

**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 | 李东炫 |
| 学号 | L173710101 |
| 班号 | 1737101 |
| 电子邮件 | 3034125066@qq.com |
| 手机号码 | 15561860820 |

**目录**

[1 实验目标概述 1](#_Toc1392798)

[2 实验环境配置 1](#_Toc1392799)

[3 实验过程 1](#_Toc1392800)

[3.1 Magic Squares 1](#_Toc1392801)

[3.1.1 isLegalMagicSquare() 1](#_Toc1392802)

[3.1.2 generateMagicSquare() 1](#_Toc1392803)

[3.2 Turtle Graphics 1](#_Toc1392804)

[3.2.1 Problem 1: Clone and import 2](#_Toc1392805)

[3.2.2 Problem 3: Turtle graphics and drawSquare 2](#_Toc1392806)

[3.2.3 Problem 5: Drawing polygons 2](#_Toc1392807)

[3.2.4 Problem 6: Calculating Bearings 2](#_Toc1392808)

[3.2.5 Problem 7: Convex Hulls 2](#_Toc1392809)

[3.2.6 Problem 8: Personal art 2](#_Toc1392810)

[3.2.7 Submitting 2](#_Toc1392811)

[3.3 Social Network 2](#_Toc1392812)

[3.3.1 设计/实现FriendshipGraph类 2](#_Toc1392813)

[3.3.2 设计/实现Person类 2](#_Toc1392814)

[3.3.3 设计/实现客户端代码main() 2](#_Toc1392815)

[3.3.4 设计/实现测试用例 3](#_Toc1392816)

[3.4 Tweet Tweet 3](#_Toc1392817)

[3.4.1 Problem 1: Extracting data from tweets 3](#_Toc1392818)

[3.4.2 Problem 2: Filtering lists of tweets 3](#_Toc1392819)

[3.4.3 Problem 3: Inferring a social network 3](#_Toc1392820)

[3.4.4 Problem 4: Get smarter 3](#_Toc1392821)

[4 实验进度记录 3](#_Toc1392822)

[5 实验过程中遇到的困难与解决途径 3](#_Toc1392823)

[6 实验过程中收获的经验、教训、感想 4](#_Toc1392824)

[6.1 实验过程中收获的经验和教训 4](#_Toc1392825)

[6.2 针对以下方面的感受 4](#_Toc1392826)

# 实验目标概述

学习和了解：

* 基本的 Java OO 编程
* 基于 Eclipse IDE 进行 Java 编程
* 基于 JUnit 的测试
* 基于 Git 的代码配置管理

# 实验环境配置

1. Eclipse Java

先在官网下载java开发工具包JDK8 for windows，然后在官网下载Eclipse 都是一步步安装提示安装。其中安装Eclipse的时候出现提示后来发现是不能放在中文目录下面，然后又提示安装时间超出正常安装失败，可能是因为选择的那个镜像本来就很慢，就反复多试几次就安装成功了。

1. Git

根据GitHub的网站提示一步步安装就好。

GitHub Lab1仓库的URL地址：

https://github.com/ComputerScienceHIT/Lab1-L173710101

# 实验过程

## Magic Squares

实验任务是关于幻方的，细化主要任务就是有读写txt文件，按格式存入数组，然后检验是否是幻方（每行每列对角线之和为同一个数字），还有生成幻方，还有包括对异常（输入）的处理，增加程序健壮性。

### isLegalMagicSquare()

这个方法的功能是判断文件的数据是否为幻方。是否是幻方又包括：是否是矩阵、是否由\t分割、是否全为正整数、然后才是是否行列对角线全为一个相同的数。

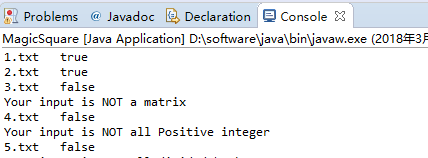
伪代码：

1. 定义一个200\*200的数组；
2. 打开文件讲数据读入数组（如果文件不存在，报错）
3. **while**(文件没有到末尾)
4. {
5. 读取一行；
6. 根据\t分割
7. 如果包含"，"、空格，说明至少不全是由\t分割的，返回**false**并报错
8. 将这一行的每一个字符串转到整形存到数组的对应行中
9. 如果不能出现不能转整形的异常，或者转换出来的整数小于等于0，说明数组文件并不全是正整数，返回**false**并报错
10. }
11. 记录行数和列数，如果读入总数不等于行数\*列数，则说明不是矩阵，返回**false**并报错
12. 求每一行每一列每对角线的和，再比较是否全部都相等
13. 如果相等 返回ture
14. 否则返回**false**

其中读入文件，判断是否分割正确、为矩阵的部分代码：

1. \\打开文件
2. **try** {
3. in = **new** BufferedReader(**new** FileReader(myFile));
4. String str;
5. **while** ((str = in.readLine()) != null) {
6. **int** rownum = 0;
7. line = str.split("\t");
8. **for** (String s : line) {
9. **try** {
10. **if** (s.contains(" ") || s.contains("\\,")) {
11. errkind = 2;  //  \\不全由”\t”分割
12. **return** **false**;
13. }
14. **if**(Integer.parseInt(s)<=0)
15. {
16. errkind = 3;   \\不是正整数
17. **return** **false**;
18. }
19. rownum++;
20. num++;
21. array[linenum][rownum] = Integer.parseInt(s);
22. } **catch** (Exception e) {
23. errkind = 3;  \\转换为整数出现异常，即输入不是整数
24. **return** **false**;
25. }
26. }

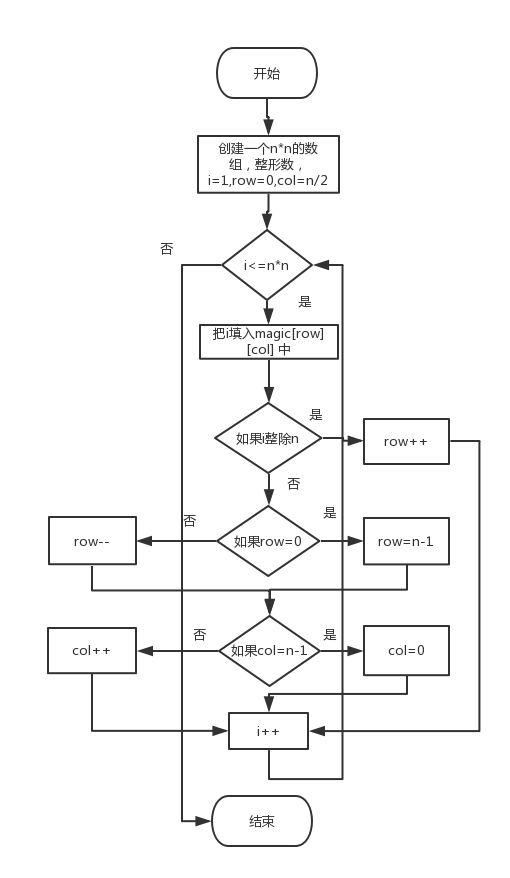
  输出结果：



### generateMagicSquare()

这个方法的功能是生成一个n阶幻方，n为正奇数，由用户输入，输入不正确会报错。由于生成幻方的主要部分题目已经给出，所有我们的重点是判断输入合法性和将生成的幻方写入一个新文件：6.txt。主要是学习了读写文件的基本操作。

流程图：



具体操作：

把1(或最小的数)放在第一行正中； 按以下规律排列剩下的n\*n-1个数：

　　(1)、每一个数放在前一个数的右上一格；

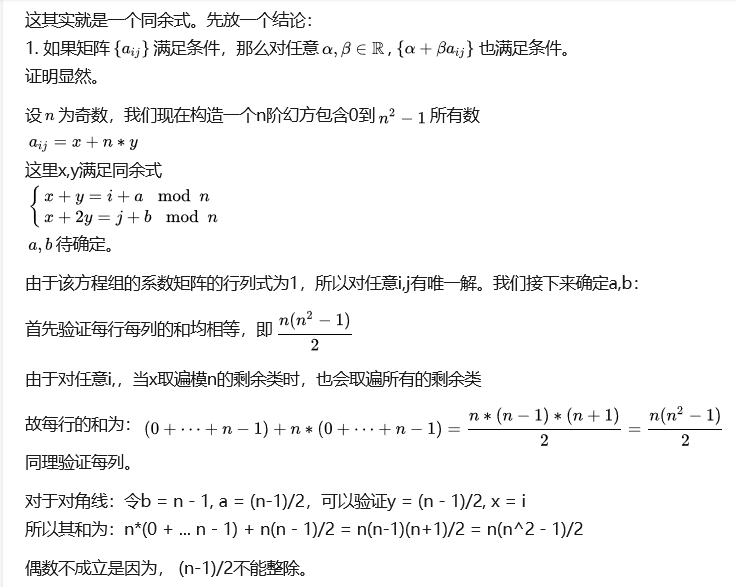
　　(2)、如果这个数所要放的格已经超出了顶行那么就把它放在底行，仍然要放在右一列；

　　(3)、如果这个数所要放的格已经超出了最右列那么就把它放在最左列，仍然要放在上一行；

　　(4)、如果这个数所要放的格已经超出了顶行且超出了最右列，那么就把它放在前一个数的下一行同一列的格内；

　　(5)、如果这个数所要放的格已经有数填入，处理方法同(4)。

原理：



出处：<https://www.zhihu.com/question/30498489/answer/161657996>

<https://www.zhihu.com/question/30498489/answer/48393702>

报错原因：

**偶数：java.lang.ArrayIndexOutOfBoundsException：**

使用不合法的索引访问数组时会报数组越界，可能的原因是使用的角标大于等于数组的长度或为负数

**负数：java.lang.NegativeArraySizeException**

如果应用程序试图创建大小为负的数组，则抛出该异常。

测试结果：

C:\Users\杨依婕\AppData\Roaming\Tencent\Users\854823622\TIM\WinTemp\RichOle\AWBX7~0{I5AM]VHGEQC@N3I.png

## Turtle Graphics

任务就像是模拟python里面的turtle画图库的最最基本的功能，而且其中主要的功能函数和GUI界面都已经写好了，我们只需要完成TurtleSoup.java的编写：根据所给的函数绘制一些图案，按照要求一步步实现几个方法，然后重要的是学习junit测试的基础知识。

### Problem 1: Clone and import

Get clone 老师项目的网址 获取到代码，

然后复制到本地的lab1仓库，在Eclipse下完成编写任务后，命令行git here,（因为老师的项目里面只是代码而不是完整的项目，所有不能直接像MIT任务一样import git项目

### Problem 3: Turtle graphics and drawSquare

forward(units)，直行，turn(angle)顺时针转弯，通过调用这两个方法画出正方形。很简单：

1. turtle.forward(sideLength);
2. turtle.turn(90);
3. turtle.forward(sideLength);
4. turtle.turn(90);
5. turtle.forward(sideLength);
6. turtle.turn(90);
7. turtle.forward(sideLength);
8. turtle.turn(90);

### Problem 5: Drawing polygons

首先是要完成calculateRegularPolygonAngle 这个方法，就是输入正多边形的边数求角度。值得注意的是这里需要不呢整除保留两位小数，能整除就是保留一位.0

用到的公式就是 每个内角角度=180\*（边数-2）/边数，保留精确度的方法是：(double) Math.round(angle \* 100) / 100.0

我这里遇到了一个问题就是用了angle = 180 \* (sides - 2) / sides;但是最后结果总是不对，后来发现这里会自动变成整数运算，所以需要改为angle = 180.0 \* (sides - 2) / sides;

calculatePolygonSidesFromAngle就是根据角度求边数。公式是：边数=360/（180-angle）这里还需要舍入到最近的整数，使用的是math.round()

最后就是绘制这个多边形了，由于缺乏空间想象力，弄了好久才发现turtle每次转弯的话，需要转（180-angle）而不是直接angle

### Problem 6: Calculating Bearings

这个首先需要完成calculateHeadingToPoint给出现在turtle的偏转角度double currentHeading,现在的坐标（int currentX, int currentY），目标的坐标( int targetX,int targetY)计算出需要偏转的角度。我觉得这个很复杂，因为不能直接简单用三角函数的公式，因为边界情况在坐标轴上这样就可能导致除以0的危险。所以我用了最笨的方法先当作锐角做出来，之后再分类讨论，根据在各个坐标轴上还有各个象限内讨论。

具体的细节：

1. **double** x = Math.abs(targetX - currentX);
2. **double** y = Math.abs(targetY - currentY);
3. **double** z = Math.sqrt(Math.pow(x, 2) + Math.pow(y, 2));
4. **double** cos = y / z;
5. **double** angle = Math.toDegrees((Math.acos(cos)));
6. **if** (targetX > currentX && targetY < currentY) {// 目标在第四象限
7. angle = 90 + angle;
8. }
9. **if** (targetX == currentX && targetY > currentY) {// 目标在y轴正方向上
10. angle = 360;
11. }
12. **if** (targetX > currentX && targetY == currentY) {// 目标在x轴正方向上
13. angle = 90;
14. }
15. **if** (targetX < currentX && targetY < currentY) {// 目标在第三象限
16. angle = 180 + angle;
17. }
18. **if** (targetX < currentX && targetY == currentY) {// 目标在x轴负方向
19. angle = 270;
20. }
21. **if** (targetX == currentX && targetY < currentY) {// 目标在Y轴负方向
22. angle = 180;
23. }
24. **if** (targetX < currentX && targetY > currentY) {// 目标在第二象限
25. angle = 360 - angle;
26. }
27. **if** (angle - currentHeading >= 360)
28. **return** Math.round((angle - currentHeading - 360) \* 100) / 100.0;

虽然这样做真的很笨……

然后就是calculateHeadings(List<Integer> xCoords, List<Integer> yCoords)

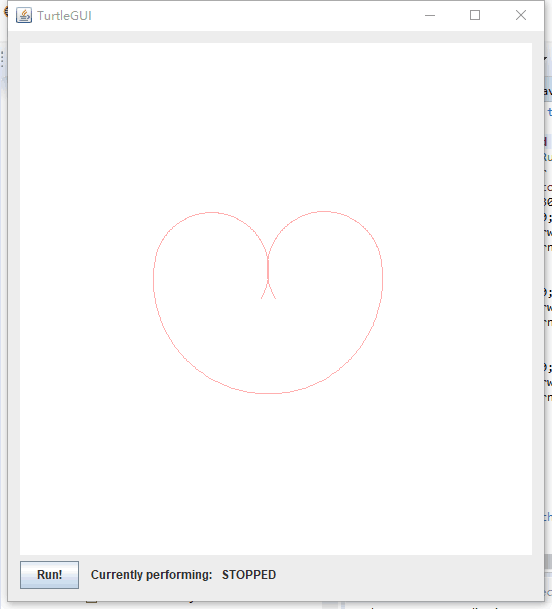
这个只需要对每个现在的x,y和下一个x,y调用之前写好的 calculateHeadingToPoint就好，但是要增加一个List来保存现在角度，第一次的角度默认为0

1. List<Double> vector = **new** ArrayList<>();
2. List<Double> currentHeading = **new** ArrayList<>();
3. currentHeading.add(0.0);
4. **for** (**int** i = 0; i < xCoords.size() - 1; i++) {
5. System.out.println(xCoords.get(i));
6. **double** angle = calculateHeadingToPoint(currentHeading.get(i), xCoords.get(i), yCoords.get(i),
7. xCoords.get(i + 1), yCoords.get(i + 1));
8. vector.add(angle);
9. currentHeading.add((currentHeading.get(i) + angle) % 360);
10. }
11. **return** vector;

### Problem 7: Convex Hulls

### Problem 8: Personal art

利用turn\forward 和for循环画出的爱心：



其中使用PenColor color = PenColor.***PINK***;

turtle.color(color); 来变换颜色

### Submitting

依此使用get add .（把工作时的**所有变化提交**到暂存区），

git commit -m '1160300324' (提交暂存区的文件),

git pull origin master(命令用于取回远程主机master分支的更新与本地的当前分支合并)，

git push origin master（将当前分支推送到origin主机的对应分支。）

## Social Network

这里就是要建立一个网络（其实就是一个图），在里面可以添加人（点）与关系（边），这样我们就可以知道两个人之间是否有联系比如他是我朋友的朋友的朋友。我们来计算之间关系的层数（几个人）来判断关系的远近，这样就像是一个社交网络，继而转化为图的一些操作。

### 设计/实现FriendshipGraph类

在friendshipgraph类里面，我定义了两个整形二维数组，people[MAXN][MAXN];和dis [MAXN][MAXN];其中MAXN为常量200（考虑到后面的算法是O(n3 )所以不能更多。还有一个ArrayList<Person> pername 用来存储每个person,当然实际的作用是储存他们的名字。

首先**public** **void** addVertex(Object obj)用来添加人的方法

1. **public** **void** addVertex(Object obj)
2. {
3. Person temp = (Person) obj;
4. **for** (**int** i = 0; i < pername.size(); i++) {
5. **if** (pername.get(i).name.equalsIgnoreCase(temp.name))
6. **throw** **new** RuntimeException("Each person has a unique name");
7. }  //如果要添加的person的名字已经在列表里面出现过了那么抛出一个异常
8. **this**.pername.add(temp);
9. }

然后就是关于添加边的方法public void addEdge(Object obj1,Object obj2)

1. **public** **void** addEdge(Object obj1,Object obj2)
2. {
3. Person temp1 = (Person) obj1;
4. Person temp2 = (Person) obj2;
5. boolean flag1 =**false** ;
6. boolean flag2 =**false** ;
7. **int** pobj1=-1,pobj2=-1;
8. **for**(**int** i=0;i<pername.size();i++) {
9. **if**(pername.contains(temp1))
10. {
11. flag1 =**true** ;//如果第一个人存在
12. pobj1= pername.indexOf(temp1);//获得数组代表第一个人的下标
13. }
14. }
15. **for**(**int** i=0;i<pername.size();i++) {
16. **if**(pername.contains(temp2))
17. {
18. flag2 =**true** ;//如果第二个人存在
19. pobj2= pername.indexOf(temp2);  //获得数组代表第二个人的下标
20. }
21. }
22. **if**(flag1&&flag2) people[pobj1][pobj2]=1;//如果要添加的两个人都存在的话，那么他们之间的关系就为1（真）
23. **else** **throw** **new**  RuntimeException("we don't have such a person");//两个人至少一个不存在那么抛出异常
24. }

最后获取两个人之间的距离public int getDistance(Object obj1, Object obj2)

1. **public** **int** getDistance(Object obj1, Object obj2)
2. {
3. Person temp1 = (Person) obj1;
4. Person temp2 = (Person) obj2;
5. boolean flag1 =**false** ;
6. boolean flag2 =**false** ;
7. **int** pobj1=-1,pobj2=-1;
8. 根据前面的people[i][j]来填写dis[i][j]
9. 如果两个人有直接关联，即people[i][j]=1，则dis[i][j]也为1
10. 如果没有直接关联，他们直接的距离dis[i][j]则为一个较大的数9999
11. 判断要操作的两个人是否存在，若是存在，获取他们的下标
12. **if**(两个人都存在)
13. {
14. 如果两个人为同一个
15. **return** 0;
16. **else** {
17. floyd算法计算多源最短路径
18. }
19. **if**(两个人之间的距离!=9999) **return** dis[pobj1][pobj2];//两人有关联，返回距离
20. **else** **return** -1;//不存在关联
21. }
22. **else** **throw** **new**  RuntimeException("we don't have such a person");//有人不存在列表中，抛出异常
23. }

### 设计/实现Person类

1. String name;
2. **public** Person (String itsname)
3. {
4. **this**.name = itsname;
5. }

Person类里面只用来储存这个人的名字和所以只有一个String和一个构造函数

### 设计/实现客户端代码main()

1. **public** **static** **void** main(String[] args) {
2. FriendshipGraph graph = **new** FriendshipGraph();
3. Person rachel = **new** Person("Rachel");
4. Person ross = **new** Person("Ross");
5. Person ben = **new** Person("Ben");
6. Person kramer = **new** Person("Kramer");
7. graph.addVertex(rachel);  //图里增加人
8. graph.addVertex(ross);
9. graph.addVertex(ben);
10. graph.addVertex(kramer);
11. graph.addEdge(rachel, ross);  //图里增加边
12. graph.addEdge(ross, rachel);  //而且题目要求关系需要相互增加即A认识B不代表B认识A
13. graph.addEdge(ross, ben);
14. graph.addEdge(ben, ross);
15. System.out.println(graph.getDistance(rachel, ross));
16. System.out.println(graph.getDistance(rachel, ben));
17. System.out.println(graph.getDistance(rachel, rachel));
18. System.out.println(graph.getDistance(rachel, kramer));
19. }

Main函数放在Person类里面，是用题目给出的例子来测试结果。

### 设计/实现测试用例

题目给出的测试用例：

Main函数输出结果是 1 2 0 -1 是符合预期的

其他测试：

1.如果 如果将上述代码的第 10 行注释掉（意即 rachel 和 ross 之间只存在单向的 社交关系 ross->rachel）那么输入的结果是 -1 -1 0 -1也满足预期，因为此时ross不认识rachel,那么ross认识ben，rachel也不会认识ben,此时关系不相互不能传递。

2.如果将第 3 行引号中的“Ross”替换为“Rachel”，那么程序会报错，因为题目默认每个人只有一个名字，而列表里面已经有rachel了。

3.自己写的测试用例：

（用图表示）

输出结果：

1. assertEquals(0, graph.getDistance(senela,senela));
2. assertEquals(1, graph.getDistance(senela,blair));
3. assertEquals(2, graph.getDistance(senela, chunk));
4. assertEquals(3, graph.getDistance(senela, nate));
5. assertEquals(2, graph.getDistance(senela, dan));
6. assertEquals(1, graph.getDistance(senela, jenny));

## Tweet Tweet

### Problem 1: Extracting data from tweets

第一个方法：public static Timespan getTimespan(List<Tweet> tweets)

功能为获取这些tweet的时间跨度，即最早时间到最晚时间。

首先完成测试：

需要测试的部分有 tweet 列表为空，1个tweet,2个tweet两个时间，3个tweet3个时间

伪代码：

1. Instant starttime=Instant.MAX,endtime=Instant.MIN ;
2. **if** (tweets为空){
3. **return** **new** Timespan(Instant.now(), Instant.now());
4. }
5. **else** {
6. **for** (Tweet tweet : tweets) {
7. **if** (tweet.getTimestamp().在endtime之后) {
8. endtime = tweet.getTimestamp();
9. }
10. **if** (tweet.getTimestamp().在starttime之前) {
11. starttime = tweet.getTimestamp();
12. }
13. }
14. **return** **new** Timespan(starttime, endtime);

第二个方法：public static Set<String> getMentionedUsers(List<Tweet> tweets)

功能为返回每个tweet提到的人

首先完成测试，分类包括没有提到任何人，提到一个人，提到多个（三个）人。

1. **public** **static** Set<String> getMentionedUsers(List<Tweet> tweets) {
2. //   throw new RuntimeException("not implemented");
3. Set<String> mentionedUsers = **new** HashSet<>();
4. Pattern p = Pattern.compile("\\B@[A-Za-z0-9\_-]+\\b");//正则匹配“ @xxx  ”
5. **for** (Tweet tweet: tweets) {
6. String text = tweet.getText().toLowerCase();//全部转化为小写
7. Matcher m = p.matcher(text);
9. **while** (m.find()) {
10. String userMentioned = m.group().substring(1);//取@之后的内容
11. mentionedUsers.add(userMentioned);//增加@的人
12. }
13. }
14. **return** mentionedUsers;
15. }

### Problem 2: Filtering lists of tweets

第一个方法**public** **static** List<Tweet> writtenBy(List<Tweet> tweets, String username)

它的功能是返回这些tweets中是username写的，只需要循环然后依此使用getAuthor()即可。

首先完成测试，包括空推特、写了一篇tweet,写了多篇（三篇）tweets

1. **public** **static** List<Tweet> writtenBy(List<Tweet> tweets, String username) {
2. // throw new RuntimeException("not implemented");
3. List<Tweet> tweetswrittenBy = **new** ArrayList<>();
4. **for** (Tweet tweet : tweets) {
5. **if** (tweet.getAuthor().equalsIgnoreCase(username)) {
6. tweetswrittenBy.add(tweet);
7. }
8. }
9. **return** tweetswrittenBy;
10. }

第二个方法是**public** **static** List<Tweet> inTimespan(List<Tweet> tweets, Timespan timespan)它的功能是找到某个时间范围内的所有推特，也只需循环判断时间即可。

首先完成测试：返回包括有0篇、1篇、多篇（三篇）推特

1. **public** **static** List<Tweet> inTimespan(List<Tweet> tweets, Timespan timespan) {
2. **throw** **new** RuntimeException("not implemented");
3. List<Tweet> tweetsinTimespan = **new** ArrayList<>();
4. **for** (Tweet tweet : tweets) {
5. **if** (tweet.getTimestamp().isBefore(timespan.getEnd()) && tweet.getTimestamp().isAfter(timespan.getStart())) {
6. tweetsinTimespan.add(tweet);
7. }
8. }
9. **return** tweetsinTimespan;
10. }

第三个方法是**public** **static** List<Tweet> containing(List<Tweet> tweets, List<String> words)

功能是找到含有words的所有推特，但是要注意大小写的问题所以直接把文本和words全部转换为小写，把每个tweet的文本切割成一个list然后使用List.contains（）方法即可。

首先完成测试，包括没有包含单词，有一篇包含，三篇包含。

1. **public** **static** List<Tweet> containing(List<Tweet> tweets, List<String> words) {
2. **throw** **new** RuntimeException("not implemented");
3. List<String> List = **new** ArrayList<>();
4. **for** (String word : words) {
5. List.add(word.toLowerCase());
6. }
7. List<Tweet> tw = **new** ArrayList<>();
8. **for** (Tweet tweet : tweets) {
10. List<String> wordsnew = **new** ArrayList<String>(Arrays.asList(tweet.getText().split(" ")));
11. **for** (String s : wordsnew) {
12. **if** (List.contains(s.toLowerCase())) {
13. tw.add(tweet);
14. **break**;
15. }
16. }
17. }
18. **return** tw;

### Problem 3: Inferring a social network

第一个方法是**public** **static** Map<String, Set<String>> guessFollowsGraph(List<Tweet> tweets)功能是给出一个Map每个人对应他关注的人（此时的关注就是根据如果有@某人就一定关注他）

首先写测试，关注0人，1人，3人

伪代码：

1. **public** **static** Map<String, Set<String>> guessFollowsGraph(List<Tweet> tweets) {
2. //     throw new RuntimeException("not implemented");
3. Map<String, Set<String>> SocialNetwork = **new** HashMap<String, Set<String>>();
4. List<Tweet> socialtweet = **new** ArrayList<Tweet>(tweets);
5. **for** (Tweet tweet: socialtweet){
6. 获得该推特提到的所有人
7. SocialNetwork.put(改tweet的作者,该推特提到的所有人);
8. **for** (String author: 该推特提到的所有人){
9. **if**(不是自己提到自己){
10. SocialNetwork.get(tweet作者.add(author);
11. }
12. }
13. }
14. **return** SocialNetwork;
15. }

第二个方法是 **public** **static** List<String> influencers(Map<String, Set<String>> followsGraph) 功能是返回一个影响力表，从高到低，这里就是被关注的人越多，影响力越大。

首先完成测试，包括影响力列表含有0人、1人、3人。

伪代码：

1. **public** **static** List<String> influencers(Map<String, Set<String>> followsGraph) {
2. //      throw new RuntimeException("not implemented");
3. List<String> List = **new** ArrayList<String>();
4. 建立一个map，映射每个人，和他的影响力指数（这里指被关注的次数）
5. **int** influenceCount;
6. **for** (每个人){
7. **for**(他关注的人中){
8. **if**(列表里有这个人)){
9. 影响力指数加一
10. }
11. **else**{
12. map加入这个人，影响力为1
13. }
14. }
15. }
17. //sort the influencers and put them in InfluenceList
18. **while**(影响力不为空){
19. 根据每个人的影响力指数，排序建立InfluenceList
20. }
21. **return** List
22. }

### Problem 4: Get smarter

这里我选择增加一个evidence: **Awareness**. If A follows B and B follows C, and B retweets a tweet made by C, then A sees the retweet and is influenced by C. 即如果A follow B ,B follow C ,B有转载过 C的推特，那么C也看到A的推特，A的影响力也会加一。那样AC也存在关系网。

测试是在上一个的基础之上增加有包含“RT @ “的推特。

伪代码：

1. **public** **static** Map<String, Set<String>> guessFollowsGraphplus(List<Tweet> tweets) {
2. //     throw new RuntimeException("not implemented");
3. Map<String, Set<String>> SocialNetwork = **new** HashMap<String, Set<String>>();
4. List<Tweet> socialtweet = **new** ArrayList<Tweet>(tweets);
5. 同guessFollowsGraphp部分；
6. Pattern p = Pattern.compile("rt\\s@[A-Za-z0-9\_-]+\\b");//匹配" @RTXXXX"
7. **for** (每个人) {
8. String text = tweetx.getText().toLowerCase();
9. Matcher m = p.matcher(text);
10. Set<String> RTUsers = Extract.getMentionedUsers(Arrays.asList(tweetx));
12. **while** (m.find()) {
13. 提取出这个人RT的人
14. }
15. 关注这个人的人，如果没有关注他RT的人，那么他和这个RT的人也增加关系。
17. }
18. **return** SocialNetwork;
19. }

# 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。

\*我写完代码以后才发现有进度记表，跟老师说过没关系的。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 任务 | 实际完成情况 |
| 2019-02-25 | 14:00-15:30 | 编写问题1的isLegalMagicSquare函数并进行测试 | 按计划完成 |
|  |  |  | 延期1小时完成 |
|  |  |  | 遇到困难，未完成 |

# 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. Java编程语言是否对你的口味？

我本来没有学过Java，这次做实验的时候基本就是边学边做，需要什么就去查什么的用法，觉得就像是用胶水粘程序一样的感觉，不够流畅那种。但是马上就喜欢上了Java这种有内在哲学的语言。

1. 关于Eclipse IDE

学习它的使用真的用了很长时间,功能超多，但是很喜欢。

1. 关于Git

第一次使用版本控制系统，发现真的很方便。

1. 关于CMU和MIT的作业
2. 关于本实验的工作量、难度、deadline
3. 关于初接触“软件构造”课程

以上三个问题，一个回答：

Hard, difficult, tough, but interesting .