

**2019年春季学期  
计算机学院大二软件构造课程**

**Lab 1实验报告**

|  |  |
| --- | --- |
| 姓名 | 李震宇 |
| 学号 | 1170300110 |
| 班号 | 1703001 |
| 电子邮件 | [1137859144@qq.com](mailto:1137859144@qq.com) |
| 手机号码 | 18800419432 |

**目录**

[1 实验目标概述 1](#_Toc3482968)

[2 实验环境配置 1](#_Toc3482969)

[3 实验过程 1](#_Toc3482970)

[3.1 Magic Squares 1](#_Toc3482971)

[3.1.1 isLegalMagicSquare() 1](#_Toc3482972)

[3.1.2 generateMagicSquare() 2](#_Toc3482973)

[3.2 Turtle Graphics 3](#_Toc3482974)

[3.2.1 Problem 1: Clone and import 3](#_Toc3482975)

[3.2.2 Problem 3: Turtle graphics and drawSquare 4](#_Toc3482976)

[3.2.3 Problem 5: Drawing polygons 4](#_Toc3482977)

[3.2.4 Problem 6: Calculating headings 5](#_Toc3482978)

[3.2.5 Problem 7: Personal art 5](#_Toc3482979)

[3.2.6 Submitting 6](#_Toc3482980)

[3.3 Social Network 7](#_Toc3482981)

[3.3.1 设计/实现FriendshipGraph类 7](#_Toc3482982)

[3.3.2 设计/实现Person类 8](#_Toc3482983)

[3.3.3 设计/实现客户端代码main() 8](#_Toc3482984)

[3.3.4 设计/实现测试用例 9](#_Toc3482985)

[3.4 Tweet Tweet 9](#_Toc3482986)

[3.4.1 Extract 9](#_Toc3482987)

[3.4.2 Filter 11](#_Toc3482988)

[3.4.3 SocialNetwork 12](#_Toc3482989)

[3.4.4 Test 15](#_Toc3482990)

[4 实验进度记录 17](#_Toc3482991)

[5 实验过程中遇到的困难与解决途径 17](#_Toc3482992)

[6 实验过程中收获的经验、教训、感想 17](#_Toc3482993)

# 实验目标概述

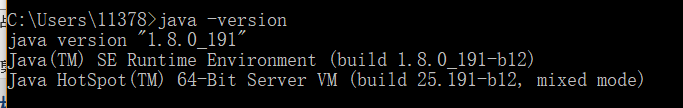
本次实验通过求解四个问题，训练基本Java编程技能，能够利用Java OO开发基本的功能模块，能够阅读理解已有代码框架并根据功能需求补全代码，能够为所开发的代码编写基本的测试程序并完成测试，初步保证所开发代码的正确性。另一方面，利用Git作为代码配置管理的工具，学会Git的基本使用方法。

* 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

IDE : Eclipse

JAVA版本信息：



URL：<http://github.com/ComputerScienceHIT/Lab1-1170300110.git>

遇到的困难：

配置git的时候有些晕。

# 实验过程

## Magic Squares

了解幻方的一些性质以及如何构造幻方。

### isLegalMagicSquare()

按行读入文件，添加入Arralist中，判断其中每一个String的长度是否相等，若不相等，直接返回false。

是否含有’-’、’.’、’ ’的非法字符，如果有直接返回false。

使用split(“\t”)的方法将其隔开分别放入一个二维数组中。分别对其每一行每一列、两个对角线进行求和，如果发现其有不相等之处就返回false。

如果其满足所有幻方的性质，则返回true。

### generateMagicSquare()

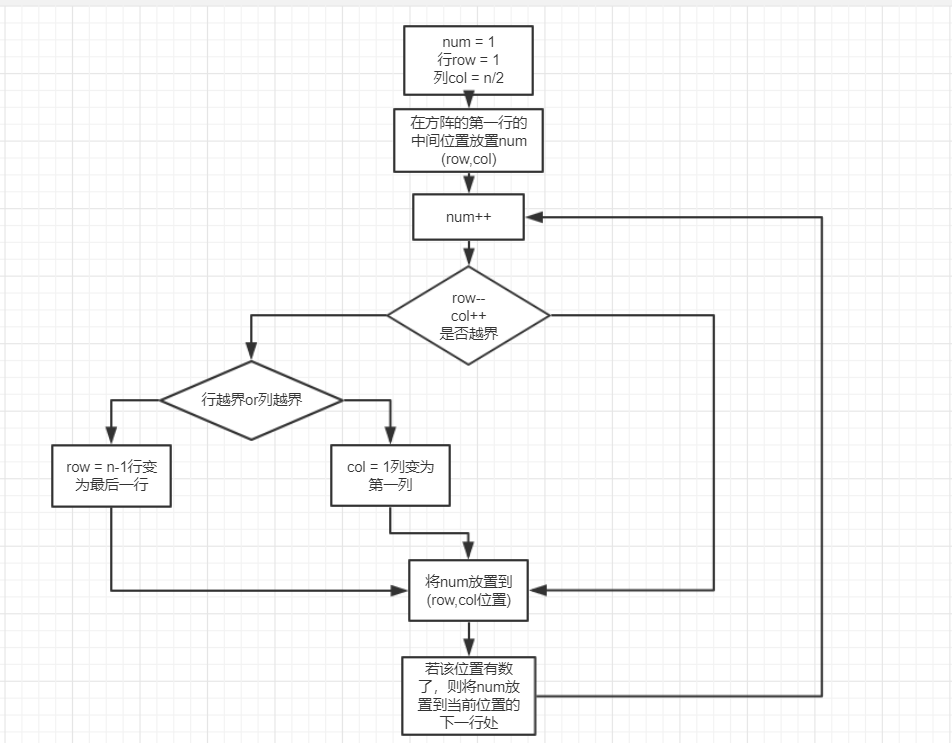
#### 整体思路：

这里只要求对行、列数为奇数的方阵构造幻方。首先将1放到第一行的中间的位置，然后试图将2放到1的右上方，如果行数越界，那么放置到最后一行，列数越界就放到第一列，如果试图防止数字的位置已经被放置过了数字，那么将该数字放到当前数字的下方，每一次放置的数字都比上个数字大1，直到填充满这个方阵，便得到了幻方。

#### 核心代码：

1. **for** (i = 1; i <= square; i++) {
2. magic[row][col] = i;
3. **if** (i % n == 0)
4. /// 每输入了n个数字，下一个数字出现在前一个数字的下方，即row++即可。
5. row++;
6. **else** {
7. /// 总结，尝试放到右上方，如果越界，则回旋回去。
8. **if** (row == 0)
9. /// 如果向上越界了，那么下一个数字出现在最后一行。
10. row = n - 1;
11. **else**
12. /// 每次尝试放到当前数字的上一行。
13. row--;
14. **if** (col == (n - 1))
15. /// 如果向右越界了，那么下一个数字出现在第一列。
16. col = 0;
17. **else**
18. /// 每次尝试放到当前数字的下一列。
19. col++;
20. }
21. }

#### 流程图：



当n为偶数时：

在进行算法实际操作时，最后一行的某一个列已经被填充过了，算法将尝试在第n行的某一个列处放置num，这会导致数组越界，并不会自动调整到从第一行开始填充，所以会报错：java.lang.ArrayIndexOutOfBoundsException

当n为负数时：

创建一个长度为负数的数组，会报错：java.lang.NegativeArraySizeException

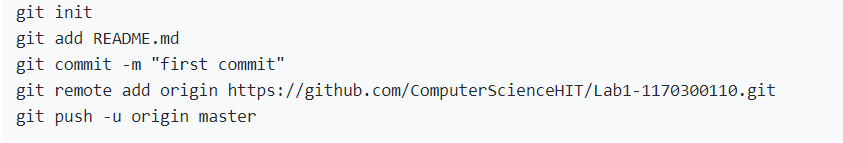
## Turtle Graphics

在这里简要概述你对该任务的理解。

### Problem 1: Clone and import

从github上下载代码，import到当前的工程文件中。然后调整import、调整包名即可。

git操作：



### Problem 3: Turtle graphics and drawSquare

画一个正方形：

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);

直走转弯，直走转弯，直走转弯，直走回到原点即可。

### Problem 5: Drawing polygons

#### 计算正多边形内角大小：

规律的公式体现 angle = 180\* ((n-2)/n)。

#### 给定某个内角计算其为正几边形：

通过多边形外角和为360°来计算。

#### 画出一个多边形：

首先是肯定要用for循环，实现转一个角度，画一个边，这个循环操作，n边形，将重复n次。

画的边的长度是输入的参数，然后转过的角度，可以通过调用函数: calculateRegularPolygonAngle(sides)来计算需要转过的角度，注意这里要向右出发，所以刚开始就要turn90°，之后每次旋转都要turn180°来调整角度。

核心代码：

1. **for** (**int** i = 1; i <= sides; i++) {
2. turtle.forward(sideLength);
3. /// 调头 hh
4. turtle.turn(90);
5. turtle.turn(90);
6. turtle.turn(angle);
7. }
8. };

### Problem 6: Calculating headings

通过当前点的坐标、目标点的横纵坐标来计算tan值，然后使用反tan的

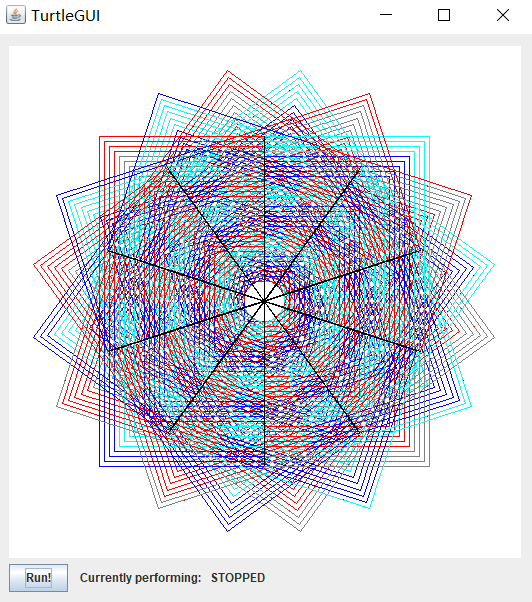
三角函数，计算出夹角大小，然后根据当前的朝向，计算转角大小，注意，这个转角应该为正数，所以要判断一下，当其为负时要+360°，将其调整为一个正的角度。

下一个函数就是将上一个函数多调用几次，注意一开始时这个初始角度为0°即可，主要循环如下：

1. **for** (**int** i = 1; i <= nums - 2; i++) {
2. turn = calculateBearingToPoint(turn, xCoords.get(i), yCoords.get(i), xCoords.get(i + 1),
3. yCoords.get(i + 1));
4. turningSet.add(turn);
5. }

### Problem 7: Personal art

首先是展示一下我自己画的图形：



主要是通过循环，旋转一个长方形，按循环，每次都增加步长并切换颜色，核心代码：

1. **while**(j <= 30) {
2. **while**(i <= 10)
3. {
4. turtle.forward(x);
5. turtle.turn(90);
6. turtle.color(PenColor.BLUE);
7. turtle.forward(x);
8. turtle.turn(90);
9. turtle.forward(x);
10. turtle.color(PenColor.RED);
11. turtle.forward(x);
12. turtle.turn(90);
13. turtle.forward(x);
14. turtle.turn(90);
15. turtle.color(PenColor.BLACK);
16. turtle.forward(2\*x);
17. turtle.turn(270);
18. turtle.color(PenColor.GRAY);
19. turtle.forward(x);
20. turtle.turn(270);
21. turtle.forward(x);
22. turtle.color(PenColor.CYAN);
23. turtle.forward(x);
24. turtle.turn(270);
25. turtle.forward(x);
26. turtle.turn(270);
27. turtle.color(PenColor.BLACK);
28. turtle.forward(x);
29. turtle.turn(36);
30. i++;
31. }
32. x = 20 + j \* 5;
33. i = 0;
34. j++;
35. }

### Submitting

git add

git commit -m “First commit”

git push -u origin master

## Social Network

根据人作为顶点，之间的关系(长度)作为权值，构建邻接矩阵并通过floyd算法，计算两点之间最短距离(权值)。

### 设计/实现FriendshipGraph类

维护一个距离二维数组。作为邻接矩阵。构建一个加入的人的Arraylist，在这个Arraylist中的位置作为二维数组的索引来寻找权值。在这个二维矩阵中，因为要使用floyd算法，所以每一个没有路径的人之间的权值为inf=999。

#### public boolean addVertex(Person name)：

向图中增加顶点，每次增加要判断Arraylist中是否有重名如果有，报错推出。然后每次增加定点后都要初始化一下二维数组。

#### public boolean addEdge(Person name1, Person name2)：

向图中加边，首先要从Arraylist中，找到name1、name2的index作为索引，然后将二维数组[index1][index2]处的权值置为1。

#### public int getDistance(Person name1, Person name2)：

返回两个人之间的距离，这个函数中要先使用floyd算法，计算出任意两点之间的最小权值，核心代码：

1. **for**(**int** k = 0; k < nums; k++) {
2. **for**(**int** i = 0; i < nums; i++) {
3. **for**(**int** j = 0; j < nums; j++) {
4. **if**(distance[i][k] + distance[k][j] < distance[i][j]) {
5. distance[i][j] = distance[i][k] + distance[k][j];
6. }
7. }
8. }
9. }

之后，从Arralist中找到name1、name2的index作为索引，然后从二维数组[index1][index2]处提取权值，如果权值为inf则说明没有路，则按题意返回-1，如果index1=index2，则返回0。

### 设计/实现Person类

写的很简单，没有什么关键的操作：

1. **public** **class** Person {
2. String name;
4. **public** Person(String name) {
5. **this**.name = name;
6. }
7. }
8. }

### 设计/实现客户端代码main()

按实验要求构造测试样例，主要检查添加用户名相同、返回相同人名的距离

的情况。

1. FriendshipGraph graph = new FriendshipGraph();
2. Person rachel = new Person("Rachel");
3. Person ross = new Person("Ross");
4. Person ben = new Person("Ben");
5. Person kramer = new Person("Kramer");
6. graph.addVertex(rachel);
7. graph.addVertex(ross);
8. graph.addVertex(ben);
9. graph.addVertex(kramer);
10. graph.addEdge(rachel, ross);
11. graph.addEdge(ross, rachel);
12. graph.addEdge(ross, ben);
13. graph.addEdge(ben, ross);
14. System.out.println(graph.getDistance(rachel, ross));

//should print 1

1. System.out.println(graph.getDistance(rachel, ben));

//should print 2

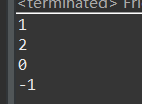
1. System.out.println(graph.getDistance(rachel, rachel));

//should print 0

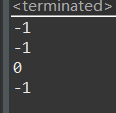
1. System.out.println(graph.getDistance(rachel, kramer));

//should print -1

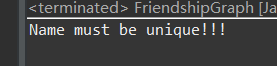
测试结果：



第十行注释掉：

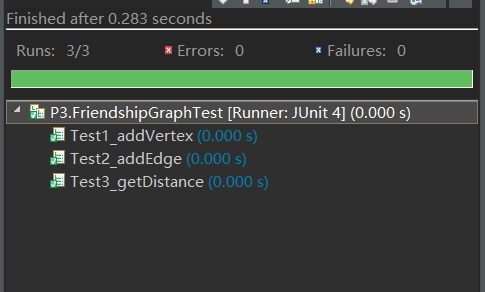
 满足期望。

修改第三行：

满足期望。

### 设计/实现测试用例

跑他给的text，结果如下：



## Tweet Tweet

### Extract

#### public static Timespan getTimespan(List<Tweet> tweets)

这个函数是为了得到一些推特被发表的时间范围，那么取第一条推特的时间赋值给max、min。

然后遍历这个推特列表，如果有推特的发表时间小于min，就用这个时间更新min，同理，如果有推特的时间晚于max，就用这个时间更新max。

最后Timespan timespan = new Timespan(min, max);构建这个timespan，然后返回它即可。

#### public static Set<String> getMentionedUsers(List<Tweet> tweets)

这个函数是为了得到一些推特中提及到的用户名。这个格式应该是@xxxx。

那么，我们需要遍历这些推特中的每一条text，将其split(“ ”)用空格隔开，然后每个单词每个单词的遍历，找到以首字母@开头的String，这个就是要处理的关键部分了。我们需要判断@后面的单词是否合法(是否为一个用户名)，然后看结果集合中是否有这个用户名，如果没有，就将其添加到结果结合中。

对@xxx进行操作：对xx中每一个字符，判断其是否在0~9A~Za~z\_-中，如果有不在的既为不合法字符。

而且我们要小心@xxx.....这种结构，这种结构xxx也是可能合法的。

核心代码：

1. **if** (j.charAt(0) == '@') {
2. ///如果发现某一个字符串的第一位是@
3. nums = 0;
4. flag = **false**;
5. /// 找到@ xxxx
6. name = j.split("[, ? @ : ' \" ! .]");
7. /// 为了防止名字后面跟着字符，再split一下
8. /// 引入计数器nums 记录分词后 得到字符串的数量，如果有两个就代表了形式：xxx.xxx不合法不用加入
9. **for** (**int** k = 0; k < name.length; k++) {
10. **if** (!name[k].isEmpty()) {
11. nums++;
12. **if** (ifLegal(name[k])) {
13. ///判断名字是否合法
14. n = name[k];
15. /// n就是那个名字
16. flag = **true**;
17. }
18. }
19. }
20. **if** (nums == 1 && flag == **true**) {
21. **if** (!ifContain(result, n)) {
22. ///检查result集合中是否含有n 如果没有则加入进去
23. result.add(n.toLowerCase());
24. }
25. }
26. }

其中判断是否合法的函数很简单，就是将那个字符的ASCII码值比较一下是

否在那个范围之间，如果不再返回false，如果都在返回true。

### Filter

#### public static List<Tweet> writtenBy(List<Tweet> tweets, String username)

这个函数是用来找到所有username写过的tweets集合。

很简单，就是遍历所有推特，然后如果get其作者名，如果相等就添加到

结果集合中去。

最后返回结果集合。

#### public static List<Tweet> inTimespan(List<Tweet> tweets, Timespan timespan)

这个函数是用来找到某个时间段内发表的所有tweets集合。

也很简单，就是遍历所有推特，然后如果get其发表时间，如果这个时间在

这个timespan时间区间内，就将这条推特添加到结果集合中去。

最后返回结果集合。

#### public static List<Tweet> containing(List<Tweet> tweets, List<String> words)

这个函数是用来将所有含有words的推特收集到结果List中去的。

依次遍历所有推特，然后提取每一条text的字符串。和上面一个函数很类似，在这里我们也需要将其按空格划分开来，然后一个单词一个单词的遍历，看这些单词是否在words列表中，如果在words列表中，就将该条推特加入到结果集合中，如果不再就继续遍历，直到所有推特都被遍历了一遍即可。

随后返回结果集合。

核心代码：

1. **for** (**int** i = 0; i < tweets.size(); i++) {
2. /// 设立flag 方便进行判断操作（加入列表中）
3. textWord = tweets.get(i).getText().split(" ");
4. **for** (**int** j = 0; j < textWord.length; j++) {
5. word = textWord[j];
6. /// 如果存在那么就加入列表中并且break掉后面的内容 无需继续判断啦
7. **if** (ifContain(word, words)) {
8. ///注意这里的ifContain要进行符号的处理 比如@之类的
9. result.add(index,tweets.get(i));
10. index++;
11. **break**;
12. }
13. }
14. }

### SocialNetwork

#### public static Map<String, Set<String>> guessFollowsGraph(List<Tweet> tweets)

这个集合是需要求得一个映射关系：

Key：用户名，Value：该用户所有推特中@过的用户。

##### 初步实现

第一步很简单，就是遍历整个推特，将所有用户名相同的推特放到一个集合

中然后一起调用getMentionedUser函数，这样就可以直接返回他所有@过的用户了。

核心代码：

1. **for** (Tweet i : tweets) {
2. name = i.getAuthor();
3. ///搜集所有作者名字相同的推特
5. **if** (times == 0) {
6. ///搜集所有作者名字相同的推特
7. **for** (Tweet j : tweets) {
8. **if** (j.getAuthor().toLowerCase().equals(i.getAuthor().toLowerCase())) {
9. tweetsOnes.add(j);
10. }
11. }
12. //搜集完一起加进去
13. nameSet = Extract.getMentionedUsers(tweetsOnes);
14. follow.put(name, nameSet);
15. times++;
16. } **else** {
17. **if** (!follow.containsKey(name.toLowerCase())) {
18. **for** (Tweet j : tweets) {
19. **if** (j.getAuthor().toLowerCase().equals(i.getAuthor().toLowerCase())) {
20. tweetsOnes.add(j);
21. }
22. }
23. nameSet = Extract.getMentionedUsers(tweetsOnes);
24. follow.put(name, nameSet);
25. }
26. }
27. //每次进行一个循环后将这个tweets集合清空
28. tweetsOnes.clear();
29. }

##### Get Smarter 1：

将所有#相同内容的用户设置为互相关注。

处理方法也很相似，遍历推特的text内容，寻找#标志，然后返回#后的内容，收集到一起，然后进行遍历，看两个用户是否有相同的#的内容，如果有就互相在MAP映射中添加对方。

核心代码：

1. **for** (Tweet i : tweets) {
2. name1 = i.getAuthor();
3. hashtags1 = getHashtags(i);
4. **for** (Tweet j : tweets) {
5. name2 = j.getAuthor();
6. /// 在不相同的tweet中遍历
7. **if** (!j.equals(i)) {
8. hashtags2 = getHashtags(j);
9. /// 得到里面的hashtag
10. **if** (ifContainEquel(hashtags1, hashtags2)) {
11. /// 如果含有相同的那么要增加啦~
12. /// 但是需要在原映射中不存在 这个关系 所以要再判断。
13. /// 只需要增加一半，另一半将在for循环中再次加入
14. nameSet = follow.get(name1);
15. **if** (!nameSet.contains(name2)) {
16. nameSet.add(name2);
17. follow.put(name1, nameSet);
18. }
19. }
20. }
21. }
22. }

##### Get Smarter 2：

如果A和B互相@，B和C互相@，那么我们认为AC之间也存在者互相@的关系。

这个思想也是比较简单，遍历整个推特集合，然后再遍历某个用户@过的用户的@其他人的集合，如果发现他们两个互相@过，那么这个某个用户@过的用户就相当于是刚刚关系中的B,再以B为起点，寻找BC互相@，如果找到就将AC的这个互相@的关系加入map映射中。

通过叙述我们可以感受到，这个算法的思想其实很简单，但是我们也发现了，它有很多层循环，所以在遍历的时候就会很麻烦，我们要注意for疯狂嵌套的时候不要混乱。

核心代码太长了，嵌套的部分有些多，在这里就不贴出来了。

##### Get Smarter 3：

如果A@B，B转发了C的推特，那么我们猜测A也会followC。

这里，我写了一个retweetMap函数，来得到某个user转发过的推特的用户名。由转发格式：RT @xx来确定，方法也是split然后再挨个判断，这里不再赘述。那么就是遍历所有map的键值对，如果某个key的value中的某个元素，转发过别人的推特(x)，那么就将那x加入到key的value中去，然后继续遍历。直到结束，我们就获得了更加smarter的followGraph。

这个核心代码也太多层嵌套了，也不贴出来了。

#### public static List<String> influencers(Map<String, Set<String>> followsGraph)

按影响人数排序，返回用户名的一个列表。

这个就比较简单了，遍历上个函数的到的map，维护一个String[]name和一

个int[]nums的数组，每次一个name被@，那它所对应的nums[index]++。

将这个map都遍历之后，将得到的这个name按照nums从大到小排序，然后依次添加到结果列表中即可。

核心代码：

1. **while** (iteratorMap.hasNext()) {
2. Map.Entry<String, Set<String>> next = iteratorMap.next();
3. **if** (cnt == 0) {
4. ///第一次进入
5. names = next.getValue();
6. **for** (String i : names) {
7. name[index] = i;
8. nums[index] = 1;
9. index++;
10. bounder++;
11. cnt++;
12. }
13. } **else** {
14. /// names 是该条推特含有的提及到的人的名字
15. names = next.getValue();
16. **for** (String i : names) {
17. **if** (ifContain(name, i, bounder)) {
18. ///如果有了就找到那个人的对应的下标
19. index = getIndex(name, i, bounder);
20. ///被提及数+1
21. nums[index]++;
22. } **else** {
23. ///如果没有就再创建新的
24. name[bounder] = i;
25. nums[bounder] = 1;
26. ///注意边界+1
27. bounder++;
28. }
29. }
30. }
31. }

### Test

text样例：

1. Instant d1 = Instant.parse("2016-02-15T10:00:00Z");
2. Instant d2 = Instant.parse("2016-02-16T11:00:00Z");
3. Instant d3 = Instant.parse("2016-02-18T11:00:00Z");
4. Instant d4 = Instant.parse("2016-02-19T11:00:00Z");
5. Instant d5 = Instant.parse("2016-02-19T11:00:00Z");
6. Instant d6 = Instant.parse("2016-02-14T11:00:00Z");
7. Instant d7 = Instant.parse("2016-02-14T11:00:00Z");
8. Instant d8 = Instant.parse("2016-02-14T11:00:00Z");
9. Instant d9 = Instant.parse("2016-02-14T11:00:00Z");
10. Instant d10 = Instant.parse("2016-02-14T11:00:00Z");
12. Tweet tweet1 = **new** Tweet(1, "a", "@b", d1);
13. Tweet tweet2 = **new** Tweet(2, "b", "@a @c", d2);
14. Tweet tweet3 = **new** Tweet(3, "c", "@b", d3);
15. /// abc 应该互相关注
16. Tweet tweet4 = **new** Tweet(4, "d", "#abb", d4);
17. Tweet tweet5 = **new** Tweet(5, "e", "#abb", d5);
18. /// de 互相关注
19. Tweet tweet6 = **new** Tweet(6, "f", "@g", d6);
20. Tweet tweet7 = **new** Tweet(7, "g", "RT @h: balabala", d7);
21. Tweet tweet8 = **new** Tweet(8, "h", "balabala", d8);
22. /// f follows g ///  g follows h /// g retweet h  -> f follows h
23. Tweet tweet9 = **new** Tweet(9, "i", "@h", d9);
24. Tweet tweet10 = **new** Tweet(10, "j", "@lc", d10);
25. /// h > a/b/c > d/e/g > lc (i,j无人follow)

根据我们写的代码，注意三个get smarter，结果应该是：

h abc(乱序) deg(乱序) lc

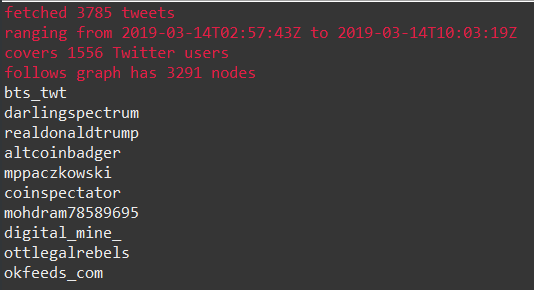
我们观察结果：



成功！

最后我们运行一下最后的main函数：

结果：



# 实验进度记录

请尽可能详细的记录你的进度情况。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 2018-02-28 | 晚间 | 编写问题1 | 按计划完成 |
| 2019-03-01  2019-03-02 | 晚间  下午 | 编写问题2 | 按计划完成  按计划完成 |
| 2019-03-05  2019-03-06 | 晚间  下午 | 编写问题3 | 遇到困难，未完成  按计划完成 |
| 2019-03-08  2019-03-09  2019-03-10  2019-03-11 | 晚间  下午  晚间  晚间 | 编写问题4 | 遇到困难，未完成  遇到困难，未完成  遇到困难，未完成  按计划完成 |

# 实验过程中遇到的困难与解决途径

首先是git下载，在官网上下载速度很慢，找了一大顿在腾讯软件库里下载到了。然后git的操作流程不太熟练，构建本地库、push操作都研究了一段时间，然后eclipse的操作不太熟练，import已存在工程、构建工程文件目录都费了一段时间，这里主要是和室友一起研究了一下。

一开始对java语言的一些语法什么的有些忘记，看了一阵网课。然后有很多方法都不知道，就用JDK查找，JDK找不到，或者看不懂就再百度。

凸包那里有一个赋初值，一开始写到了循环外部，一直没有发现导致DEBUG了好长时间。

然后tweet那里，对MAP的一些方法很不熟悉，遍历的时候有毛病，然后总是报错说无法再强循环的时候对map中的value进行更改，采用的解决办法是重新建立一个新的副本。后来上课的时候感觉到应该是用了变化的变量。地址中的那个值一边，所有的都变了，那个迭代器为了放置这个事情的发现，用hax防止在遍历的时候对其进行修改了。

# 实验过程中收获的经验、教训、感想

本节除了总结你在实验过程中收获的经验和教训，也可就以下方面谈谈你的感受（非必须）：

1. Java编程语言是否对你的口味？

感觉比C强多了。

1. 关于Eclipse IDE

很棒的编程体验~

1. 关于Git

还不太懂但是感觉很厉害。

1. 关于CMU和MIT的作业

英文有些要求看不懂、或者总是落。

1. 关于本实验的工作量、难度、deadline

都比较适中吧。

1. 关于初接触“软件构造”课程

感觉自己还有很多要学习的东西，希望能够好好加油！