

**2021年春季学期  
计算学部《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 陈广焕 |
| 学号 | 1190501614 |
| 班号 | 1903006 |
| 电子邮件 | 2718458514@qq.com |
| 手机号码 | 15778541719 |

**目录**

[1 实验目标概述 1](#_Toc74243300)

[2 实验环境配置 1](#_Toc74243301)

[2.1 安装EclEmma 1](#_Toc74243302)

[3 实验过程 2](#_Toc74243303)

[3.1 Poetic Walks 2](#_Toc74243304)

[3.1.1 Get the code and prepare Git repository 3](#_Toc74243305)

[3.1.2 Problem 1: Test Graph <String> 3](#_Toc74243306)

[3.1.3 Problem 2: Implement Graph <String> 4](#_Toc74243307)

[3.1.3.1 Implement ConcreteEdgesGraph 4](#_Toc74243308)

[3.1.3.2 Implement ConcreteVerticesGraph 5](#_Toc74243309)

[3.1.4 Problem 3: Implement generic Graph<L> 7](#_Toc74243310)

[3.1.4.1 Make the implementations generic 7](#_Toc74243311)

[3.1.4.2 Implement Graph.empty() 7](#_Toc74243312)

[3.1.5 Problem 4: Poetic walks 7](#_Toc74243313)

[3.1.5.1 Test GraphPoet 8](#_Toc74243314)

[3.1.5.2 Implement GraphPoet 8](#_Toc74243315)

[3.1.5.3 Graph poetry slam 8](#_Toc74243316)

[3.1.6 使用Eclemma检查测试的代码覆盖度 9](#_Toc74243317)

[3.1.7 Before you’re done 10](#_Toc74243318)

[3.2 Re-implement the Social Network in Lab1 11](#_Toc74243319)

[3.2.1 FriendshipGraph类 11](#_Toc74243320)

[3.2.2 Person类 11](#_Toc74243321)

[3.2.3 客户端main() 11](#_Toc74243322)

[3.2.4 测试用例 13](#_Toc74243323)

[3.2.5 提交至Git仓库 14](#_Toc74243324)

[4 实验进度记录 15](#_Toc74243325)

[5 实验过程中遇到的困难与解决途径 15](#_Toc74243326)

[6 实验过程中收获的经验、教训、感想 15](#_Toc74243327)

[6.1 实验过程中收获的经验和教训 15](#_Toc74243328)

[6.2 针对以下方面的感受 15](#_Toc74243329)

# 实验目标概述

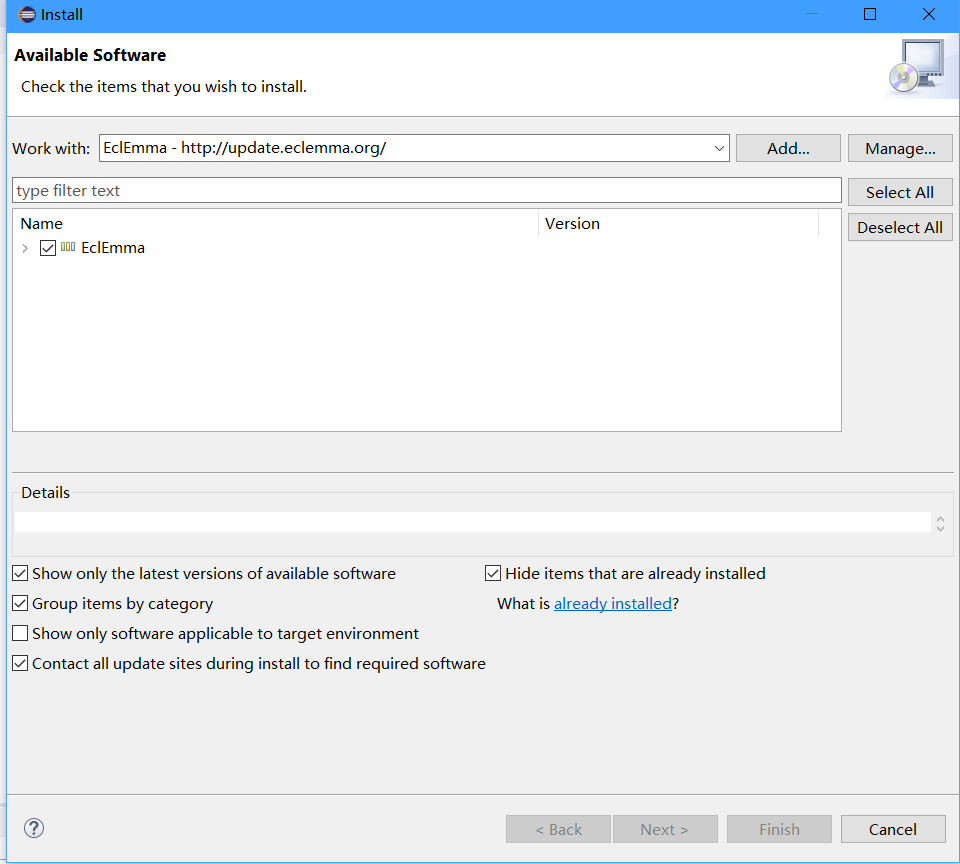
本次实验训练抽象数据类型（ADT）的设计、规约、测试，并使用面向对象编程（OOP）技术实现ADT。具体来说：

* 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。

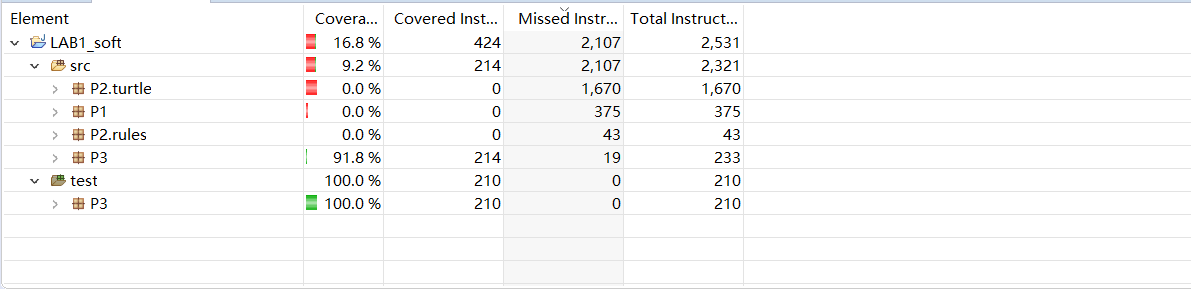
# 实验环境配置

## 安装EclEmma

打开eclipse，在菜单栏中选择Help，然后点击Install New Software,在打开的对话框中输入安装网址，选择EclEmma，点击下一步即可（这里之前已经装好了，所以无法点击下一步）



安装效果：（这里用第一个实验的工程文件演示）



在这里给出你的GitHub Lab2仓库的URL地址（Lab2-学号）

https://github.com/ComputerScienceHIT/HIT-Lab2-1190501614

# 实验过程

## Poetic Walks

该任务主要是通过实现Graph接口类中的方法并可额外增加一些方法来完成Graph的基本设计，然后通过Graph来实现GraphPoet类。该任务着重于抽象数据类型的训练，包括设计、实现、测试；同时注重不可变类型和可变类型的规范

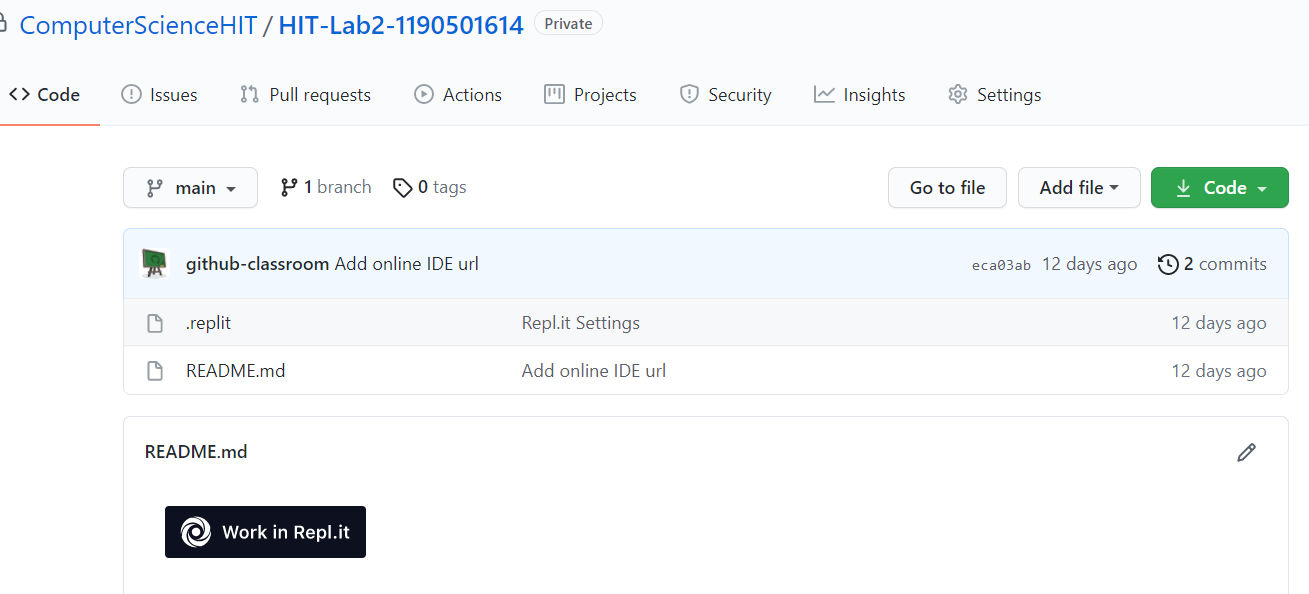
### Get the code and prepare Git repository

由于网速问题，这里通过老师给出的另一条链接

<https://github.com/rainywang/Spring2020_HITCS_SC_Lab2/tree/master/P1>

去Github直接下载压缩包即可

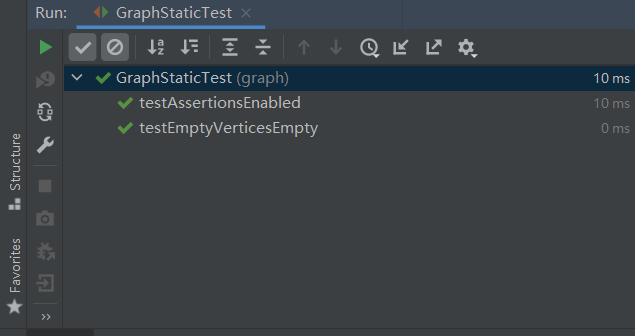
然后通过访问GitHub Classroom的URL地址建立仓库即可



### Problem 1: Test Graph <String>

测试静态方法生成String类型的Graph。

测试结果：



### Problem 2: Implement Graph <String>

通过重写Graph接口中的方法并额外增加一些方法来实现Graph<string>的基本功能

#### Implement ConcreteEdgesGraph

1. **ConcreteEdgesGraph实现**

* **void checkRep()**
* for (L vertex : vertices)
* assert (vertex != null);
* for (Edge<L> edge : edges)
* assert (edge != null);
* **boolean add(L vertex)**

**如果vertices包含点vertex，返回false，否则添加vertex，返回true**

* **int set(L source, L target, int weight)**

**如果weight<0，抛出异常；如果vertices不包含source或target则添加source或target到vertices中；如果source和target的值相同，则返回0；然后使用迭代器遍历edges，找到边(source,target)，如果weight>0,将其值改为weight，返回这条边之前的weight，否则remove这条边，返回0；如果找不到这条边并且weight大于0，则添加这条边，返回0，否则直接返回0。**

* **boolean remove(L vertex)**
* if (!vertices.contains(vertex))
* return false;
* edges.removeIf(edge -> edge.source().equals(vertex) || edge.target().equals(vertex));
* vertices.remove(vertex);
* checkRep();
* return true;
* **Set<L> vertices()**
* return new HashSet<>(vertices);
* **Map<L, Integer> sources(L target)**

**遍历edges，如果这条边的target等于参数target，则将这条边的source和weight放进一个Map<L,Integer>的容器中，最后返回这个容器**

* **Map<L, Integer> targets(L source)**

**同上，和Map<L, Integer> sources(L target)类似**

* **String toString()**

**返回关于vertices和edges的基本信息的字符串**

1. **Edge实现**

* **Edge(L source, L target, int weight) //构造函数**
* this.source = source;
* this.target = target;
* this.weight = weight;
* checkRep();
* **void checkRep()**
* assert (weight > 0 && !source.equals(target));
* **int getWeight()**
* return this.weight;
* **L getSource()**
* return this.source;
* **L getTarget()**
* return this.target;
* **boolean equaledge(L source,L target)**
* if(source.equals(this.source) && target.equals(this.target))
* return true;
* return false;
* **String toString()**

**返回source，target，weight的基本信息**

#### Implement ConcreteVerticesGraph

1. ConcreteVerticesGraph实现

* **void checkRep()**

**检查vertices中的每个点的source和target的key和value**

* **boolean add(L vertex)**

**遍历vertices，如果其中有一个点的ver等于参数vertex，则返回false，否则new Vertex<L>(vertex),并将其加入到vertices，返回true**

* **int set(L source, L target, int weight)**

**首先检查weight是否小于0，小于0则抛出异常；然后检查vertices是否包含source和target；接下来检查source和target是否相同；然后遍历vertices找到ver等于的source的Vertex和ver等于target的Vertex，然后分别设置出边表和入边表，返回之前的边的weight**

* **boolean remove(L vertex)**

**通过迭代器遍历vertices，找到其值等于参数Vertex的点，然后遍历更改与这个点有关的出边和入边，最后移除这个点**

* **Set<L> vertices()**

**返回vertices的一个副本即可**

* **Map<L, Integer> sources(L target)**

**放回以target为终点的边的<source，weight>的容器,简单遍历即可**

* **Map<L, Integer> sources(L source)**

**同上，基本类似**

* **String toString()**

**放回一个关于vertices大小信息的字符串**

1. **Vertex**

* **构造函数**
* Vertex(L vertex){ this.ver=vertex; }
* **void checkRep()**
* for (L key : inEdges.keySet())
* assert (inEdges.get(key) > 0);
* for (L key : outEdges.keySet())
* assert (outEdges.get(key) > 0);
* **L getVer()**
* return this.ver;
* **Map<L, Integer> getsources()**
* Map<L, Integer> sources = new HashMap<>();
* sources.putAll(inEdges);
* return sources;
* **Map<L, Integer> gettargets()**
* Map<L, Integer> targets = new HashMap<>();
* targets.putAll(outEdges);
* return targets;
* **int set\_outEdge(L target, int weight)**

**设置出边，遍历outEdges找到出边表中key等于target的元素，然后更改weight，返回之前的weight值。注意检查weight是否为正数**

* **int setInEdge(L source, int weight)**

**设置入边，同上类似**

* **int removeInEdge(L source)**

**判断入边表中的key值是否包含source，后直接调用内置函数移除即可**

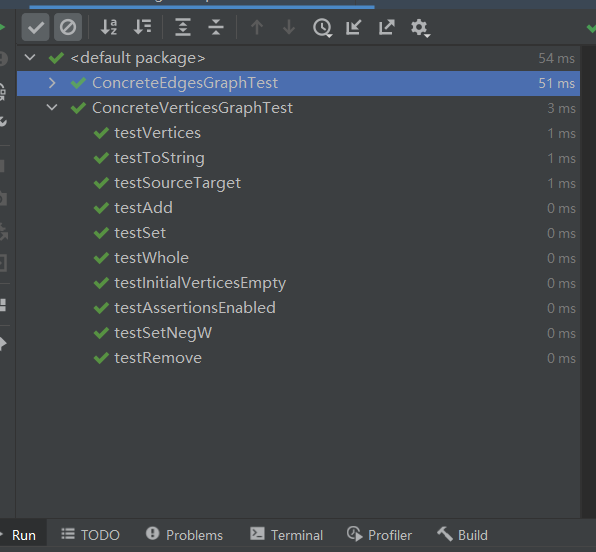
* **int removeOutEdge(L target)**

**同上，类似**

* **String toString()**

**返回ver的信息**

**Junit测试结果：**

****

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

#### Make the implementations generic

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

#### Implement Graph.empty()

**public static Graph<String> empty() {**

**return new ConcreteEdgesGraph();**

**}**

### Problem 4: Poetic walks

根据给定文本中的信息生成一个单词图，然后给定一条输入，在图中搜索词之间的关系，自动补全语句。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，这条有向边的权值等于相邻单词对出现的次数。在输入信息时，检查相邻单词A和B，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

在基本测试之后，增加特殊的测试用例和边界检查，包括单连接，多连接，孤立点以及结尾的符号

#### Implement GraphPoet

1. **void checkRep()**

**检查vertices中的每一点vertex是否非空**

1. **GraphPoet(File corpus)**

**根据文件内容构造单词关系图，首先读入文件，然后分割字符串，将字符串传给graph，然后调用函数设置单词之间的边**

1. **String poem(String input)**

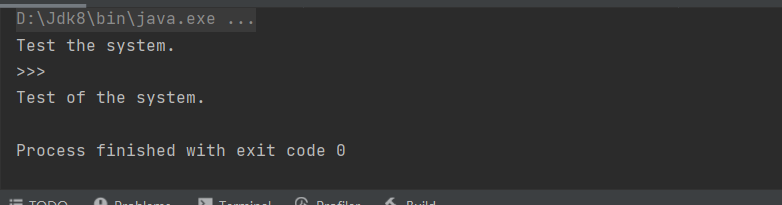
**根据输入内容完善原来的单词图，首先分割输入的字符串，然后判断单词是否已经存在，若不存在，直接加入；若存在，则调用target和source函数找到其中的点的信息，然后修改**

1. **String toString()**

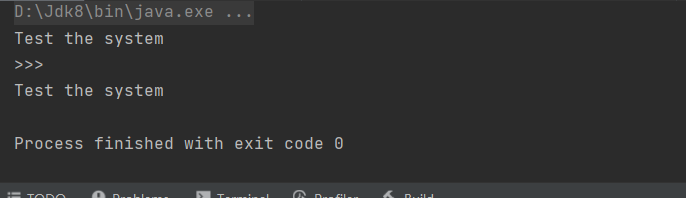
**return "an instance of GraphPoet, hashcode: " + this.hashCode();**

#### Graph poetry slam

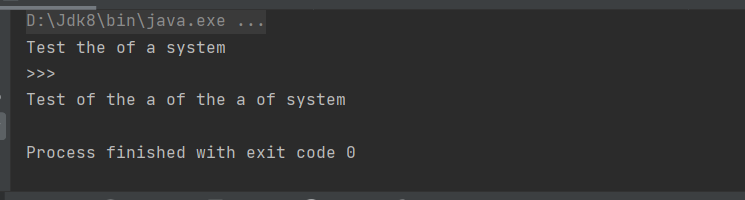
文件样例测试：



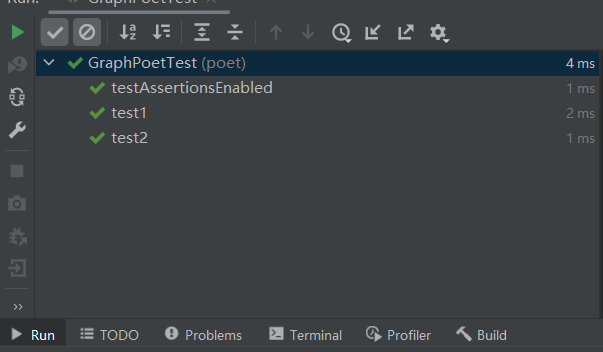
边界检查：This is a system Omni Theater native system test of.（去掉所有的the）



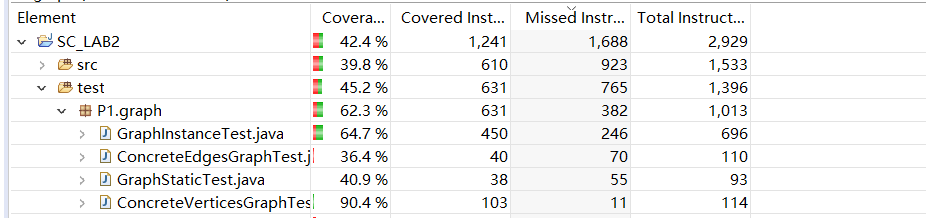
测试用例2：This is the a of system Omni Theater native system test of the.（制造回路）

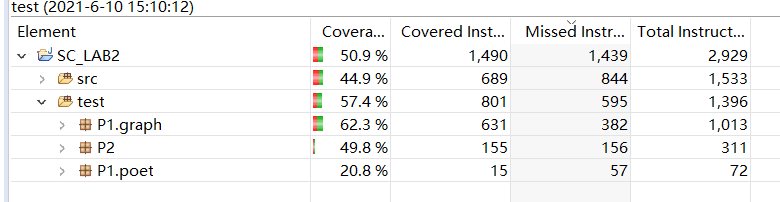


Junit测试：

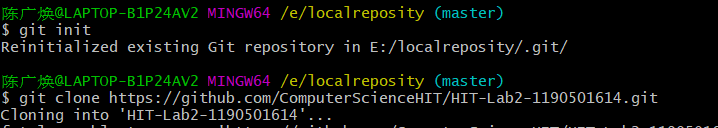


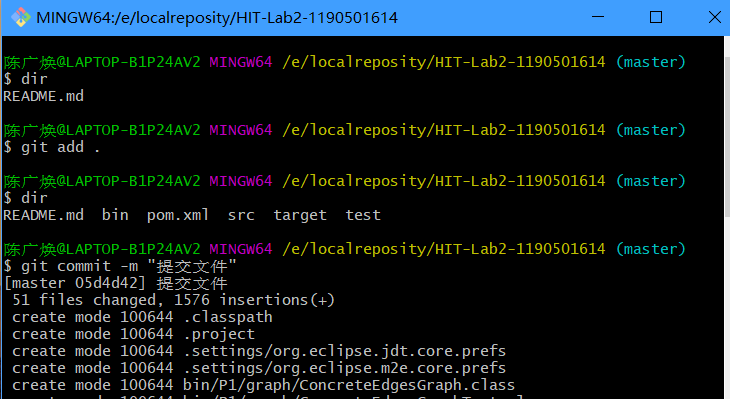
### 使用Eclemma检查测试的代码覆盖度

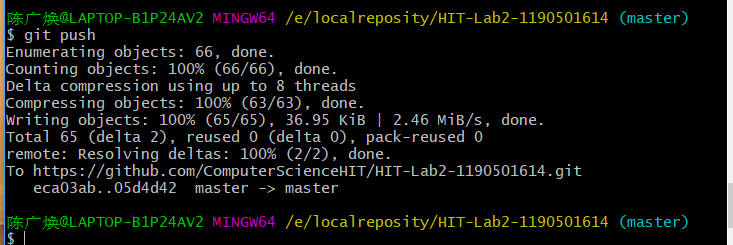




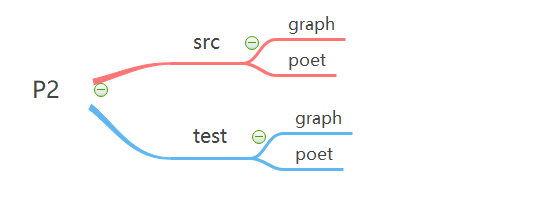
### Before you’re done







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

该任务要求我们用P1的ADT把第一次实验中的FriendshipGraph重新实现一遍，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

1. **private final Graph<Person<L>> friendshipgraph = Graph.empty();**

**初始化一个为空的graph**

1. **boolean addVertex(Person<L> person)**

return friendshipgraph.add(person);

1. **int addEdge(Person<L> a, Person<L> b)**

调用graph.set()两次，添加双向边，默认权值为1，并返回旧边的权值。

1. **int getDistance(Person<L> source, Person<L> target)**

**用dijkstra算法即可（由于情况特殊，也可以用广度优先搜索）**

### Person类

该类每一个人对应到一个Person对象，并存储名字的信息。为了保护信息被破坏，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 {

    /\*\*

     \* esstensial Network Test

     \*/

    @Test

    public void teste1() {

        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(ben, ross));

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

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

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

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

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

    }

    /\*\*

     \* flexible Test

     \*/

    @Test

    public void testf2() {

        final FriendshipGraph graph = new FriendshipGraph();

        final Person a = new Person("1");

        final Person b = new Person("2");

        final Person c = new Person("3");

        final Person d = new Person("4");

        final Person e = new Person("5");

        final Person f = new Person("6");

        final Person g = new Person("7");

        final Person h = new Person("8");

        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(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(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));

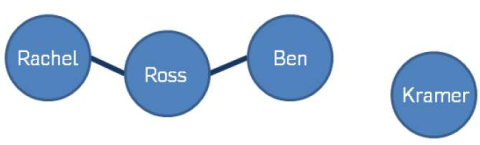
        assertEquals(-1, graph.getDistance(f, h));

    }

}

### 测试用例

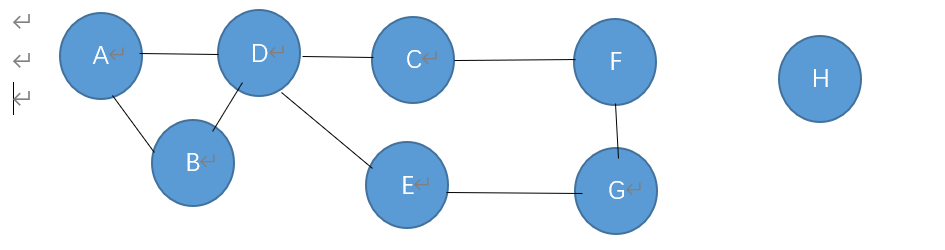
Ppt提供的简单图测试：

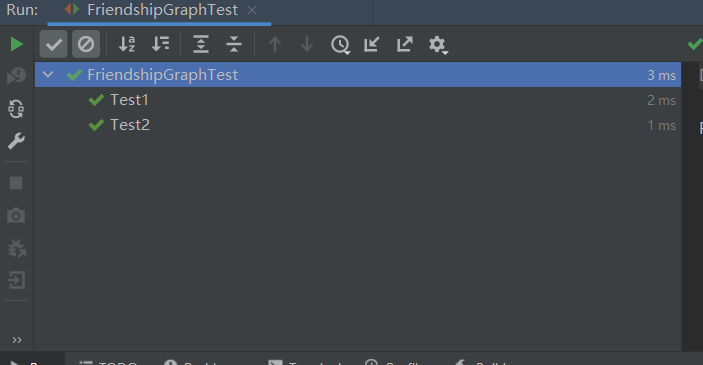


测试内容：

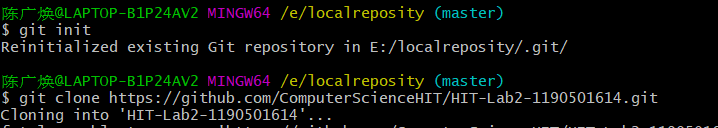
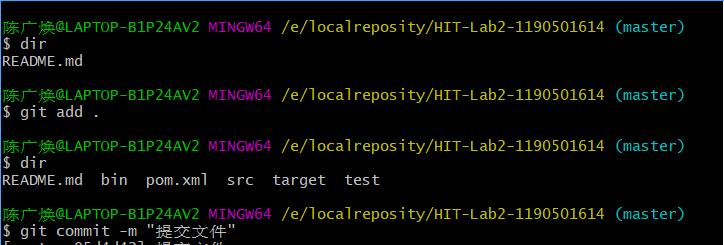
1. Rachel和Ross距离是1，Rachel和Ben距离是2
2. Rachel和Rachel距离是0
3. Rachel和Kramer距离是-1

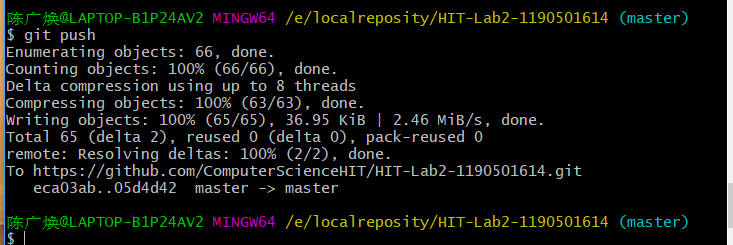
复杂图测试：

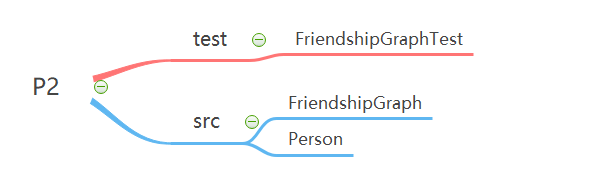




### 提交至Git仓库





# 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。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 6-06 | 下午、晚上 | 完成P1graph | 完成 |
| 6-07 | 晚上 | 完成P1.poet | 未完成 |
| 6-08 | 下午、晚上 | 完成P1和测试部分 | 完成 |
| 6-9 | 下午 | 完成P2构建 | 完成 |
| 6-10 | 下午 | 完成全部 | 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 |
| 用set的函数的代码实现出现bug | 使用contains函数  网上查找资料，使用Map的容器的一些内置函数 |
| 用Dijstra算法实现求路径出现问题 | 改用广度优先搜索 |

# 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

要认真写spec很重要，包括注释

学会编写测试用例，检验程序正确性

## 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的编程更加抽象

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

没有太多差异

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？

能把RI和AF考虑全面，方便后面编程

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？

减低开发的时间成本

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？

减少程序出现错误

1. 关于本实验的工作量、难度、deadline。

还行

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

无