



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

# 2019 年春季学期 计算机学院《软件构造》课程

## Lab 1 实验报告

姓名	
学号	
班号	
电子邮件	
手机号码	

## 目录

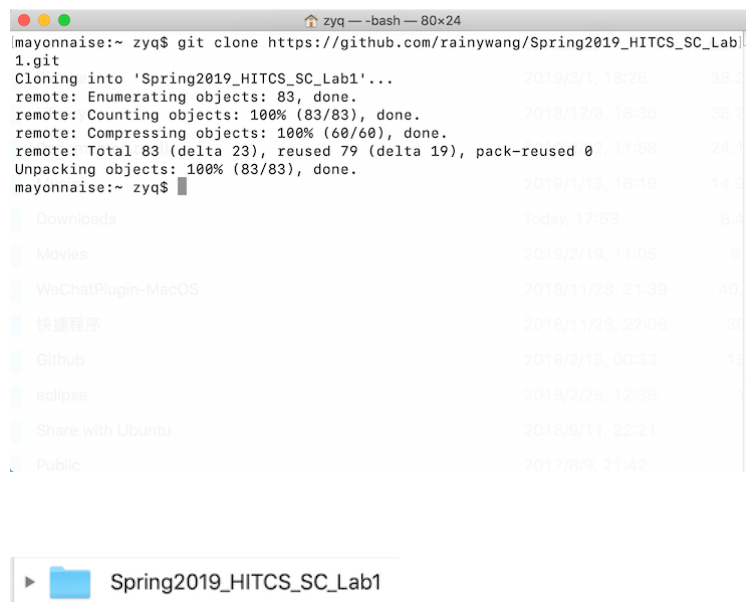
1 实验目标概述.....	1
2 实验环境配置.....	1
3 实验过程.....	2
3.1 Magic Squares .....	2
3.1.1 isLegalMagicSquare() .....	2
3.1.2 generateMagicSquare() .....	3
3.2 Turtle Graphics .....	3
3.2.1 Problem 1: Clone and import .....	3
3.2.2 Problem 3: Turtle graphics and drawSquare .....	3
3.2.3 Problem 5: Drawing polygons .....	3
3.2.4 Problem 6: Calculating Bearings.....	3
3.2.5 Problem 7: Convex Hulls.....	4
3.2.6 Problem 8: Personal art .....	4
3.2.7 Submitting .....	4
3.3 Social Network.....	4
3.3.1 设计/实现 FriendshipGraph 类 .....	4
3.3.2 设计/实现 Person 类.....	5
3.3.3 设计/实现客户端代码 main().....	5
3.3.4 设计/实现测试用例 .....	5
3.4 Tweet Tweet .....	7
3.4.1 Problem 1: Extracting data from tweets .....	7
3.4.2 Problem 2: Filtering lists of tweets .....	8
3.4.3 Problem 3: Inferring a social network.....	8
3.4.4 Problem 4: Get smarter.....	9
4 实验进度记录.....	10
5 实验过程中遇到的困难与解决途径.....	11
6 实验过程中收获的经验、教训、感想 .....	11
6.1 实验过程中收获的经验教训.....	11
6.2 针对以下方面的感受 .....	11

## 1 实验目标概述

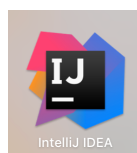
1. 训练基本 Java 编程技能, 能够利用 Java OO 开发基本的功能模块, 能够阅读理解已有代码框架并根据功能需求补全代码, 能够为所开发的代码编写基本的测试程序并完成测试, 初步保证所开发代码的正确性。 另一方面利用 Git 作为代码配置管理的工具, 学会 Git 的基本使用方法。
2. 学习使用基于 JUnit 的测试。
3. 熟练掌握使用 IDE 编程的技能。

## 2 实验环境配置

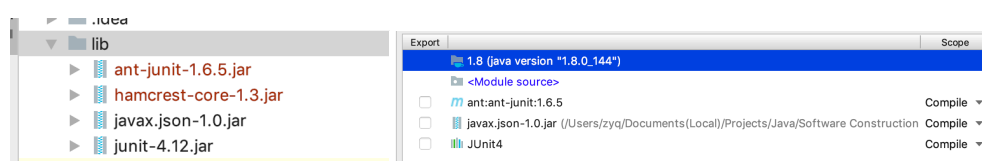
### 1. 从 Git 上获取实验资源



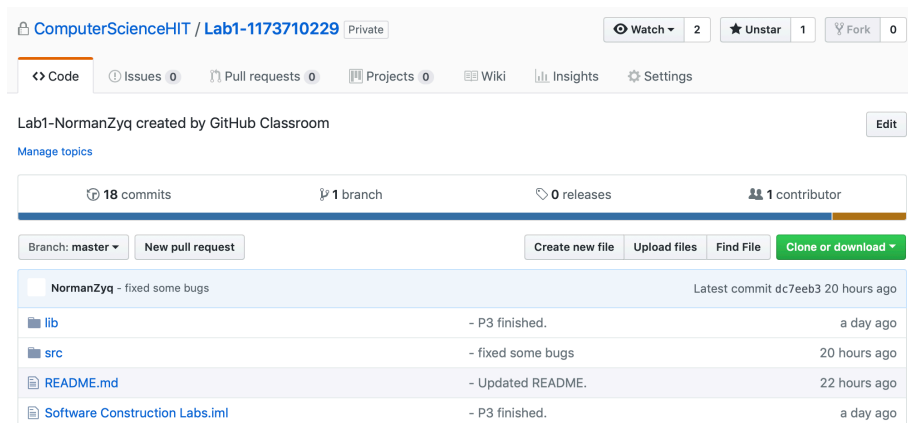
### 2. IDE 使用 IntelliJ IDEA



3. 创建项目, 将实验资源导入项目
4. 安装配置 JUnit 4.12



## 5. git remote, 与远程仓库建立连接



6. 仓库链接: <https://github.com/ComputerScienceHIT/Lab1-1173710229.git>
7. 遇到的困难: 在配置 JUnit 时最初直接选了最新版, 然后没法测试, 后来通过 IDEA 自动修复的建议改成了 4.x 版, 后查到资料说也可以通过加入 hamcrest-core 包来解决。

## 3 实验过程

请仔细对照实验手册, 针对四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 (但无需把你的源代码全部粘贴过来! )。

为了条理清晰, 可根据需要在各节增加三级标题。

### 3.1 Magic Squares

该任务考验基础的 Java 编程能力和简单的 Java 类库, 例如 IO、包装类等等。

#### 3.1.1 isLegalMagicSquare()

首先是读文件, 我使用 Scanner 对象配合 File 读文件。

判断幻方的思路是先读取第一行并且截取出数字, 然后判断这一行有多少个数, 然后求和, 这两个结果作为基准用来判断剩下的行列和对角线。

然后计算剩下的每一行、每一列和两个对角线的数之和，如果中途出现缺数则说明这不是个方阵；如果出现类型不匹配则说明不是使用\t 分隔或存在小数；如果出现任何一个结果不同则说明不是幻方。

### 3.1.2 generateMagicSquare()

这个方法根据罗伯法，总体思路是从第 0 行的中间开始放置数，然后依次向其右上角放大 1 的数，如果已经是最后一行，则挪动到最后一行，如果已经是最后一列，则挪动到第 0 列，这样放置完成后便是幻方。

但是这个方法只适用于生成奇数阶的幻方，如果是偶数阶，则会因为某一次的冲突而需要向下挪动一行放置时导致数组越界。

根据实验手册的要求，将该方法最后增加了利用 `PrintWriter` 写文件的操作。

## 3.2 Turtle Graphics

Turtle Graphics 是 MIT 编写的绘图程序，通过设定 turtle 的移动能画出不同的图像，我们在此实验中要做的是实现部分方法使 turtle 完成要求的操作。此部分更偏重对数学和逻辑的考察。

### 3.2.1 Problem 1: Clone and import

使用 `git clone` [https://github.com/rainywang/Spring2019\\_HITCS\\_SC\\_Lab1.git](https://github.com/rainywang/Spring2019_HITCS_SC_Lab1.git) 命令获取实验资源，放到项目文件夹。

### 3.2.2 Problem 3: Turtle graphics and drawSquare

绘制正方形较为简单，只需要画四条边，每次旋转 90 度。

### 3.2.3 Problem 5: Drawing polygons

首先实现计算多边形内角度数的方法 `calculateRegularPolygonAngle()`，根据公式  $180 \cdot (n-2) / n$  即得内角度数，然后只需循环  $n$  边，每次旋转 180-内角度数即可。

### 3.2.4 Problem 6: Calculating Bearings

先完成 `calculateBearingToPoint()` 方法，思路是先计算从 0 度开始需要旋转的角度，然后再和当前的角度作差即可。

有了 `calculateBearingToPoint()` 方法之后，只需要在 `calculateBearings()` 方法中遍历  $x$  和  $y$  坐标的列表，取出  $x$  和  $y$ ，然后调用 `calculateBearingToPoint()` 并储存结果。

### 3.2.5 Problem 7: Convex Hulls

首先找到点集中最左下角的点，然后利用 `calculateBearingToPoint()` 方法遍历点集中除了当前点之外的所有点，计算从 0 度和 180 度开始到目标点需要旋转的角度，挑取旋转角度最大的点作为下一个在凸包中的点，其中，如果存在目标点在当前点的右边，则忽略所有在左边的点。如果连续找到三个点在同一条直线上，则从凸包点集中移除中间的点，只保留前后的点以使凸包最小化。

### 3.2.6 Problem 8: Personal art

本任务可以随意作画。

### 3.2.7 Submitting

有两种方式：

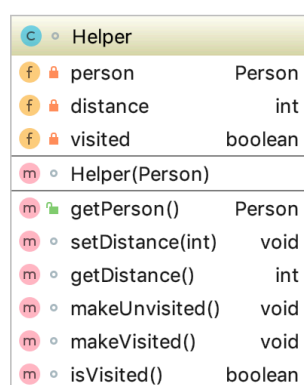
1. 利用 IDEA 中集成的 Git，VCS-Git-Commit/Push 可以十分便利地将本地仓库 push 到 GitHub 中的远程仓库；
2. 利用命令行：`git commit` 和 `git push --all` 将作出的更改 push 到远程仓库

## 3.3 Social Network

该任务模拟朋友之间的交际圈，可为无向图也可以是有向图

### 3.3.1 Helper 类

为了能够未来引入更多内容并且计算距离，引入了 Helper 类，类的设计如图：



Helper	
person	Person
distance	int
visited	boolean
Helper(Person)	
getPerson()	Person
setDistance(int)	void
getDistance()	int
makeUnvisited()	void
makeVisited()	void
isVisited()	boolean

其中 Person 对象即人，距离是从起始点开始的距离，visited 是访问标记。

### 3.3.2 设计/实现 FriendshipGraph 类

类的设计如图：

FriendshipGraph		
f	vertexCount	int
f	friendship	HashMap<Integer, List<Helper>>
f	person2Helper	HashMap<Person, Helper>
f	person2Index	HashMap<Person, Integer>
m	FriendshipGraph()	
m	addVertex(Person)	boolean
m	resetDistanceVisited()	void
m	addEdge(Person, Person)	boolean
m	getDistance(Person, Person)	int
m	getVertexCount()	int
m	getFriendship()	HashMap<Integer, List<Helper>>
m	getPerson2Helper()	HashMap<Person, Helper>
m	getPerson2Index()	HashMap<Person, Integer>
m	printFriendship()	void

vertexCount 是当前图的顶点数目, friendship 是关系表, key 是整型数, value 是 Helper 的列表; person2Helper 和 person2Index 分别都是将 person 作为 key, 前者的 value 是 helper, 后者的 value 是整数, 用于实现 person 和 helper 和在 map 中的索引的映射关系。这三个 map 中的 helper 对象都是共享的。

### 3.3.3 设计/实现 Person 类

Person		
f	allPeopleNames	Map<String, Boolean>
f	name	String
m	Person(String)	
m	getName()	String
m	toString()	String

name 是姓名; ALL\_PEOPLE\_NAMES 是静态常量, 记录所有已实例化的用户, 用于判断是否重名, 每次在实例化 FriendshipGraph 时会清空这个 map。

### 3.3.4 设计/实现客户端代码 main()

调用 Test 类的测试方法。

### 3.3.5 设计/实现测试用例

FriendshipGraphTest		
m	addVertexTest()	void
m	addEdgeTest()	void
m	friendshipTest1()	void
m	friendshipGraphTest2()	void
m	friendshipGraphTest3()	void

## 1. addVertexTest()

测试过程:

- i. 创建两个 person 对象, name 分别是 rachel 和 ross, 执行 addVertex 方法两次, 将两个 person 都添加进去, 然后再次添加 rachel;
- ii. 使用 assert 判断结果的正确性。

测试结果: 重复添加相同的人时会输出一段提示, 例如 “Person name: rachel, hash code: 425918570 的顶点已存在”, 不会重复添加; 添加顶点后和手动创建的正确答案相比没有任何差别, 测试通过。

## 2. addEdgeTest()

测试过程: 创建三个 person 对象, 姓名分别是 rachel, ross, ben。添加这三个人到顶点, 调用 addEdge() 方法添加两条边: rachel 到 ross, 和 ross 到 ben, 使用 assert 语句断言 rachel 可达 ross, ben 不可达 ross。

测试结果: 测试通过。

## 3. friendshipTest1() 与实验手册要求的测试内容相同。

## 4. friendshipTest2()

本方法仅测试无向图的情况。

测试过程: 添加较为复杂的图, 邻接表如图所示:

```
Rachel---->Ross---->a1
Ross---->Rachel---->Ben---->c1
Ben---->Ross
Kramer---->c1
a1---->Rachel---->b1
b1---->a1
c1---->Ross---->Kramer
alone
```

(本图中, 任意两个人若有联系, 则都是双向的)

经过这些断言:

```
assertEquals( expected: 1, graph.getDistance(rachel, ross));
assertEquals( expected: 2, graph.getDistance(rachel, ben));
assertEquals( expected: 0, graph.getDistance(rachel, rachel));
assertEquals( expected: 3, graph.getDistance(rachel, kramer));
assertEquals( expected: 2, graph.getDistance(ben, rachel));
assertEquals( expected: 3, graph.getDistance(kramer, rachel));
assertEquals( expected: 0, graph.getDistance(kramer, kramer));
assertEquals( expected: -1, graph.getDistance(alone, rachel));
```

测试结果: 全部通过。

## 5. friendshipTest3()

本方法用于测试有向图的情况。

测试过程: 添加较为复杂的有向图, 邻接表如图所示:

```
Rachel---->Ross
Ross---->Rachel---->c1
Ben---->Ross
Kramer
a1---->Rachel
b1---->a1
c1---->Ross---->Kramer
alone
```



(本图中, 任意两个人之间并非都是双向的)

经过这些断言:

```
assertEquals( expected: 1, graph.getDistance(rachel, ross));
assertEquals( expected: 1, graph.getDistance(ross, c1));

// directed graph, so ben can access to ross but ross cannot access to ben
assertEquals( expected: 1, graph.getDistance(ben, ross));
assertEquals( expected: -1, graph.getDistance(ross, ben));
```

测试结果: 全部通过。

## 3.4 Tweet Tweet

### 3.4.1 Problem 1: Extracting data from tweets

#### 1. getTimespan()

**实现:** 计算能够包含传入参数中的 list 全部 tweet 的最小时间范围, 只需便利一遍全部推文, 记录最早和最晚的推文发送时间, 然后计算差值并返回。

**测试:** 创建四个不同的 Instant 对象和四条 tweet:

```
private static final Instant d1 = Instant.parse("2016-02-17T10:00:00Z");
private static final Instant d2 = Instant.parse("2016-02-17T11:00:00Z");
private static final Instant d3 = Instant.parse("2016-02-18T11:00:00Z");
private static final Instant d4 = Instant.parse("2016-02-15T11:00:00Z");

private static final Tweet tweet1 = new Tweet( id: 1, author: "alyssa", text: "is it reasonable to talk about rivest so much?", d1);
private static final Tweet tweet2 = new Tweet( id: 2, author: "bbitdiddle", text: "rivest talk in 30 minutes #hype", d2);
private static final Tweet tweet3 = new Tweet( id: 3, author: "zz", text: "I found your pen in my bag. @iceiwant", d3);
private static final Tweet tweet4 = new Tweet( id: 4, author: "iceiwant", text: "Thank you. @zz. Did you find my CSAPP book? @yyy", d4);
```

加入列表, 调用方法, 正确的返回结果应该满足: start 是 d4, end 是 d3。

**测试结果:** 满足正确结果。

#### 2. getMentionedUsers()

**实现:** 从参数的 tweet 列表中获得所有被@到了的用户名。首先遍历整个推文列表, 用正则表达式按照实验要求匹配@符号和后面的合法用户名, 将提取到的用户名加入 set, 且不要重复添加相同的用户名。

**测试:** 使用这些 tweet 测试

```
private static final Tweet tweet3 = new Tweet( id: 3, author: "zz", text: "I found your pen in my bag. @iceiwant", d3);
private static final Tweet tweet4 = new Tweet( id: 4, author: "iceiwant", text: "Thank you. @zz. Did you find my CSAPP book? @yyy", d4);
private static final Tweet tweet5 = new Tweet( id: 5, author: "wang", text: "my email is nhjskd@hit.edu.cn", d2);
private static final Tweet tweet6 = new Tweet( id: 6, author: "wang", text: "@hitt.edu.cn", d2);
private static final Tweet tweet7 = new Tweet( id: 7, author: "wang", text: "@iceiwantq@jisdf @ra @ra @rb @rc- mit@hit.edu.cn @haveme/ @nome//", d2);
```

正确返回应该是:

```
answer.add("yyy");
answer.add("iceiwant");
answer.add("zz");
answer.add("ra");
answer.add("rb");
answer.add("rc-");
answer.add("iceiwantq");
answer.add("hitt");
answer.add("haveme");
answer.add("nome");
```




**测试结果:** 返回结果正确。

### 3.4.2 Problem 2: Filtering lists of tweets

#### 1. `writtenBy(List<Tweet> tweets, String username)`

**实现:** 从 tweets 中提取出作者是 username 的 tweet 后添加到列表作为返回值。遍历 tweets, 取出作者, 与 username 对比, 若相同则包含此 tweet。

**测试:** 测试分为三组, 分别测试结果为空, 单个结果和多个结果。

 <code>testWrittenByEmpty()</code>	<code>void</code>
 <code>testWrittenByMultipleTweetsSingleResult()</code>	<code>void</code>
 <code>testWrittenByMultipleTweetsMultipleResult()</code>	<code>void</code>

**测试结果:** 全部正确。

#### 2. `inTimespan(List<Tweet> tweets, Timespan timespan)`

**实现:** 根据 timespan 从 tweets 中筛选出在此范围内的 tweet




**测试:** 创建一个时间范围和三条 tweet, 其中只有两条在此范围内, 调用该方法, 返回结果应该只包含这两条 tweet。

**测试结果:** 全部正确。

#### 3. `containing(List<Tweet> tweets, List<String> words)`

**实现:** 根据 words 中的每个条目从 tweets 中筛选出包含 words 中任意一个条目的内容。遍历每个 tweet, 在内部便利 words, 若有匹配则将这条 tweet 加入返回列表中, 接触对当前 tweet 的遍历, 因已经被包含了, 结束这个双重循环时便筛选完毕。

**测试:** 分为三个部分, 分别是测试空结果、测试简单筛选、测试复杂单词筛选。其中, 测试复杂筛选包括测试大小写忽略、测试特殊符号是否会干扰筛选等

 <code>testContainingEmpty()</code>	<code>void</code>
 <code>testContaining()</code>	<code>void</code>
 <code>testContainingStrict()</code>	<code>void</code>

**测试结果:** 全部正确

### 3.4.3 Problem 3: Inferring a social network

#### 1. `guessFollowsGraph(List<Tweet> tweets)`

**实现:** 采用@的方式进行推测, 例如: a 在一条推文中@b, 则 a 有可能关注了 b

**测试:** 分为两部分, 一组测试的返回结果长度应为 0, 另一组不为 0。测试 tweet 为以下内容:

```
public static final Tweet tweet1 = new Tweet(1, author: "stuA", text: "I love you @stuB", d1);
public static final Tweet tweet2 = new Tweet(2, author: "stuB", text: "Thank you @stuA", d2);
public static final Tweet tweet3 = new Tweet(3, author: "zz", text: "@yyy hi @stuB @stuA", d2);
public static final Tweet tweet4 = new Tweet(4, author: "yyy", text: "ok @wang @stuB", d2);
public static final Tweet tweet5 = new Tweet(5, author: "wang", text: "come to my office tomorrow @stuA @stuB", d2);
public static final Tweet tweet6 = new Tweet(6, author: "wang", text: "@hahaha @yyy", d2);
```

正确结果应该是: stuA 关注了 stuB; stuB 关注了 stuA; zz 关注了 yyy、stuB、stuA; yyy 关注了 wang、stuB; wang 关注了 stuA、stuB、hahaha、

yyy。

**测试结果：**两组都与预期相同

## 2. smarterGuessFollowsGraph(List<Tweet> tweets)

**实现：**对原有的交际圈进行一次传递性的猜测，例如 a 关注了 b，b 关注了 c，则 a 可能也关注了 c。以如何为 a 进行拓展为例：遍历 a 的关注列表，取出每个 a 关注的人，再去遍历他们的关注列表，将这些列表中的除了 a 自己的所有人添加到 a 的关注列表中。

**测试：**测试用的 tweet 与 1 中的相同，正确结果应为：stuA 关注 stuB；stuB 关注 stuA；zz 关注 yyy、stuB、stuA、wang；yyy 关注 wang、stuA、stuB、hahaha；wang 关注 stuA、stuB、hahaha、yyy

**测试结果：**与预期相同

## 3. influencers(Map<String, Set<String>> followsGraph)

**实现：**引入内部类 User 用来记录用户的粉丝数。

```
// 遍历所有key和value
for (Map.Entry<String, Set<String>> entry : followsGraph.entrySet()) {
    String currentUsername = entry.getKey(); // 获得作为索引的用户名
    User currentUser = new User(currentUsername); // 创建用户的对象，粉丝数初始化为0
    users.add(currentUser); // 添加到user列表

    for (Set<String> usernameSets : followsGraph.values()) {
        for (String username : usernameSets) { // entry.getValue()返回的是Set
            // 每次getValue()都是一个新的Set，遍历这个Set的所有内容
            if (currentUsername.equals(username)) { // 如果作为索引的用户名出现在了Set中，则说明多了一个粉丝
                currentUser.appendFollower(); // 粉丝数++
            }
        }
    }
}
```

利用 user 列表快速排序后按顺序取出用户名作为返回值。

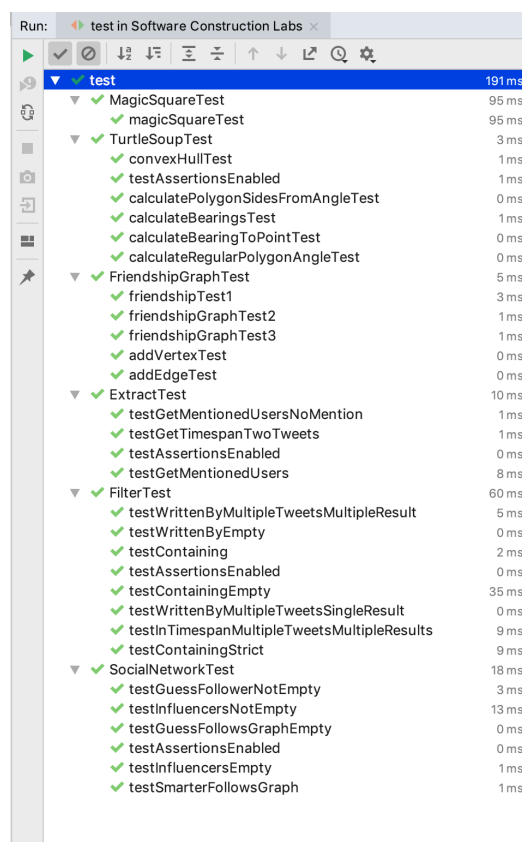
**测试：**测试用例与 1 相同，正确的按影响力排序结果应为：stuB, stuA, yyy, wang, zz

**测试结果：**与预期相同

### 3.4.4 Problem 4: Get smarter

采用对已有的交际圈进行一次传递性拓展来扩大和完善交际圈推测，具体内容见 [3.4.3-2](#)

### 3.5 测试运行结果



## 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。

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	任务	实际完成情况
2019-02-25	14:00-15:30	写了一部分的 P1	按计划完成
2019-03-02	15:00-15:15	P1 搞定了	按计划完成
2019-03-03	15:30-16:00	P2 一部分	遇到困难，未完成
2019-03-04	13:45-15:30	P2 一部分，开始设计 P3 数据结构	遇到困难
2019-03-04	19:00-22:00	P2 中除了最后一个测试，其他已完成； P3 开始尝试	超时完成
2019-03-05	18:30-21:00	P3 数据结构整改，完成大部分	按计划完成
2019-03-06	18:30-22:30	P3 P4 完成，修复了之前的许多问题	按计划完成
2019-03-08	20:00-22:00	P4 的 get smarter 和实验报告	按计划完成
2019-03-09	16:00-16:40	修复 P2 凸包一个已知的 bug	提前完成

2019-03-09	18:30-22:00	写实验报告	
2019-03-09	19:42-20:35	完成实验报告的大部分内容	

## 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
计算凸包时，最开始没有考虑三个点共线的问题。	重新理清思路，举例有哪些情况才应该归类到这个三点共线情况，重新设计算法的结构然后修复问题。
FriendshipGraph 的数据结构设计问题。	考虑到这个人脉圈（即“谁是谁的朋友”这一个关系）应该只存在于 FriendshipGraph 的对象中，并且每个不同的 graph 对象都应该具备“实现全新的人脉圈”的能力，所以我没有简单的把“谁是谁的朋友”这一关系涉及到 Person 类中，同时为了实现部分 Person 不能拥有的功能（如计算距离），于是引入了新的 Helper 类，Person 类与其成为组合关系，在每一个人脉圈中，都是 Helper 对象与它产生直接的关系，Person 对象与 graph 的关系成为了间接。

## 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教训

1. 打代码时需要保持冷静，安静的环境或者听听轻音乐有利于理清思绪
2. 实现每一个部分前应该先设计好，不然可能会变成无头苍蝇，一顿乱撞，修复一个错误回头发现带来了新的错误

### 6.2 针对以下方面的感受

Java 很有意思，很强大，我现在都只触碰到了 Java 的一面皮毛，它实在是太高深了，值得我们深入学习，深入理解。它不仅作为一门编程语言存在，还是一个帮助大家接触面向对象、面向接口等编程思想，抑或各种设计模式的工具，还是陪伴我们入门到入土的好伴侣。Eclipse IDE 实在是强大，但 UI 在 macOS 上不那么友好，不像 Windows 那般惹人喜爱，所以我选择了 IntelliJ IDEA 这款 IDE，它集成的功能更强大，抛开代码补全等普通的智能功能，它还能帮助我们优化代码、转化代码，提示性能低等等的问题，例如它会提醒将某些性能过低的索引 for 循环建议修改为 foreach 或者迭代器；它会建议将用来排序而实例化的比较器的匿名内部类改成 lambda 表达式，或者使用 Comparator 的静态方法……但这些都

只占了功能的冰山一角，开发这些工具的开发团队值得我们尊敬和学习！

这次试验不是我初次接触 Git，但是一直以来我都比较依赖于 IDE 集成的 Git 管理，几乎不接触命令行，这次我试着用部分命令取代以前的可视化操作，体验一下 IDE 在我们每次点击 `commit`、`push` 等操作的时候都执行了什么。

从上个学期的计算机系统到现在的软件构造，我们也经历了不少国外的实验作业了，不得不说，他们作业真是有创意，难度中等或偏上。软件构造这门课有了实验才有了灵魂，因为能真正让我们体会一下“软件构造”。可是，如果一下子把他们几种作业或实验揉到一起，虽然没有内容上的重复，但总有些奇怪的感觉，也许是量有点多的感觉，但是结合 `ddl` 来看，可能又是一个量适中的实验了。总的来说，这次试验挺有意思的。