

【华为OD机考 统一考试机试C卷】围棋的气（C++ Java JavaScript Python C语言）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

目前在考C卷，经过两个月的收集整理，**C卷真题已基本整理完毕**

抽到原题的概率为2/3到3/3，**也就是最少抽到两道原题。请注意：大家刷完C卷真题，最好要把B卷的真题刷一下，因为C卷的部分真题来自B卷。**

另外订阅专栏还可以联系笔者开通在线 [OJ](#) 进行刷题，提高刷题效率。

真题目录：华为OD机考机试 真题目录（C卷 + D卷 + B卷 + A卷） + 考点说明

专栏：2023华为OD机试(B卷+C卷+D卷) (C++JavaJSPy)

华为OD面试真题精选：华为OD面试真题精选

在线OJ：点击立即刷题，模拟真实机考环境

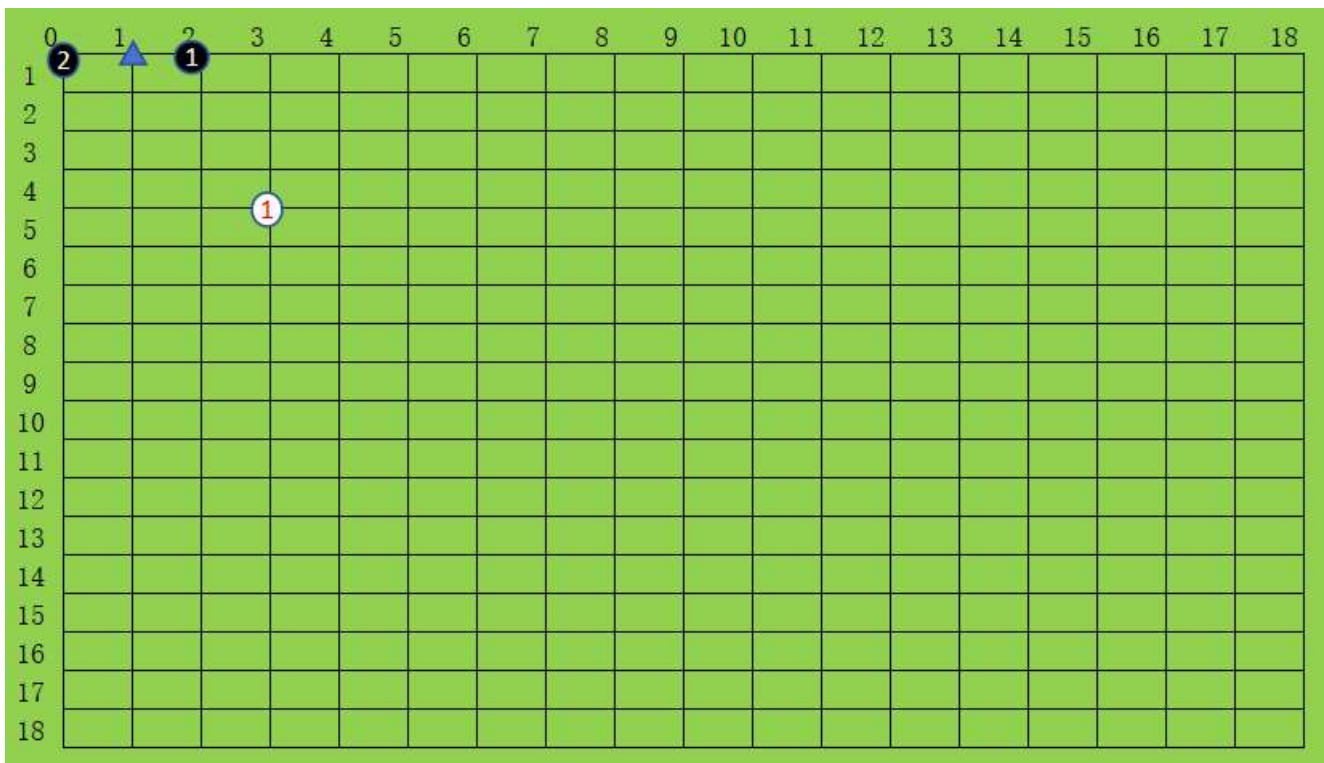
题目描述

围棋棋盘由纵横各19条线垂直相交组成，棋盘上一共 $19 \times 19 = 361$ 个交点，对弈双方一方执白棋，一方执黑棋，落子时只能将棋子置于交点上。

“气”是围棋中很重要的一个概念，某个棋子有几口气，是指其上下左右方向四个相邻的交叉点中，有几个交叉点没有棋子，由此可知：

1. 在棋盘的边缘上的棋子最多有 3 口气（黑1），在棋盘角点的棋子最多有2口气（黑2），其他情况最多有4口气（白1）
2. 所有同色棋子的气之和叫做该色棋子的气，需要注意的是，同色棋子重合的气点，对于该颜色棋子来说，只能计算一次气，比如下图中，黑棋一共4口气，而不是5口气，因为黑1和黑2中间红色三角标出来的气是两个黑棋共有的，对于黑棋整体来说只能算一个气。
3. 本题目只计算气，对于眼也按气计算，如果您不清楚“眼”的概念，可忽略，按照前面描述的规则计算即可。

现在，请根据输入的黑棋和白棋得到坐标位置，计算黑棋和白棋一共各有多少气？



输入描述

输入包含两行数据，
每行数据以空格分隔，数据个数是2的整数倍，每两个数是一组，代表棋子在棋盘上的坐标；
坐标的原点在棋盘左上角点，第一个值是行号，范围从0到18；第二个值是列号，范围从0到18。

举例说明：如：

```
0 5 8 9 9 10
5 0 9 9 9 8
```

第一行数据表示三个坐标 (0, 5) 、(8, 9)、(9, 10)
第一行表示黑棋的坐标，第二行表示白棋的坐标。
题目保证输入两行数据，无空行且每行按前文要求是偶数个，每个坐标不会超出棋盘范围。

输出描述

两个数字以空格分隔，第一个数代表黑棋的气数，第二个数代表白棋的气数。

8 7

用例

输入

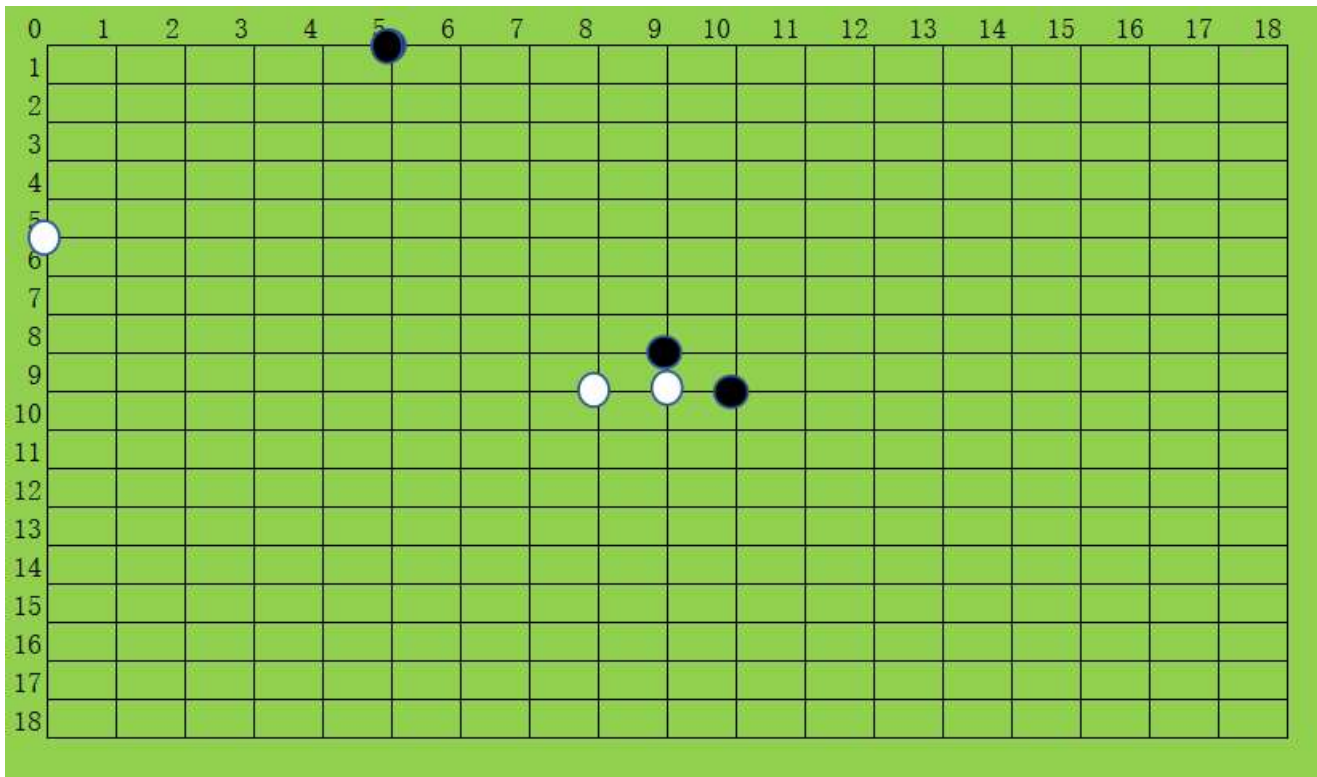
1	0 5 8 9 9 10
2	5 0 9 9 9 8

输出

1	8 7
---	-----

说明

数数黑棋一共8口气，数数白棋一共7口气。



解法1

解题思路

在这个例子中，我们有两组棋子的坐标。第一组是黑棋的坐标，第二组是白棋的坐标。

首先，得到两个整数数组，分别代表黑棋和白棋的位置。

黑棋的坐标数组为：{0_5, 8_9, 9_10}

白棋的坐标数组为：{5_0, 9_9, 9_8}

然后，我们计算每组棋子的“气”。

对于黑棋，我们检查每个棋子周围的四个位置，得到的“气”的坐标集合为：{“0_4”, “0_6”, “1_5”, “7_9”, “8_8”, “8_10”, “9_9”, “9_11”, “10_10”}。然后，我们从这个集合中减去黑棋的位置和白棋占据的位置，得到最终的“气”的数量为8。

对于白棋，我们同样检查每个棋子周围的四个位置，得到的"气"的坐标集合为：{"4_0", "5_1", "6_0", "8_9", "9_8", "9_10", "10_9", "10_8"}。然后，我们从这个集合中减去白棋的位置和黑棋占据的位置，得到最终的"气"的数量为7。

所以，黑棋的"气"的数量为8，白棋的"气"的数量为7。

C++

```
1  #include <iostream>
2  #include <sstream>
3  #include <vector>
4  #include <unordered_set>
5
6  using namespace std;
7
8  // 最大的棋盘边界索引
9  const int maxSide = 18;
10
11 // 解析输入的坐标字符串, 返回整数数组
12 vector<int> parseCoordinates(string input) {
13     stringstream ss(input);
14     int num;
15     vector<int> coordinates;
16     while (ss >> num) {
17         coordinates.push_back(num);
18     }
19     return coordinates;
20 }
21
22 // 计算棋子的气数
23 int counting(vector<int>& alias, vector<int>& enemy) {
24     // 创建一个新的unordered_set, 用于存储棋子的气
25     unordered_set<string> count;
26     // 遍历己方棋子的坐标
27     for (int i = 0; i < alias.size(); i += 2) {
28         int x = alias[i];
29         int y = alias[i + 1];
30         string pos = to_string(x) + "_" + to_string(y);
31         // 将己方棋子的位置添加到集合中
32         count.insert(pos);
33         // 分别检查上下左右四个方向, 如果存在气则添加到集合中
34     }
```

```

34     if (x > 0) {
35         count.insert(to_string(x - 1) + "_" + to_string(y));
36     }
37     if (x < maxSide) {
38         count.insert(to_string(x + 1) + "_" + to_string(y));
39     }
40     if (y > 0) {
41         count.insert(to_string(x) + "_" + to_string(y - 1));
42     }
43     if (y < maxSide) {
44         count.insert(to_string(x) + "_" + to_string(y + 1));
45     }
46 }
47 // 计算得到的气数减去己方棋子的数量
48 int res = count.size();
49 // 减去敌方棋子占据的气点
50 for (int i = 0; i < enemy.size(); i += 2) {
51     string pos = to_string(enemy[i]) + "_" + to_string(enemy[i + 1]);
52     if (count.find(pos) != count.end()) {
53         res--;
54     }
55 }
56 return res - alias.size() / 2;
57 }
58
59 int main() {
60     string line;
61     // 读取黑棋的坐标
62     getline(cin, line);
63     vector<int> locBlacks = parseCoordinates(line);
64     // 读取白棋的坐标
65     getline(cin, line);
66     vector<int> locWhites = parseCoordinates(line);
67     // 输出黑棋和白棋的气数, 两个数字以空格分隔
68     cout << counting(locBlacks, locWhites) << " " << counting(locWhites, locBlacks) << endl;
69     return 0;
70 }

```

```

1  import java.util.Scanner;
2  import java.util.Set;
3  import java.util.HashSet;
4
5  public class Main {
6
7      // 最大的棋盘边界索引
8      static int maxSide = 18;
9
10     public static void main(String[] args) {
11         Scanner in = new Scanner(System.in);
12         // 读取黑棋的坐标
13         int[] locBlacks = parseCoordinates(in.nextLine());
14         // 读取白棋的坐标
15         int[] locWhites = parseCoordinates(in.nextLine());
16         // 输出黑棋和白棋的气数, 两个数字以空格分隔
17         System.out.println(counting(locBlacks, locWhites) + " " + counting(locWhites, locBlacks));
18     }
19
20     // 计算棋子的气数
21     static int counting(int[] alias, int[] enemy) {
22         // 创建一个新的HashSet, 用于存储棋子的气
23         Set<String> count = new HashSet<>();
24         // 遍历己方棋子的坐标
25         for (int i = 0; i < alias.length; i += 2) {
26             int x = alias[i];
27             int y = alias[i + 1];
28             String pos = x + "_" + y;
29             // 将己方棋子的位置添加到集合中
30             count.add(pos);
31             // 分别检查上下左右四个方向, 如果存在气则添加到集合中
32             if (x > 0) {
33                 count.add((x - 1) + "_" + y);
34             }
35             if (x < maxSide) {
36                 count.add((x + 1) + "_" + y);
37             }
38             if (y > 0) {
39                 count.add(x + "_" + (y - 1));
40             }
41         }

```

```

41         if (y < maxSide) {
42             count.add(x + "_" + (y + 1));
43         }
44     }
45     // 计算得到的气数减去己方棋子的数量
46     int res = count.size();
47     // 减去敌方棋子占据的气点
48     for (int i = 0; i < enemy.length; i += 2) {
49         String pos = enemy[i] + "_" + enemy[i + 1];
50         if (count.contains(pos)) {
51             res--;
52         }
53     }
54     return res - alias.length / 2;
55 }
56
57 // 解析输入的坐标字符串, 返回整数数组
58 static int[] parseCoordinates(String input) {
59     String[] tokens = input.split(" ");
60     int[] coordinates = new int[tokens.length];
61     for (int i = 0; i < tokens.length; i++) {
62         coordinates[i] = Integer.parseInt(tokens[i]);
63     }
64     return coordinates;
65 }
66 }

```

javaScript

```

1  const readline = require('readline');
2
3  // 创建readLine接口
4  const rl = readline.createInterface({
5      input: process.stdin,
6      output: process.stdout
7  });
8
9  // 最大的棋盘边界索引
10 const maxSide = 18;
11
12

```



```

14 // 解析输入的坐标字符串, 返回整数数组
15 function parseCoordinates(input) {
16     return input.split(' ').map(Number);
17 }
18 // 计算棋子的气数
19 function counting(alias, enemy) {
20     // 创建一个新的Set, 用于存储棋子的气
21     let count = new Set();
22     // 遍历己方棋子的坐标
23     for (let i = 0; i < alias.length; i += 2) {
24         let x = alias[i];
25         let y = alias[i + 1];
26         let pos = x + "_" + y;
27         // 将己方棋子的位置添加到集合中
28         count.add(pos);
29         // 分别检查上下左右四个方向, 如果存在气则添加到集合中
30         if (x > 0) {
31             count.add((x - 1) + "_" + y);
32         }
33         if (x < maxSide) {
34             count.add((x + 1) + "_" + y);
35         }
36         if (y > 0) {
37             count.add(x + "_" + (y - 1));
38         }
39         if (y < maxSide) {
40             count.add(x + "_" + (y + 1));
41         }
42     }
43     // 计算得到的气数减去己方棋子的数量
44     let res = count.size;
45     // 减去敌方棋子占据的气点
46     for (let i = 0; i < enemy.length; i += 2) {
47         let pos = enemy[i] + "_" + enemy[i + 1];
48         if (count.has(pos)) {
49             res--;
50         }
51     }
52     return res - alias.length / 2;
53 }

```

```

53 }
54
55 // 读取黑棋的坐标
56 rl.on('line', (blackInput) => {
57     let locBlacks = parseCoordinates(blackInput);
58     // 读取白棋的坐标
59     rl.on('line', (whiteInput) => {
60         let locWhites = parseCoordinates(whiteInput);
61         // 输出黑棋和白棋的气数, 两个数字以空格分隔
62         console.log(counting(locBlacks, locWhites) + " " + counting(locWhites, locBlacks));
63         rl.close();
64     });
65 });

```

Python

```

1 # 最大的棋盘边界索引
2 maxSide = 18
3
4 # 解析输入的坐标字符串, 返回整数数组
5 def parseCoordinates(input):
6     return list(map(int, input.split()))
7
8 # 计算棋子的气数
9 def counting(alias, enemy):
10     # 创建一个新的set, 用于存储棋子的气
11     count = set()
12     # 遍历己方棋子的坐标
13     for i in range(0, len(alias), 2):
14         x = alias[i]
15         y = alias[i + 1]
16         pos = str(x) + "_" + str(y)
17         # 将己方棋子的位置添加到集合中
18         count.add(pos)
19         # 分别检查上下左右四个方向, 如果存在气则添加到集合中
20         if x > 0:
21             count.add(str(x - 1) + "_" + str(y))
22         if x < maxSide:
23             count.add(str(x + 1) + "_" + str(y))
24         if y > 0:
25

```

```

25         count.add(str(x) + "_" + str(y - 1))
26     if y < maxSide:
27         count.add(str(x) + "_" + str(y + 1))
28 # 计算得到的气数减去己方棋子的数量
29 res = len(count)
30 # 减去敌方棋子占据的气点
31 for i in range(0, len(enemy), 2):
32     pos = str(enemy[i]) + "_" + str(enemy[i + 1])
33     if pos in count:
34         res -= 1
35 return res - len(alias) // 2
36
37 # 读取黑棋的坐标
38 locBlacks = parseCoordinates(input())
39 # 读取白棋的坐标
40 locWhites = parseCoordinates(input())
41 # 输出黑棋和白棋的气数，两个数字以空格分隔
42 print(counting(locBlacks, locWhites), counting(locWhites, locBlacks))
43

```

C语言

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4
5  // 定义棋盘大小
6  #define MAX_SIDE 18
7  #define MAX_STONES 361 // 最多棋子数 (19x19)
8
9  // 解析输入的坐标字符串，返回坐标数组
10 void parseCoordinates(char* input, int* coordinates) {
11     char* token = strtok(input, " ");
12     int index = 0;
13     while (token != NULL) {
14         coordinates[index++] = atoi(token);
15         token = strtok(NULL, " ");
16     }
17 }
18
19

```

```

17 // 检查点是否在数组中
20 int contains(int x, int y, int* stones) {
21     for (int i = 0; stones[i] != -1 && i < MAX_STONES * 2; i += 2) {
22         if (stones[i] == x && stones[i + 1] == y) {
23             return 1;
24         }
25     }
26     return 0;
27 }
28
29 // 计算棋子的气数
30 int counting(int* alias, int* enemy) {
31     int count = 0;
32     for (int i = 0; alias[i] != -1 && i < MAX_STONES * 2; i += 2) {
33         int x = alias[i];
34         int y = alias[i + 1];
35         int directions[4][2] = {{x-1, y}, {x+1, y}, {x, y-1}, {x, y+1}};
36
37         for (int j = 0; j < 4; j++) {
38             int dx = directions[j][0];
39             int dy = directions[j][1];
40             // 检查是否在棋盘内且不被任何棋子占据
41             if (dx >= 0 && dx <= MAX_SIDE && dy >= 0 && dy <= MAX_SIDE &&
42                 !contains(dx, dy, alias) && !contains(dx, dy, enemy)) {
43                 count++;
44             }
45         }
46     }
47     return count;
48 }
49
50 int main() {
51     char line[1024];
52     int locBlacks[MAX_STONES * 2 + 1], locWhites[MAX_STONES * 2 + 1];
53
54     // 初始化数组
55     memset(locBlacks, -1, sizeof(locBlacks));
56     memset(locWhites, -1, sizeof(locWhites));
57
58     // 读取黑棋的坐标
59
60

```

```

60     fgets(line, sizeof(line), stdin);
61     parseCoordinates(line, locBlacks);
62     // 读取白棋的坐标
63     fgets(line, sizeof(line), stdin);
64     parseCoordinates(line, locWhites);
65
66     // 输出黑棋和白棋的气数, 两个数字以空格分隔
67     printf("%d %d\n", counting(locBlacks, locWhites), counting(locWhites, locBlacks));
68     return 0;
}

```

解法2

一种骚操作, 可忽略

C++

```

1  #include <iostream>
2  #include <sstream>
3  #include <set>
4  #include <vector>
5  using namespace std;
6  // 最大的棋盘边界索引
7  static int maxSide = 18;
8
9  // 读取棋子的坐标集合
10 set<int> readChessSet(string line) {
11     istringstream iss(line);
12     set<int> chessSet; // 创建一个set来存储坐标
13     int x, y;
14     while (iss >> x >> y) { // 解析行坐标和列坐标
15         // 将坐标转换为一个唯一的整数表示, 并添加到集合中
16         chessSet.insert(x * 19 + y);
17     }
18     return chessSet; // 返回包含所有棋子坐标的集合
19 }
20
21 // 计算棋子的气数
22 int counting(set<int> alias, set<int> enemy) {
23     // 创建一个新的set, 用于存储棋子的气
24

```

```

24     set<int> count(alias); // 初始包含所有己方棋子的位置
25     for (int pos : alias) {
26         int x = pos / 19; // 计算行坐标
27         int y = pos % 19; // 计算列坐标
28         // 分别检查上下左右四个方向, 如果存在气则添加到集合中
29         if (x > 0) count.insert((x - 1) * 19 + y);
30         if (x < maxSide) count.insert((x + 1) * 19 + y);
31         if (y > 0) count.insert(x * 19 + (y - 1));
32         if (y < maxSide) count.insert(x * 19 + (y + 1));
33     }
34     // 计算得到的气数减去己方棋子的数量
35     int res = count.size() - alias.size();
36     // 减去敌方棋子占据的气点
37     for (int pos : enemy) {
38         if (count.find(pos) != count.end()) {
39             res--;
40         }
41     }
42     return res; // 返回最终的气数
43 }
44
45
46 int main() {
47     string blackLine, whiteLine;
48     getline(cin, blackLine); // 读取黑棋的坐标集合
49     getline(cin, whiteLine); // 读取白棋的坐标集合
50     set<int> blackSet = readChessSet(blackLine);
51     set<int> whiteSet = readChessSet(whiteLine);
52     // 输出黑棋和白棋的气数, 两个数字以空格分隔
53     cout << counting(blackSet, whiteSet) << " " << counting(whiteSet, blackSet) << endl;
54     return 0;
55 }

```

Java

```

1 import java.util.Scanner;
2 import java.util.Set;
3 import java.util.HashSet;

```

```

4

```

```

5

```

```

6

```

```

0 public class Main {
1
2 // 最大的棋盘边界索引
3 static int maxSide = 18;
4
5
6
7
8 public static void main(String[] args) {
9     Scanner in = new Scanner(System.in);
10    // 读取黑棋的坐标集合
11    Set<Integer> blackSet = readChessSet(in.nextLine());
12    // 读取白棋的坐标集合
13    Set<Integer> whiteSet = readChessSet(in.nextLine());
14    // 输出黑棋和白棋的气数, 两个数字以空格分隔
15    System.out.println(counting(blackSet, whiteSet) + " " + counting(whiteSet, blackSet));
16 }
17
18 // 读取棋子的坐标集合
19 static Set<Integer> readChessSet(String line) {
20     String[] parts = line.split(" "); // 将输入的行按空格分割
21     Set<Integer> chessSet = new HashSet<>(); // 创建一个HashSet来存储坐标
22     for (int i = 0; i < parts.length; i += 2) {
23         int x = Integer.parseInt(parts[i]); // 解析行坐标
24         int y = Integer.parseInt(parts[i + 1]); // 解析列坐标
25         // 将坐标转换为一个唯一的整数表示, 并添加到集合中
26         chessSet.add(x * 19 + y);
27     }
28     return chessSet; // 返回包含所有棋子坐标的集合
29 }
30
31 // 计算棋子的气数
32 static int counting(Set<Integer> alias, Set<Integer> enemy) {
33     // 创建一个新的HashSet, 用于存储棋子的气
34     Set<Integer> count = new HashSet<>(alias); // 初始包含所有己方棋子的位置
35     for (int pos : alias) {
36         int x = pos / 19; // 计算行坐标
37         int y = pos % 19; // 计算列坐标
38         // 分别检查上下左右四个方向, 如果存在气则添加到集合中
39         if (x > 0) count.add((x - 1) * 19 + y);
40         if (x < maxSide) count.add((x + 1) * 19 + y);
41         if (y > 0) count.add(x * 19 + (y - 1));
42     }
43 }
44
45
46

```

```

47         if (y < maxSide) count.add(x * 19 + (y + 1));
48     }
49     // 计算得到的气数减去己方棋子的数量
50     int res = count.size() - alias.size();
51     // 减去敌方棋子占据的气点
52     for (int pos : enemy) {
53         if (count.contains(pos)) {
54             res--;
55         }
56     }
57     return res; // 返回最终的气数
58 }
}

```

JavaScript

```

1 // 最大的棋盘边界索引
2 const maxSide = 18;
3
4 // 读取棋子的坐标集合
5 function readChessSet(line) {
6     const parts = line.split(' '); // 将输入的行按空格分割
7     const chessSet = new Set(); // 创建一个Set来存储坐标
8     for (let i = 0; i < parts.length; i += 2) {
9         const x = parseInt(parts[i]); // 解析行坐标
10        const y = parseInt(parts[i + 1]); // 解析列坐标
11        // 将坐标转换为一个唯一的整数表示, 并添加到集合中
12        chessSet.add(x * 19 + y);
13    }
14    return chessSet; // 返回包含所有棋子坐标的集合
15 }
16
17 // 计算棋子的气数
18 function counting(alias, enemy) {
19     // 创建一个新的Set, 用于存储棋子的气
20     const count = new Set(alias); // 初始包含所有己方棋子的位置
21     for (let pos of alias) {
22         const x = Math.floor(pos / 19); // 计算行坐标
23         const y = pos % 19; // 计算列坐标
24         // 分别检查上下左右四个方向, 如果存在气则添加到集合中
25     }
26 }

```



```

25     if (x > 0) count.add((x - 1) * 19 + y);
26     if (x < maxSide) count.add((x + 1) * 19 + y);
27     if (y > 0) count.add(x * 19 + (y - 1));
28     if (y < maxSide) count.add(x * 19 + (y + 1));
29 }
30 // 计算得到的气数减去己方棋子的数量
31 let res = count.size - alias.size;
32 // 减去敌方棋子占据的气点
33 for (let pos of enemy) {
34     if (count.has(pos)) {
35         res--;
36     }
37 }
38 return res; // 返回最终的气数
39 }
40
41 const readline = require('readline').createInterface({
42     input: process.stdin,
43     output: process.stdout
44 });
45
46 readline.on('line', blackLine => {
47     readline.on('line', whiteLine => {
48         const blackSet = readChessSet(blackLine); // 读取黑棋的坐标集合
49         const whiteSet = readChessSet(whiteLine); // 读取白棋的坐标集合
50         // 输出黑棋和白棋的气数, 两个数字以空格分隔
51         console.log(counting(blackSet, whiteSet) + " " + counting(whiteSet, blackSet));
52         readline.close();
53     });
54 });

```

Python

```

1
2 maxSide = 18
3
4
5 def readChessSet(line):
6     parts = line.split(' ')
7     chessSet = set()
8

```

```

8     for i in range(0, len(parts), 2):
9         x = int(parts[i])
10        y = int(parts[i + 1])
11        chessSet.add(x * 19 + y)
12    return chessSet
13
14
15    def counting(alias, enemy):
16        count = set(alias)
17        for pos in alias:
18            x = pos // 19
19            y = pos % 19
20            if x > 0: count.add((x - 1) * 19 + y)
21            if x < maxSide: count.add((x + 1) * 19 + y)
22            if y > 0: count.add(x * 19 + (y - 1))
23            if y < maxSide: count.add(x * 19 + (y + 1))
24
25        res = len(count) - len(alias)
26
27        for pos in enemy:
28            if pos in count:
29                res -= 1
30        return res
31
32    # 主方法, 程序入口
33    blackLine = input()
34    whiteLine = input()
35    blackSet = readChessSet(blackLine)
36    whiteSet = readChessSet(whiteLine)
37    print(counting(blackSet, whiteSet), counting(whiteSet, blackSet))

```

文章目录

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

[题目描述](#)

[输入描述](#)

[输出描述](#)

[用例](#)

解法1

解题思路

C++

Java

JavaScript

Python

C语言

解法2

C++

Java

JavaScript

Python

机考真题 华为OD



CSDN @算法大师