT（接口、类），各自的rep和实现，各自的mutability/ immutability说明、AF、RI、safety from rep exposure。

必要时请使用UML class diagram（请自学）描述你设计的各ADT间的关系。

#### interface Game

|  |  |
| --- | --- |
| Member | Specification |
|  |  |

#### class chessGame

#### class goGame

#### class Board

#### class Player

#### class Position

#### class Piece

#### interface Action

#### class chessAction

#### class goAction

### 主程序MyChessAndGoGame设计/实现方案

辅之以执行过程的截图，介绍主程序的设计和实现方案，特别是如何将用户在命令行输入的指令映射到各ADT的具体方法的执行。

### ADT和主程序的测试方案

介绍针对各ADT的各方法的测试方案和testing strategy。

介绍你如何对该应用进行测试用例的设计，以及具体的测试过程。

# 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 |
| 2020-03-09 | 晚上 | 初始化项目 | 完成 |
| 2020-03-10 | 中午 | Problem1 3.1.1-3.1.2 | 完成 |
| 2020-03-10 | 晚上 | Problem1 3.1.3.1 Edge Graph | 完成 |
| 2020-03-11 | 晚上 | Problem1 3.1.3.2 Vertex Graph | 完成 |
| 2020-03-12 | 下午 | 通过Graph Instance Test | 完成 |
| 2020-03-12 | 晚上 | Problem1 3.1.5 Graph Poet | 完成 |
| 2020-03-13 | 上午 | Problem1 3.1.5 Test | 完成 |
| 2020-03-13 | 上午 | Problem2 3.2整体完成 | 完成 |
| 2020-03-13 | 晚上 | Problem3 设计框架 写AF&RI | 完成 |
| 2020-03-14 | 上午 | Problem3 完成框架 | 完成 |
| 2020-03-14 | 下午 | 实现Action接口、Player、Board、Position、Piece具体功能 | 完成 |
| 2020-03-14 | 晚上 | 完善上述功能，修改bug | 完成 |
| 2020-03-15 | 上午 | 实现Game接口、chessAction、goAction功能 | 完成 |
| 2020-03-15 | 晚上 | 实现chessGame、goGame功能，调试测试用例 | 完成 |
| 2020-03-16 | 晚上 | 通过chessGame测试 | 完成 |
| 2020-03-17 | 下午 | 通过goGame测试 | 完成 |
| 2020-03-17 | 晚上 | 验收完成 | 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 |
|  |  |
|  |  |
|  |  |

# 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

## 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？
2. 使用泛型和不使用泛型的编程，对你来说有何差异？
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？
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？
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？
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？
7. 关于本实验的工作量、难度、deadline。
8. 《软件构造》课程进展到目前，你对该课程有何体会和建议？