

华为OD机试 - 五子棋迷 (Java & JS & Python)

原创 伏城之外 于 2023-06-04 23:08:25 修改 1770 收藏 11 版权

分类专栏: 华为OD机试AB (Java & JS & Python) 文章标签: 算法 华为机试 Java JavaScript Python

OD 华为OD机试AB (Ja... 同时被 2 个专栏收录 ▾ 该专栏为热销专栏榜 第2名 ¥59.90 3382 订阅 371 篇文章 已订阅

题目描述

张兵和王武是五子棋迷，工作之余经常切磋棋艺。这不，这会儿又下起来了。走了一会儿，轮张兵了，对着一条线思考起来了，这条线上的棋子分布如下：

用数组表示: -1 0 1 1 1 0 1 0 1 -1

棋子分布说明:

- 1代表白子，0代表空位，1代表黑子
- 数组长度L，满足 $1 < L < 40$, L为奇数

你得帮他写一个程序，算出最有利的出子位置。最有利定义：

- 找到一个空位(0)，用棋子(1/-1)填充该位置，可以使得当前子的最大连续长度变大
- 如果存在多个位置，返回最靠近中间的那个坐标
- 如果不存在可行位置，直接返回-1
- 连续长度不能超过5个(五子棋约束)

输入描述



第一行: 当前出子颜色

第二行: 当前的棋局状态

输出描述

1个整数，表示出子位置的数组下标

用例

输入	1 -1 0 1 1 1 0 1 -1 1
输出	5
说明	当前为黑子 (1)，放置在下标为5的位置，黑子的最大连续长度，可以由3到5

输入	-1 -1 0 1 1 1 0 1 0 1 -1 1
输出	1
说明	当前为白子，唯一可以放置的位置下标为1，白子的最大长度，由1变为2

输入	1 0 0 0 0 1 0 0 0 0 1 0
输出	5
说明	可行的位置很多，5最接近中间的位置坐标

题目解析

本题可以使用 双指针 解题。

定义两个指针L, R, 我们假设L,R范围就是要求的连棋范围, 那么L,R范围内必须要包含一个0, 用于落子, 且只能有一个0, 范围内其余棋子必须是下棋者对应的颜色 (第一行输入的颜色)。

另外, 根据题目描述:

连续长度不能超过5个(五字棋约束)

即L,R范围内需要满足三个条件:

- L,R范围内必须要包含一个0, 用于落子, 且只能有一个0
- 范围内其余棋子必须是下棋者对应的颜色
- L,R范围长度不能超过5

上面三个条件约束着双指针的运动, 下面给出三个用例的L,R指针运动示意图:

落子颜色为1

0	1	2	3	4	5	6	7	8	
-1	0	1	1	1	0	1	-1	1	
0	1	2	3	4	5	6	7	8	
-1	0	0	1	1	1	0	1	-1	1
0	1	2	3	4	5	6	7	8	
-1	0	0	1	1	1	0	1	-1	1
0	1	2	3	4	5	6	7	8	
-1	0	0	1	1	1	0	1	-1	1
0	1	2	3	4	5	6	7	8	
-1	0	0	1	1	1	0	1	-1	1
0	1	2	3	4	5	6	7	8	
-1	0	0	1	1	1	0	1	-1	1
0	1	2	3	4	5	6	7	8	
-1	0	0	1	1	1	0	1	-1	1
0	1	2	3	4	5	6	7	8	
-1	0	0	1	1	1	0	1	-1	1
0	1	2	3	4	5	6	7	8	
-1	0	0	1	1	1	0	1	-1	1
0	1	2	3	4	5	6	7	8	
-1	0	0	1	1	1	0	1	-1	1
0	1	2	3	4	5	6	7	8	
-1	0	0	1	1	1	0	1	-1	1
0	1	2	3	4	5	6	7	8	
-1	0	0	1	1	1	0	1	-1	1
0	1	2	3	4	5	6	7	8	
-1	0	0	1	1	1	0	1	-1	1

通过上面例子，我们可以知道，在什么时机进行连棋的统计，需要统计连棋的落子位置和长度

落子颜色-1

0	1	2	3	4	5	6	7	8	9	10
-1	0	1	1	1	0	1	0	1	-1	1
当前位置有棋子，颜色与落子颜色一致，符合连棋要求										
-1	0	1	1	1	0	1	0	1	-1	1
当前位置为空位，可以落子，此时落子数量为1，符合连棋要求										
-1	0	1	1	1	0	1	0	1	-1	1
当前位置有棋子，但是颜色和落子不一致，不符合连棋要求，因此，需要统计L~R-1范围连棋，然后L,R同时移动到当前位置右边										
-1	0	1	1	1	0	1	0	1	-1	1
同上										
-1	0	1	1	1	0	1	0	1	-1	1
同上										
-1	0	1	1	1	0	1	0	1	-1	1
当前位置为空位，因此可以落子，且落子数量为1，符合连棋要求										
-1	0	1	1	1	0	1	0	1	-1	1
当前位置有棋子，但是颜色和落子不一致，因此连棋中断，需要统计L, R-1范围连棋										
-1	0	1	1	1	0	1	0	1	-1	1
当前位置为空位，因此可以落子，且落子数量为1，符合连棋要求										
-1	0	1	1	1	0	1	0	1	-1	1
当前位置有棋子，但是颜色和落子不一致，因此连棋中断，需要统计L, R-1范围连棋										
-1	0	1	1	1	0	1	0	1	-1	1
当前位置有棋子，且与落子颜色一致，符合连棋要求										
-1	0	1	1	1	0	1	0	1	-1	1
当前位置有棋子，但是颜色和落子不一致，因此连棋中断，需要统计L, R-1范围连棋，且L,R需要同时移动到当前位置右侧										
-1	0	1	1	1	0	1	0	1	-1	1
R越界，统计结束										
-1	0	1	1	1	0	1	0	1	-1	1

上面例子中，如果连棋中断，即遇到不同颜色的棋子，则L,R需要同时移动到该不同颜色棋子的右侧

```
1 import java.util.ArrayList;
2 import java.util.Arrays;
3 import java.util.Scanner;
4
5 public class Main {
6     public static void main(String[] args) {
7         Scanner sc = new Scanner(System.in);
8
9         int color = Integer.parseInt(sc.nextLine());
10        int[] nums = Arrays.stream(sc.nextLine().split(
11            "")).mapToInt(Integer::parseInt).toArray();
12
13        System.out.println(getResult(color, nums));
14    }
15
16    public static int getResult(int color, int[] nums) {
17        // 获取初始的最大连续长度
18        int initMaxConstantLen = getInitMaxConstantLen(color, nums);
19
20        ArrayList<int[]> ans = new ArrayList<>();
21
22        // l~r之间必须且只能包含一个0，即必须落子一次，其余都是color颜色的棋子
23        int l = 0;
24        int r = 0;
25
26        // l~r之间包含的0的数量，即落子数量
27        int zero = 0;
28        // l~r之间0的位置，即落子位置
29        int pos = -1;
```



```
68     ++r;
69 }
70 }
71
72 // 收尾操作
73 if (zero == 1 && r - 1 <= 5 && r - 1 > initMaxConstantLen) {
74     ans.add(new int[] {pos, r - 1});
75 }
76
77 // 如果没有统计到连棋情，则返回-1
78 if (ans.size() == 0) return -1;
79
80 int mid = nums.length / 2;
81
82 // 如果统计到连棋
83 // 先按照连棋长度降序，如果长度相同，则按照接近中心位置mid的距离升序（越近的越优），如果距离中心位置mid相同，则按照落子位置升序（越小的越优）
84 ans.sort((a, b) -> a[1] != b[1] ? b[1] - a[1] : cmp(a[0], b[0], mid));
85 return ans.get(0)[0]; // 取最优情况的落子位置
86 }
87
88 // 比较pos1, pos2谁更接近mid，如果距离mid相同，则返回较小的
89 public static int cmp(int pos1, int pos2, int mid) {
90     int dis1 = Math.abs(pos1 - mid);
91     int dis2 = Math.abs(pos2 - mid);
92
93     if (dis1 != dis2) {
94         return dis1 - dis2;
95     } else {
96         return pos1 - pos2;
97     }
98 }
99
100 // 获取初始最大连续长度，即未落子前的最大连续长度
101 public static int getInitMaxConstantLen(int color, int[] nums) {
102     int maxLen = 0;
103
104     int len = 0;
105     for (int num : nums) {
106         if (num != color) {
```

```
107         maxLen = Math.max(maxLen, len);
108         len = 0;
109     } else {
110         len++;
111     }
112 }
113
114 return Math.max(maxLen, len);
115 }
116 }
```