**华南农业大学**

**《算法分析与设计》  
课程实验**

专业年级： 2014级信息与计算科学3班

学生学号： 201430120308

学生姓名： 何 唯

实验题目： 雷达覆盖(贪心算法)

指导老师： 赵 峰

实验时间： 2017/04/18

1. **实验题目**

• [题目描述]

假设海岸线是一条无限延伸的直线，陆地在海岸线的一侧，海洋在另外一侧。每个小岛相当于海洋侧的一个点。坐落在海岸线上的雷达设备只能覆盖半径为d的范围。应用直角坐标系，将海岸线作为x轴，设海洋侧在x轴上方。给定海洋中各小岛的位置，以及雷达的覆盖半径，要求用最少的雷达覆盖所有小岛。

• [输入说明]

小岛数目，雷达覆盖半径，各小岛坐标（第一行为小岛数与雷达半径，后面各行为小岛坐标）

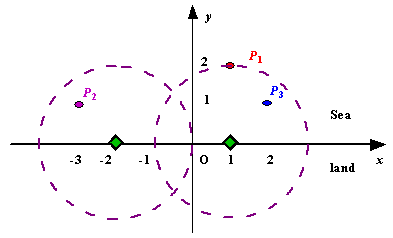
• [输出说明]

覆盖小岛的最少雷达数目，如果无解，输出-1

• [实验要求]

1.至少运行3组不同的输入数据；

2.用两种方法求解：动态规划、备忘录法.



输出样例

2

输入样例

3 2

1 2

-3 1

2 1

**实验实现**

测试数据3

9 3  
0 2  
-3 2  
-4 2  
-5 2  
3 2  
6 2  
9 2  
12 2  
15 2

测试数据2

5 4  
-2 2  
1 2  
0 3  
3 3  
4 3

测试数据1

3 2

1 2

-3 1

2 1

预计输出1

2

预计输出2

1

预计输出3

4

**• 1．**[编程语言]:实验用Java完成算法设计和程序设计并调试通过。

**• 2．**[解题思想、方法]

**符号说明**

|  |  |
| --- | --- |
| **符号** | **含义** |
| **total** | 输入小岛总数 |
| **radius** | 雷达覆盖圆形区域的半径 |
| **Coordinates[][]** | 表示各个小岛的坐标 |
| **augment** | 表示某一小岛的雷达覆盖范围为对应横坐标**augment** |
| **Radar** | 表示保存雷达覆盖范围的对象，其中**Radar.Min**表示左区间坐标，**Radar.Max**表示右区间坐标 |

**算法描述**

本雷达覆盖问题采用贪心算法进行实现，贪心策略采用当前安装的雷达能覆盖更多的小岛，可产生最优覆盖问题的最优解。本问题首先通过小岛的坐标计算出此小岛需要雷达的有效区间，左右区间的值分别保存在一个Radar对象中，并且根据Radar对象区间最大值进行升序排列。

首先，据此，问题的第一个贪心选择是第一个Radar对象的区间最大值作为第一个雷达的建立点，后面考虑下一个区间，如果该区间内雷达为空，则选区间最右边的端点放雷达，并删掉左端点在雷达前的区间，进入子问题。并且，在每个区间只能有一个雷达(多了明显不是最优解)。现在要证明此贪心选择一定是在此覆盖问题的某一最优解中。由于所有的坐标已经按照右区间大小升序排列，下面可以分两种情况讨论：

，即除了最左边的区间外所有的雷达覆盖区间都不与第一区间交叉，此时将雷达建在上必然是在最优解内的。

,即存在区间i与最左边区间交叉。此时，可以由于是交叉区域最右点，容易证明在交叉区域中选择比其他位置可覆盖更多的接近下一个小岛区域的，无论有多少个。（可以将整片交叉区域共同考虑，因为需要尽可能的覆盖右边的小岛，所以在满足最左边小岛覆盖的情况下，必然建雷达在越右越好）

**综上所述，可以得出总存在以贪心选择开始的最优覆盖方案。**

下面证明所采用的贪心策略具有“贪心选择性质”

**贪心选择性质**

设Radar对象按Radar.Max从小到大排好序，雷达覆盖最优解为其中表示第i个雷达的横坐标。当时，必然不可行，小岛1没有被覆盖，不是最优解。因此只需说明的情况。

1.当的时候，必然是一个满足贪心选择性质的最优解，因为要覆盖必须要使，是在某一最优解中的；

2.当时，由于所有小岛在对应Radar排序的时候也就按x坐标递增做好了相应排序，由于，对于区间:

* 如果区间与区间没有交集的时候，必然是在某一最优解中的。
* 如果区间与区间有交集的时候,设交叉区域为，当在某一最优解中，必然也在此最优解中

**综上所述，雷达覆盖问题具有贪心选择性质**

**最优子结构性质**

由上述讨论可以知道贪心策略为，一开始从最左区间考虑，将其右区间端点作为建站目标点，即如果该区间内雷达为空为空，则选区间最右边的端点放雷达，并删掉左端点在雷达前的区间，进入子问题。

设对于问题最优解是一个满足贪心选择性质的最优解，即，则容易知道代表选择后，去除左端点在雷达前的区间的子问题最优解。由于第一次贪心选择后，对应删除了一些在建站可覆盖的区间，所以可以说明，在此可以回到前面贪心选择性质的说明来证明是该子问题的最优解。

**所以，综上所述本问题是具有最优子结构性质的。同时，可以采用贪心算法对问题进行求解。**

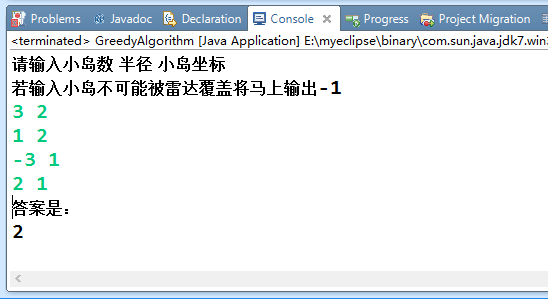
**• 3．**[实验说明，包括输入、输出、结果截图]

**本实验测试数据，在题目要求的三个测试输出的基础上，添加了两个有代表性的测试用例，具体如下：**

**测试一：**

输入：小岛数为3，雷达半径是2，小岛坐标依次为(1,2)(-3,1)(2,1)

输出：2

****

**图 测试一**

**测试二：**

输入：小岛数为5，雷达半径是4，小岛坐标依次为(-2，2)(1,2)(0,3)(3,3)(4,3)

输出：1

****

**图 测试二**

**测试三：**

输入：小岛数为9，雷达半径是3，

小岛坐标依次为(0，2)(-3,2)(-4,2)(-5,2)(3,2)(6,2)(9,2)(12,2)(15,2)

输出：4

****

**图 测试三**

**测试四：**

输入：小岛数为5，雷达半径是19，小岛坐标依次为

(23.115，18.44)(-5.23,8.441)(-8.14,10.2214)(6.1458,2.22)(3.699,7.894)

输出：2

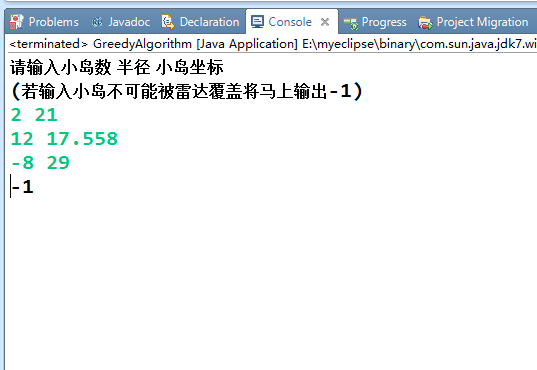
****

**图 测试四**

**测试五：**

输入：小岛数为2，雷达半径是21，小岛坐标依次为(12，17.558）(-8,29)

输出：-1

****

**图 测试五**

**• 4．**[算法复杂性计算过程]

**4.1 时间复杂性**

对于雷达覆盖问题来说，假如事先输入的小岛坐标没有按照x轴坐标进行排序的话，程序需要调用*SortRadar*()方法对雷达可建区间进行排序。该排序算法，这里使用的是链表的冒泡排序时间复杂度为(可以使用快速排序等)。

假如事先已经排好小岛坐标以及雷达可建区间的话，只需考虑两层嵌套for循环，最坏情况下时间复杂度为

**4.2空间复杂性**

从上面算法可知 ，用到了两个数组 ，分别为coordinate[n][2]，flag[n]。空间复杂性主要由coordinate[n][2]决定，为 0(2×n)。因此，该算法的空间复杂性为0(2×n) ，可以应用到即使是小内存、低速度的计算机上。因此本算法有广阔的应用前景。

**小结和心得**

[例如：设计中遇到的问题，如何解决，实验心得，对算法改进的设想等]

* 本次算法实验考察的是对贪心算法的应用以及设计。设计中遇到的问题首先是贪心策略的选择，选择有很多比如可以选从最右区间开始按左区间大小进行排序，通过优先选择最右区间最大的区间进行建站。这里选择了从最左区间开始考虑，以区间右端大小进行排序，优先建立雷达站在右区间端点最小的坐标处。本题目的贪心策略选择是需要一定转换计算才得到的，不能从已有的数据中获取贪心策略的判断数据。比如说建站区间的坐标等等。
* 其次，就是对贪心算法中的贪心选择性质以及最优子结构性质进行证明的难度上，本人一开始是对如何证明没有头绪的，后来是参考了课本中的证明才有了自己的想法。
* 实验的心得的话，就是贪心算法本身看起来比较容易想的出来，但是如何说明贪心算法得到的就是最优解是有一定难度的。相比于动态规划或者是分治法这些从一开始就明确得到的是最优解的算法，贪心算法的使用是需要一定证明的。这也是麻烦之处，但是每个算法都有自己的优缺点，平时自己使用的时候需要对应特定条件进行特定的选择。
* 最后，说到算法的改进设想的话，首先是对排序算法进行一定的改进，如何选择合理的算法是一方面的解决措施。比如选择快速排序、插入排序等常见排序算法是需要对给定数据分析来选择的，比如数据给出的时候是完全倒序，这样用快速排序是比较麻烦的。当数据量比较大的时候也是可以考虑桶排序等等。
* 另一方面就是对两层嵌套for循环的改进上面，可以考虑描述贪心选择上通过链表先排好给定区间，然后每次判断完该区间已经有雷达建站的时候，将该区间进行弹出，下一次比较的时候不需要考虑此区间了。这样能有效的减少对比次数。（其实本程序在后期已经更新使用这样的方法了）
* 总的来说，这次实验学到的东西还是比较多的，第三次算法实验已经是对算法分析有了自己的一些小理解了，对前面学习数据结构的相关存储结构知识也有复习作用，后面希望能继续通过学习不断巩固自己的知识面。

**源程序清单**

**package** Algorism;

**import** java.awt.Point;

**import** java.util.ArrayList;

**import** java.util.Iterator;

**import** java.util.LinkedList;

**import** java.util.Scanner;

**import** org.junit.Test;

/\*\*

\*

\* **@Project**：random

\* **@File**：GreedyAlgorithm

\* **@Author**：何唯

\* **@Date**：2017年4月18日 下午9:12:32

\* **@Description**：

\*/

**public** **class** GreedyAlgorithm {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Scanner scan = **new** Scanner(System.*in*);

System.*out*.println("请输入小岛数 半径 小岛坐标 \n(若输入小岛不可能被雷达覆盖将马上输出-1)");

**int** total = scan.nextInt();// 小岛个数

**double** radius = scan.nextDouble();// 雷达半径

**double**[][] coordinate = **new** **double**[total][2];// 将各个小岛坐标设置为一个double二维数组

// 接收小岛坐标

**for** (**int** i = 0; i < total; i++) {

coordinate[i][0] = scan.nextDouble();

coordinate[i][1] = scan.nextDouble();

**if** (radius < coordinate[i][1]) {

// 若输入的雷达半径小于小岛纵坐标，即不可能覆盖

System.*out*.println("-1");

**return**;

}

}

/\*

\* for(int[]a:coordinate) for(int k:a) System.out.println(k);

\*/

System.*out*.println("答案是：");

System.*out*.println(*findRadarNum*(coordinate, radius, total));

}

// 设计函数findRadarNum来求出问题最优解

**private** **static** **int** findRadarNum(**double**[][] coordinate, **double** radius,

**int** total) {

// **TODO** Auto-generated method stub

**int** ans = 0;

**int** i;

// 声明布尔数组表示该小岛是否已被覆盖

**boolean**[] flag = **new** **boolean**[total];

// 以小岛为圆心，雷达探测半径为半径进行画圆，该圆与x轴交点即为需要雷达建造的范围（无交点的情况已输出为-1，有一个交点的情况将该交点看成两个来处理）

// 将雷达能覆盖的x轴区域最大值最小值放入链表

LinkedList<Radar> radar = **new** LinkedList<Radar>();

**for** (i = 0; i < total; i++) {

**double** augment = Math.*sqrt*(Math.*pow*(radius, 2)

- Math.*pow*(coordinate[i][1], 2));

Radar p1 = **new** Radar(coordinate[i][0] - augment, coordinate[i][0]

+ augment);

radar.add(p1);

}

// 将此链表元素按每个雷达能覆盖的右区间，从小到大排序

*SortRadar*(radar);

// 贪心策略为：从左到右建立雷达，贪的是可以单个雷达能覆盖的岛屿个数

// 若第j个区域的左区间小于当前区间右区间的话，即可使用同一个雷达，否则需要等下一次循环，以其右区间来建站

**for** (i = 0; i < total; i++) {

**if** (flag[i] == **false**) {

flag[i] = **true**;

**for** (**int** j = i + 1; j < total; j++) {

**if** (radar.get(j).getMin() < radar.get(i).getMax())

flag[j] = **true**;

}

// 计算雷达个数

ans++;

}

}

**return** ans;

}

// 将链表中的区间，按照右区间大小，按x轴正方向排序

**private** **static** **void** SortRadar(LinkedList<Radar> radar) {

// **TODO** Auto-generated method stub

**for** (**int** i = 0; i < radar.size(); i++) {

**for** (**int** j = i + 1; j < radar.size(); j++) {

**if** (radar.get(j - 1).getMax() > radar.get(j).getMax()) {

Radar radar1 = radar.get(j - 1);

Radar radar2 = radar.get(j);

radar.set(j - 1, radar2);

radar.set(j, radar1);

}

}

}

}

}

**五、评语及评分**

|  |
| --- |
| **评语及评分**  评阅人签名： |