2022 年春季学期

计算学部《软件构造》课程

Lab 2 实验报告

姓名	王炳轩
学号	120L022115
班号	2003007
电子邮件	1487819688@qq.com
手机号码	13552161805

目录

1 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。

2 实验环境配置

本次实验需要在 Eclipse IDE 中安装配置 EclEmma（一个用于统计 JUnit 测试用例的代码覆盖度的 plugin）。

遵循指引进行配置：下载、解压到目录。

For manual installation please download the latest EclEmma release. Unzip the archive into `dropins` folder of your Eclipse installation and restart Eclipse:

```
<your eclipse installation>/
+- dropins/
    +- eclemma-x.y.z/
        +- plugins/
        |   +- ...
```

+- feature/

+- ...

在这里给出你的 GitHub Lab2 仓库的 URL 地址（Lab2-学号）：https://github.com/ComputerScienceHIT/HIT-Lab2-120L022115

3 实验过程

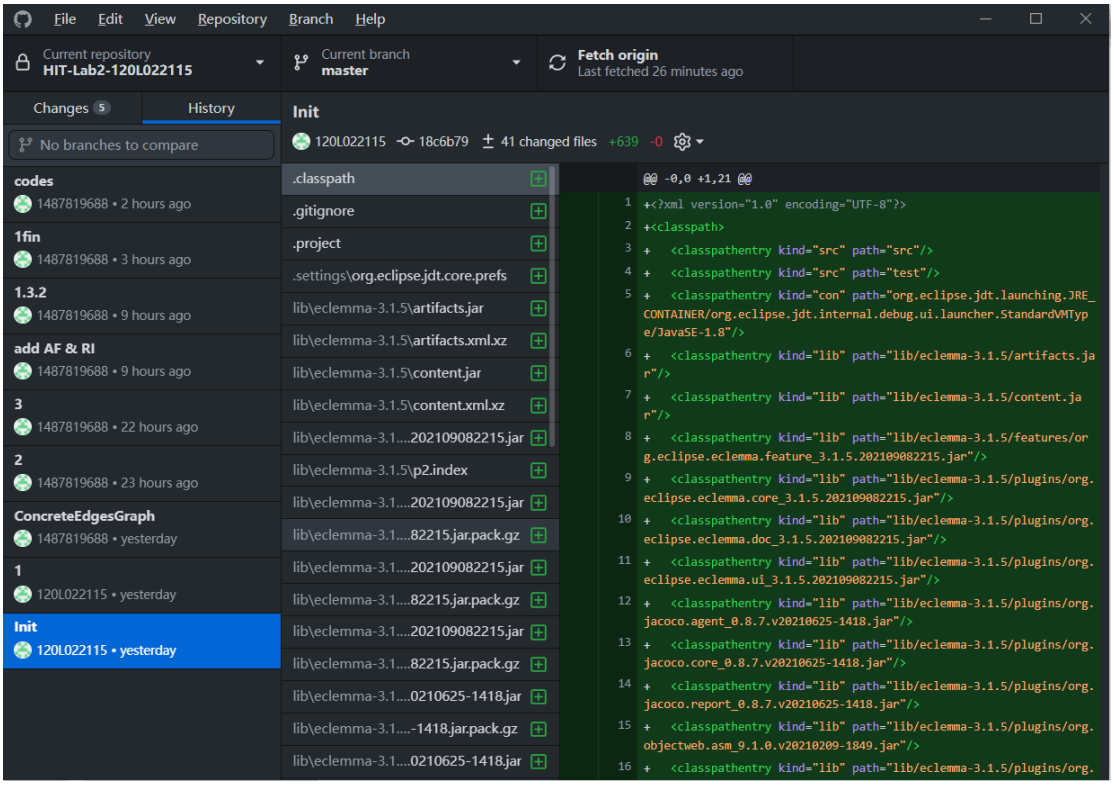
请仔细对照实验手册，针对三个问题中的每一项任务，在下面各节中记录你的实验过程、阐述你的设计思路和问题求解思路，可辅之以示意图或关键源代码加以说明（但千万不要把你的源代码全部粘贴过来！）。

3.1 Poetic Walks

Java Collections 框架为处理对象集合提供了许多有用的数据结构：列表、映射、队列、集合等等。它不提供图形数据结构。因此我们实现一个 Graph。

3.1.1 Get the code and prepare Git repository

如何从 GitHub 获取该任务的代码、在本地创建 git 仓库、使用 git 管理本地开发。



使用 git clone 克隆代码，然后 push 到自己的仓库中。

3.1.2 Problem 1: Test `Graph <String>`

以下各部分，请按照 MIT 页面上相应部分的要求，逐项列出你的设计和实现思路/过程/结果。

Writing Testing strategy & Test Case:

```
// Testing strategy
//   build a graph and add vertices and edges to it
//   then using some method to compare equals or not
//   if equals then test passed.
//   empty()
//     no inputs, only output is empty graph
//     observe with vertices()
```

测试用例已经写到 GraphStaticTest.java

3.1.3 Problem 2: Implement `Graph <String>`

以下各部分，请按照 MIT 页面上相应部分的要求，逐项列出你的设计和实现思路/过程/结果。

- **Document the abstraction function and representation invariant**.
- Along with the rep invariant, **document how the type prevents rep exposure**.
- **Implement `checkRep`** to check the rep invariant.
- **Implement `toString`** with a useful human-readable representation of the abstract value.

3.1.3.1 Implement `ConcreteEdgesGraph`

For `ConcreteEdgesGraph`, I use the rep provided:

```
private final Set<String> vertices = new HashSet<>();
private final List<Edge> edges = new ArrayList<>();
```

I not add fields to the rep or choose not to use one of the fields.

(1)Complete Test strategy

```
//    Testing strategy for Edge
// toString()
//        no inputs, only output is String
//        evaluate with equals()
// equals()
```

```
//        input another Edge, output is if or not equal.
// Testing strategy for ConcreteEdgesGraph.toString()
// build a graph, to string, then evaluate equals.
```

(2)按步骤完成实现

a.先完成 Edge 类

For class `Edge`, define the specification and representation; however, `Edge` must be **immutable**.

i. 写 AF\RI\SRE

```
// Abstraction function:
//   AF({start,end,weight}) = a edge in graph
//      start: the start vertex of the edge
//      end: the end vertex of the edge
//      weight: the weight of the edge

// Representation invariant:
//   the weight is a positive integer

// Safety from rep exposure:
//   all fields are private.
//   all returns are immutable.
```

ii.写各类方法实现

Then proceed to implement `Edge` and `ConcreteEdgesGraph`.

```
/**
 * get the start Vertex of this edge.
 * @return the start Vertex
 */
1、public L start()
/**
 * get the end Vertex of this edge.
 * @return the end Vertex
 */
2、public L end()
/**
 * get the weight of this edge.
 * @return the weight
 */
3、public int weight()
4、// constructor
/**
 * Create a Edge between two Vertex.
 * @param start
 * @param end
 */
public Edge(L start,L end,int weight)
```

```
5、// checkRep
private boolean checkRep()
/**
 * evaluate equals
 * @param p the vertex to compare
 * @return equals or not
 */
6、public boolean equals(Edge<L> e)
/**
 * to String
 * convert the edge to string.
 * formatted with a--w->b.
 * @return string
 */
7、@Override public String toString()
```

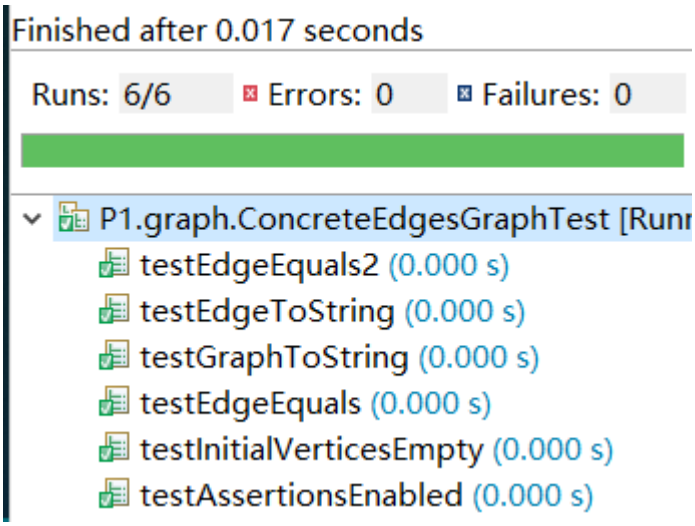
b.再完成 EdgeGraph 要求的方法

constructor、checkRep、add、set、remove、source、target、toString

(3) 再写 AF、RI、RE Safety

```
// Abstraction function:
//   Vertex in Vertices, edge in edges,
//   to describe a graph.
//   AF(vertices,edges) = a graph which has vertices and edges
//   the vertices are a set of vertices.
//   the edges are a list of edges.

// Representation invariant:
//   no Vertex of edges is not in vertices.
//   no two Vertices has the same name.

// Safety from rep exposure:
//   all fields are private.
//   all returns are immutable.
//  except the Map but it has deal with Defensive Coping.
```

(4) Run the tests by right-clicking on `ConcreteEdgesGraphTest.java` and selecting *Run As* → *JUnit Test*.



(5) 最后 commit & push

3.1.3.2 Implement `ConcreteVerticesGraph`

For `ConcreteVerticesGraph`, I use the rep provided:

```
private final List<Vertex> vertices = new ArrayList<>();
```

I not add fields to the rep.

（1）先完成 Test strategy

```
// Testing strategy for Vertex
// toString()
//        no inputs, only output is String
//        evaluate with equals()
// equals()\addIn()\addOut()\addInOut()
//        input Vertexs and edges, output is if or not equal.

//   tests for operations of Edge
```

（2）按步骤完成实现

a.先完成 Vertex 类

For class `Vertex`, I define the specification and representation; however, `Vertex` must be **mutable**.

i.写 AF\RI\SRE

```
// Abstraction function:
//     in<OtherVertex,0> -> a name Vertex to OtherVertex has a 0 weight edge.
//     out<OtherVertex,0> -> a OtherVertex to name Vertex has a 0 weight edge.
//
// Representation invariant:
//     true
// Safety from rep exposure:
```

// all fields are private.
// all returns are immutable, besides the L due to the user using.

ii.写各类方法实现

```
/**
 * get the name of this Vertex.
 * @return the name of this vertex
 */
1、public L name()
/**
 * get the in edge to this Vertex.
 * @return in edges of this vertex
 */
2、public Map<L,Integer> in()
/**
 * get the out edge to this Vertex.
 * @return out edges of this vertex
 */
3、public Map<L,Integer> out()
4、// constructor
/**
 * Create a Vertex.
 * @param name of the vertex
 */
public Vertex(L name)
5、// checkRep
private boolean checkRep()
/**
 * delete a vertex and its relative edges
 * @param p the deleted vertex
 */
6、public void synDelVertex(L v)
/**
 * delete this Vertex
 * @return the set of relative Vertices
 */
7、public Set<L> delVertex()
/**
 * add or update in edges
 * @param p the parent vertex
 * @param w weight
 * @return previous weight or 0 if not exists before add.
 */
8、public int addIn(L p,int w)
/**
```

```
 * add or update out edges
 * @param p the child vertex
 * @param w weight
 * @return previous weight or 0 if not exists before add.
 */
9、public int addOut(L p,int w)
/**
 * add both out and in edge
 * @param p vertex
 * @param w weight
 */
10、public void addInOut(L p,int w)
/**
 * remove out edge
 * @param t the target vertex
 * @return the weight if exists
 */
11、public int removeEdgeTarget(L t)
/**
 * remove in edge
 * @param s the source vertex
 * @return the weight if exists
 */
12、public int removeEdgeSource(L s)
/**
 * evaluate equals
 * @param p the parent vertex
 * @return equals
 */
13、public boolean equals(Vertex<L> v)
/**
 * to String
 * formatted with name1--weight->name2
 * @return string
 */
14、// toString()
@Override public String toString()
```

b.再完成 VertexGraph 的各种方法

constructor

checkRep

add

set

remove

source

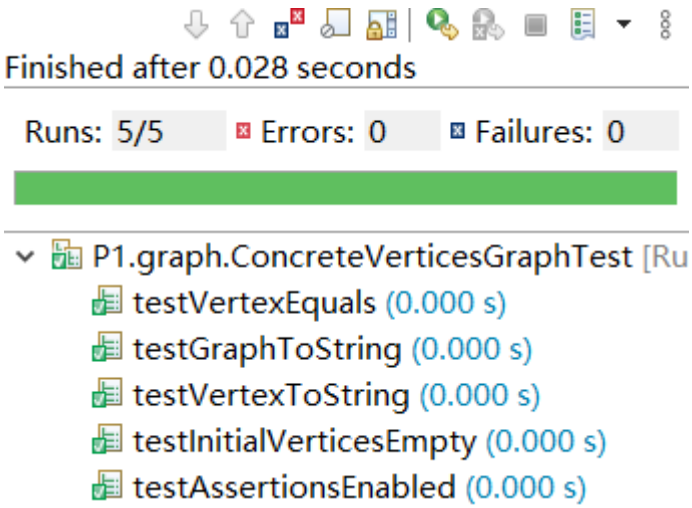
target

toString

（3）再写 AF、RI、RE Safety

```
// Abstraction function:
//   AF(vertices) = a graph.
//   vertices[i]={a--0->b} -> a to b as a 0 weight edge.
//
// Representation invariant:
//   no two Vertices has a same name.
//
// Safety from rep exposure:
//   all fields are private.
//   all returns are immutable.
//   except the Set but it has deal with Defensive Coping.
```

（4） Run the tests by right-clicking on `ConcreteVerticesGraphTest.java` and selecting *Run As* → *JUnit Test*.



(5) commit and push to github

3.1.4 Problem 3: Implement generic `Graph<L>`

3.1.4.1 Make the implementations generic

把所有除了 toString 的 String 改成 L。
并为所有的集合类添加/修改泛型。

```
public class ConcreteEdgesGraph<L> implements Graph<L> { ... }
class Edge<L> { ... }
public class ConcreteVerticesGraph<L> implements Graph<L> { ... }
class Vertex<L> { ... }
```

Update both of your implementations to support any type of vertex label, using placeholder `L` instead of `String`.

`Edge` or `List<Edge>` become `Edge<L>` and `List<Edge<L>>`.

`new ConcreteEdgesGraph()` or `new Edge()` become `new ConcreteEdgesGraph<String>()` and `new Edge<L>()`

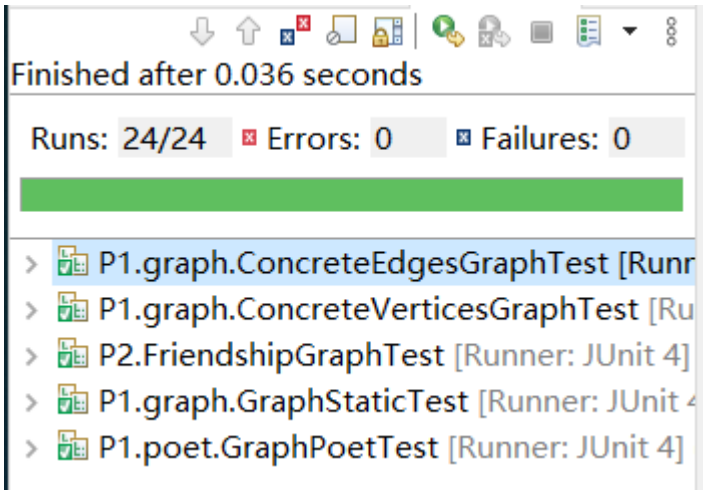
3.1.4.2 Implement `Graph.empty()`

默认选取 ConcreteEdgeGraph 实现方法来构造空图。

```
public static <L> Graph<L> empty() {
        return new ConcreteEdgesGraph<L>();
    }
```

At this point, all of my code (implementations and tests) have **no warnings** from the compiler (warnings have a ⚠ symbol, as opposed to the ⊗ for errors) and no `@SuppressWarnings` annotations.

Run *all* the tests in the project by right-clicking on the `test` folder and selecting *Run As* → *JUnit Test*.



3.1.5 Problem 4: Poetic walks

3.1.5.1 Test `GraphPoet`

Devise, document, and implement tests for `GraphPoet` in **`GraphPoetTest.java`**

TEST1

```
final GraphPoet nimoy = new GraphPoet(new File("src/P1/poet/mugar-omni-theater.txt"));
final String input = "Test the system.";
assertEquals("Test of the system.",nimoy.poem(input));
```

TEST2

```
final GraphPoet nimoy = new GraphPoet(new File("src/P1/poet/mugar-omni-theater.txt"));
assertEquals("{\n"
        + "  This--1->is\n"
        + "  is--1->a\n"
        + "  a--2->test\n"
        + "  test--2->of\n"
        + "  of--2->the\n"
        + "  the--1->Mugar\n"
        + "  Mugar--1->Omni\n"
        + "  Omni--1->Theater\n"
        + "  Theater--1->sound\n"
        + "  sound--1->system.\n"
        + "  system.--1->is\n"
        + "  is--1->data\n"
        + "  data--1->a\n"
        + "}",nimoy.toString());
```

3.1.5.2 Implement GraphPoet

Implement `GraphPoet` in **`GraphPoet.java`**.

I use `Graph` in the rep of `GraphPoet`. My `GraphPoet` rely only on the specs of `Graph`, not on the details of a particular `Graph` implementation.

（1）writing AF、RI、SRE

```
// Abstraction function:
//    R = edges    A = String
//    AF(a--n->b) = "a b a b ..."(n times)
//
// Representation invariant:
//    no edge which weight negative.
//
// Safety from rep exposure:
//   all fields are private.
//   all returns are immutable.
```

（2）implement Spec

```
/**
 * Create a new poet with the graph from corpus (as described above).
 *
 * @param corpus text file from which to derive the poet's affinity graph
 * @throws IOException if the corpus file cannot be found or read
 */
```

```
public GraphPoet(File corpus) throws IOException
// checkRep
private boolean checkRep()

/**
 * Generate a poem.
 *
 * @param input string from which to create the poem
 * @return poem (as described above)
 */
public String poem(String input)
// toString()
/**
 * toString
 * @return toString
 */
@Override public String toString()
```

3.1.5.3 Graph poetry slam

update the main method in Main.java with a cool example: an interesting poem from minimal input, a surprising poem given the corpus and input, or something else cool, such as this sentence.

相关更改已经应用到 Main.java

3.1.6 使用 Eclemma 检查测试的代码覆盖度

HIT-Lab2-120L022115 (2022年5月11日 下午10:32:56)

Element	Covera...	Covered Inst...
HIT-Lab2-120L022115	94.8 %	2,820
src	92.8 %	1,695
P1.graph	91.5 %	942
ConcreteVerticesGraph.jav	86.7 %	484
ConcreteEdgesGraph.java	96.8 %	363
Graph.java	97.9 %	95
P1.poet	81.3 %	126
Main.java	0.0 %	0
GraphPoet.java	96.9 %	126
P2	97.8 %	627
FriendshipGraph.java	97.0 %	453
Person.java	100.0 %	174
test	97.9 %	1,125
P1.graph	96.9 %	667
ConcreteVerticesGraphTes	93.4 %	214
GraphInstanceTest.java	88.0 %	22
GraphStaticTest.java	98.7 %	231
ConcreteEdgesGraphTest.j	100.0 %	200
P1.poet	97.7 %	130
GraphPoetTest.java	97.7 %	130
P2	100.0 %	328
FriendshipGraphTest.java	100.0 %	328

3.1.7 Before you're done

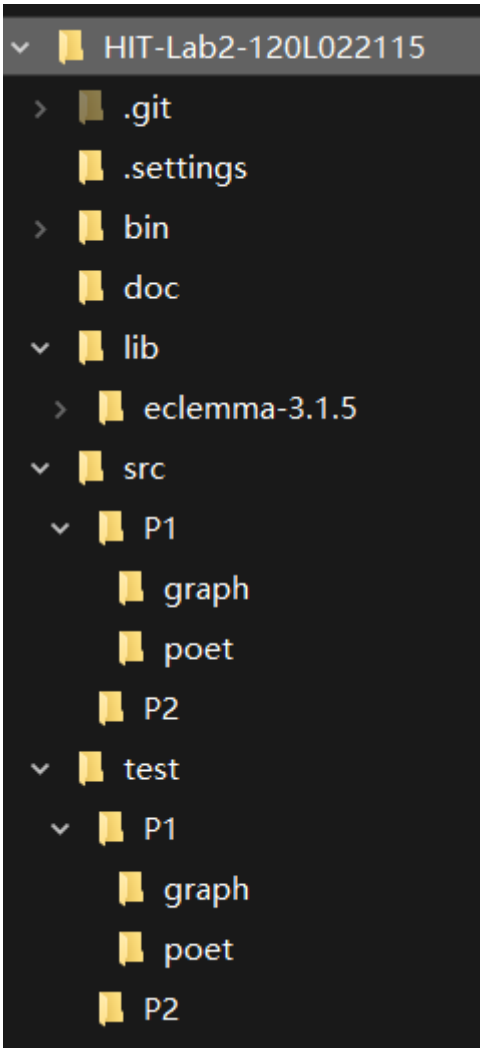
请按照 http://web.mit.edu/6.031/www/sp17/psets/ps2/#before_youre_done 的说明，检查你的程序。

如何通过 Git 提交当前版本到 GitHub 上你的 Lab2 仓库。

Git commit -a

Git push

在这里给出你的项目的目录结构树状示意图。



3.2 Re-implement the Social Network in Lab1

回顾 Lab1 实验手册中的 3.2 节 Social Network，针对所提供的客户端代码实现了 FriendshipGraph 类和 Person 类。在本次实验中，基于在 3.1 节 Poetic Walks 中定义的 Graph 及其两种 实现，重新实现 Lab1 中 3.3 节的 FriendshipGraph 类。

忽略在 Lab1 中实现的代码，无需其基础上实现本次作业；

在本节 FriendshipGraph 中，图中的节点仍需为 Person 类型。

3.2.1 `FriendshipGraph` 类

针 对 addVertex() 和 addEdge() ，尽可能复用 ConcreteEdgesGraph 中已经实现的 add() 和 set()方法。针对 getDistance()方法，基于所选定的 ConcreteEdgesGraph 的 rep 来实现，而不能修改其 rep。

仍然有三个方法，使用 Graph 代替原有 Set。

```
Graph<Person> set = Graph.empty();
```

并更新方法调用。

```
/**
 * 添加节点
 *
 * @param p 被添加的节点
 * @return 是否成功添加
 */
public boolean addVertex(Person p)

/**
 * 添加有向边
 *
 * @param s 开始节点
 * @param e 结束节点
 * @return 是否成功添加
 */
public boolean addEdge(Person s, Person e)

/**
 * 获取两个节点的距离
 *
 * @param s 起始节点
 * @param e 终止节点
 * @return 距离
 */
public int getDistance(Person s, Person e)
```

3.2.2 Person 类

仍按 LAB1 方式进行重构，并完善了 Spec & Safety。

```
/**
 * create a person
 * @param name of the person
 */
public Person(String name)
/**
 * get person name
 * @return name of the person
 */
public String getName()
/**
 * get friends which $this are not accepted.
 * @return set of persons
 */
```

```
public Set<Person> getFriendFather()
/**
 * get friends which only accepted by $this.
 * @return set of persons
 */
public Set<Person> getFriendChild()
/**
 * add friends which $this are not accepted.
 * @param p the added friend
 */
public void addFriendFather(Person p)
/**
 * add friends which only accepted by $this.
 * @param p the added friend
 */
public void addFriendChild(Person p)
/**
 * add friends by two people accepted.
 * @param p the added friend
 */
public void addFriend(Person p)
/**
 * get a string to describe.
 */
@Override
public String toString()
public String toStringAll()
```

3.2.3 客户端 main()

不变动 Lab1 的 3.3 节给出的客户端代码（例如 main()中的代码），即 同样的客户端代码仍可运行。

```
欢迎来到友情图！
使用以下命令来操作：
new name //创建一个新人物
add name //把一个人物加入图
dis name1 name2 //计算两个人物的距离
edg name1 name2 //添加name1到name2的友情线
info name1 //查看name1人的信息
exit //退出
>>new 1
1 已创建
>>new 1
已有名字为1的人，请更换名字后再试
>>new 2
2 已创建
>>add 1
1 已添加
>>add2
命令错误！
```

```
>>edg 1 2
1->2 已添加
>>edg 1 3
1->3 已添加
>>edg 3 2
3->2 已添加
>>dis 1 2
1->2的距离是：1
>>info 1
{
        name=1
        childs=3,2
        fathers=
}
>>dis 2 3
2->3没有路径！
```

```java
public static void main(String[] args) {
        // Auto-generated method stub
        HashMap<String, Person> map = new HashMap<String, Person>();
        FriendshipGraph graph = new FriendshipGraph();
        Scanner sn = new Scanner(System.in);
        System.out.println("欢迎来到友情图！");
        System.out.println("使用以下命令来操作：");
        System.out.println("new name //创建一个新人物");
        System.out.println("add name //把一个人物加入图");
        System.out.println("dis name1 name2 //计算两个人物的距离");
        System.out.println("edg name1 name2 //添加name1到name2的友情线");
        System.out.println("info name1 //查看name1人的信息");
        System.out.println("exit //退出");
        while (true) {
            System.out.print(">>");
            String input = sn.nextLine();
            if(input.equals("exit")) {
                sn.close();
                break;
            }
            String[] arr;
            try {
                arr = input.split(" ");
            } catch (PatternSyntaxException e) {
                System.out.println("命令错误！");
                continue;
            }
            //System.out.println(Arrays.toString(arr));
```

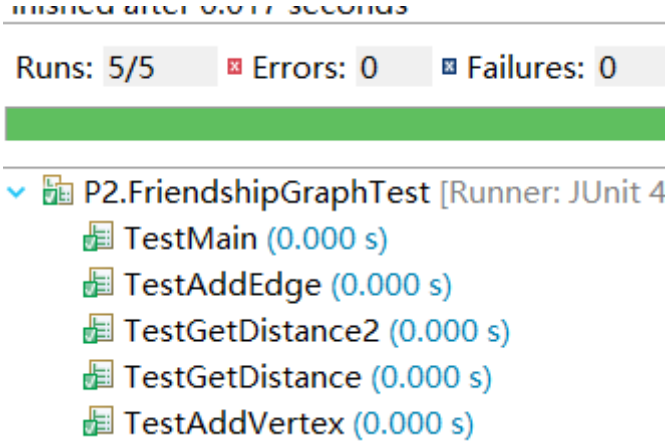
```
            if (arr.length == 0)
                continue;
            try {
                try {
                    if (arr[0].equals("new")) {
                        if (arr.length > 2) {
                            System.out.println("命令错误！ ");
                            continue;
                        }
                        map.put(arr[1], new Person(arr[1]));

                        System.out.println(arr[1]+" 已创建");
                    }else if(arr[0].equals("add")) {
                        if (arr.length > 2) {
                            System.out.println("命令错误！ ");
                            continue;
                        }
                        Person p = map.get(arr[1]);
                        if(p==null) {
                            System.out.println("没有这个人！请重新输入");
                            continue;
                        }
                        graph.addVertex(p);
                        System.out.println(p.getName()+" 已添加");

                    }else if(arr[0].equals("info")) {
                        if (arr.length > 2) {
                            System.out.println("命令错误！ ");
                            continue;
                        }
                        Person p = map.get(arr[1]);
                        if(p==null) {
                            System.out.println("没有这个人！请重新输入");
                            continue;
                        }
                        System.out.println(p.toStringAll());

                    }else if(arr[0].equals("edg")) {
                        if (arr.length > 3) {
                            System.out.println("命令错误！ ");
                            continue;
                        }
                        Person s = map.get(arr[1]), e = map.get(arr[2]);
                        if(s==null||e==null) {
                            System.out.println("没有这个人！请重新输入");
                            continue;
                        }
                        graph.addEdge(s, e);
                        System.out.println(s.getName()+"->"+e.getName()+" 已
添加");
                    }else if(arr[0].equals("dis")) {
                        if (arr.length > 3) {
                            System.out.println("命令错误！ ");
                            continue;
                        }
                        Person s = map.get(arr[1]), e = map.get(arr[2]);
                        if(s==null||e==null) {
```

```
                        System.out.println("没有这个人！请重新输入");
                        continue;
                    }
                    int i = graph.getDistance(s, e);
                    if(i>=0) {

System.out.println(s.getName()+"->"+e.getName()+"的距离是："+i);
                    }else {

System.out.println(s.getName()+"->"+e.getName()+"没有路径！");
                    }

                }else {
                    System.out.println("命令错误！");
                    continue;
                }
            } catch (IndexOutOfBoundsException e) {
                System.out.println("命令错误！");
                continue;
            }
        } catch (IllegalArgumentException e) {
                System.out.println(e.getMessage());
                continue;
        }

    }
    }
```

3.2.4 测试用例

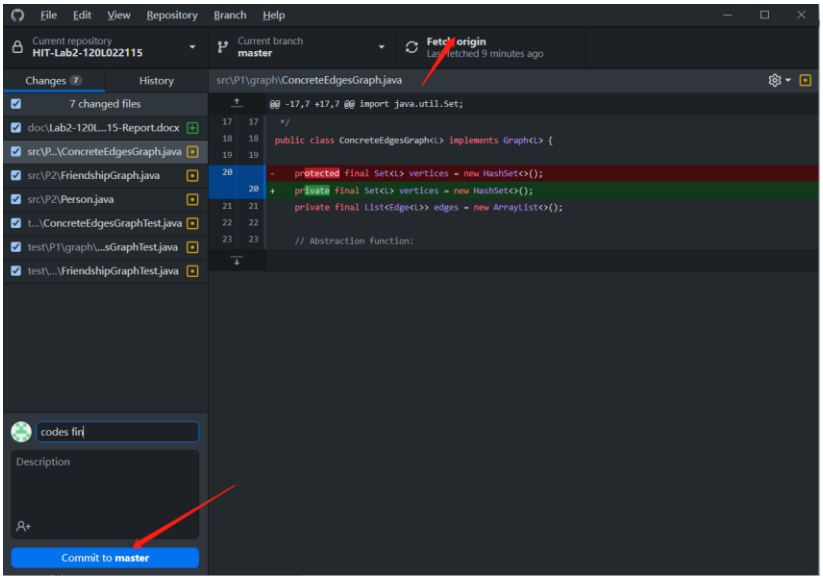
重新执行在 Lab1 里所写的 JUnit 测试用例，测试在本实验里新实现的 FriendshipGraph 类仍然表现正常。

重新执行了测试用例，表现依然正常。



3.2.5 提交至 Git 仓库

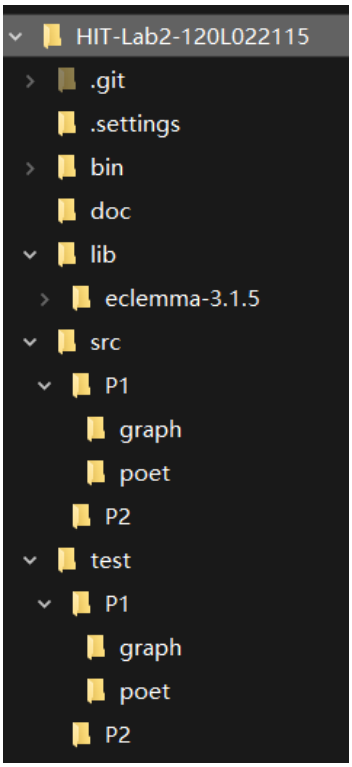
如何通过 Git 提交当前版本到 GitHub 上你的 Lab3 仓库。



1、commit

2、push

在这里给出你的项目的目录结构树状示意图。

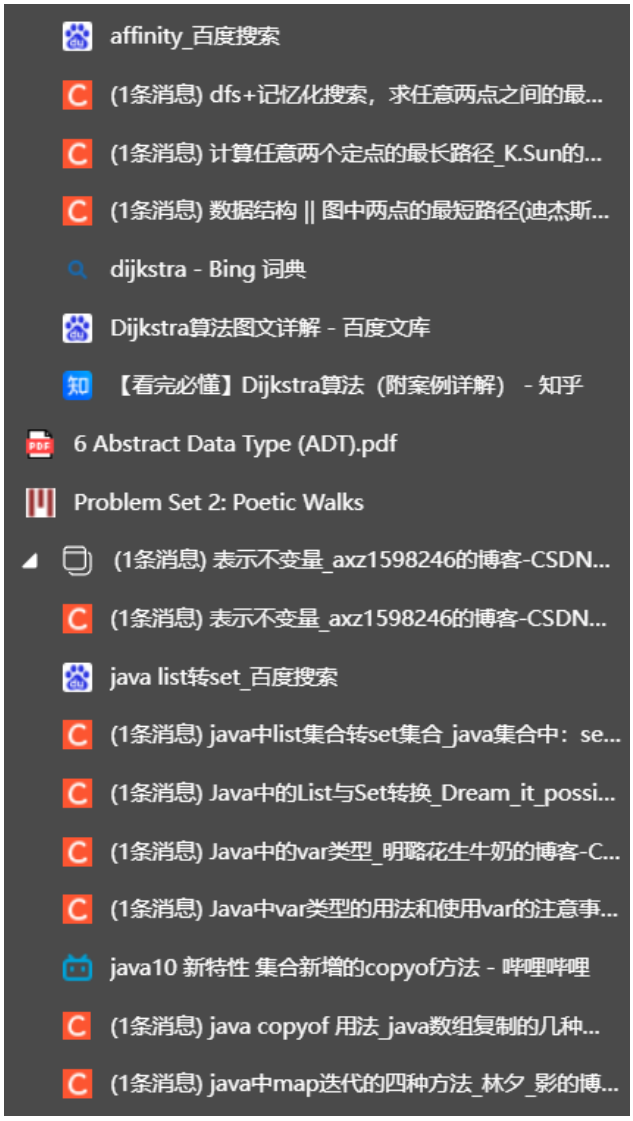


4 实验进度记录

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
2022-05-10	16:00-18:00, 18:30-23:30	完成 3.1	按计划完成
2022-05-11	12:20-13:00, 16:00-18:00 18:30-23:30	完成 3.2 与报告	按计划完成
2022-05-17	16:00-17:30	完成报告最终部分	按计划完成

5 实验过程中遇到的困难与解决途径

遇到的难点	解决途径
如图所示	

6 实验过程中收获的经验、教训、感想

6.1 实验过程中收获的经验和教训（必答）

学会了使用接口以及接口的实现方法。
学会了类的继承。
学会了泛型的使用。
复习了一种算法。
写了很多 spec 和 doc。

6.2 针对以下方面的感受（必答）

(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 更抽象，需要建立数学模型，设计接口。
而面向应用场景编程更加具体，更易上手。

(2) 使用泛型和不使用泛型的编程，对你来说有何差异？
差异很大，最好还是使用泛型，以免造成不必要的麻烦（改一堆代码）。

(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？
优势是能设计很多测试用例（黑盒测试），而非局限于自己所编写的代码实现。
能。

(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？
减少工作量，代码复用提高效率。

(5) 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？
为了以后/合作时不出错，因为这种一出错就是很难发现的错误。如果出错就会不知道在哪里出错，查错非常麻烦。
愿意。

(6) 关于本实验的工作量、难度、deadline。
工作量：合适。
难度：合适。
deadline：合适。

(7) 《软件构造》课程进展到目前，你对该课程有何收获和建议？
无建议。
收获是学会了版本管理工具 git 和一些 OOP、ADT 的常用方法，这有助于提升我们的编程技能和掌握编程规范。