【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

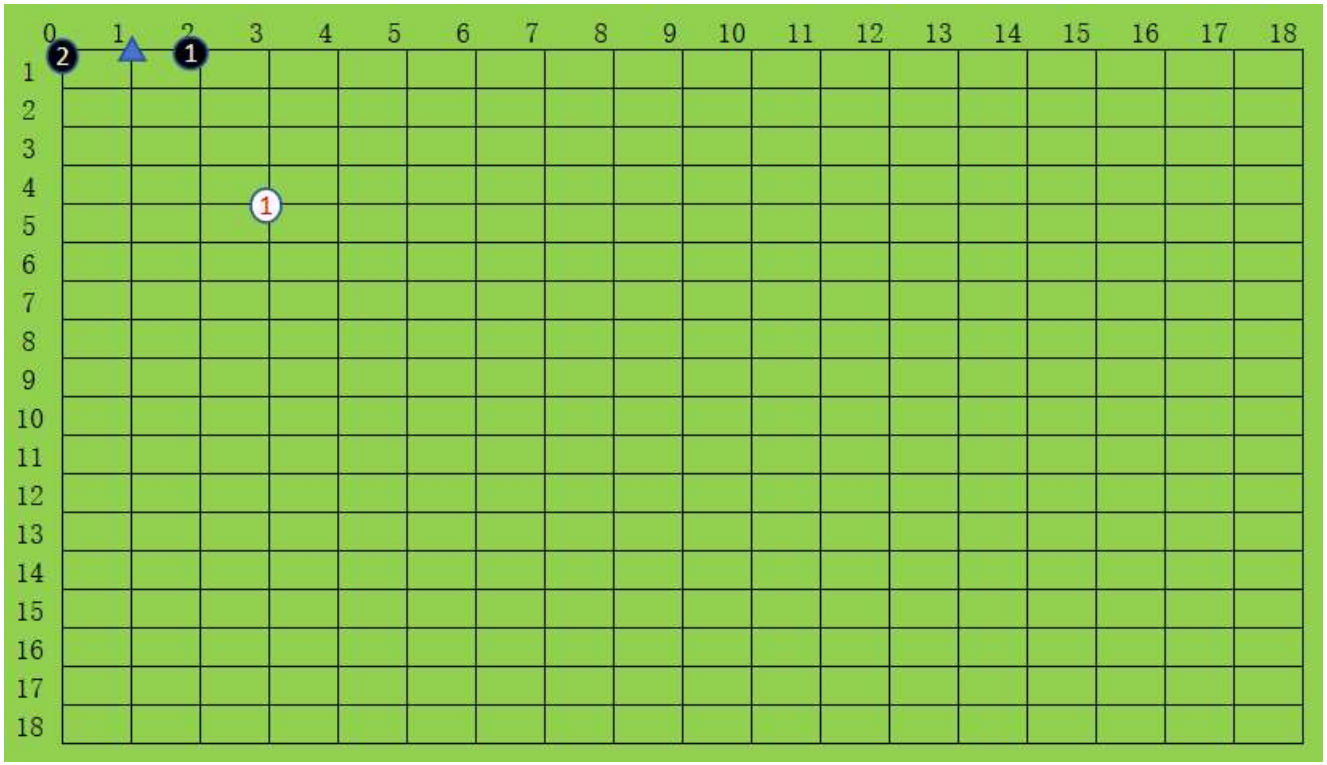
题目描述

围棋棋盘由纵横各19条线垂直相交组成，棋盘上一共19 x 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

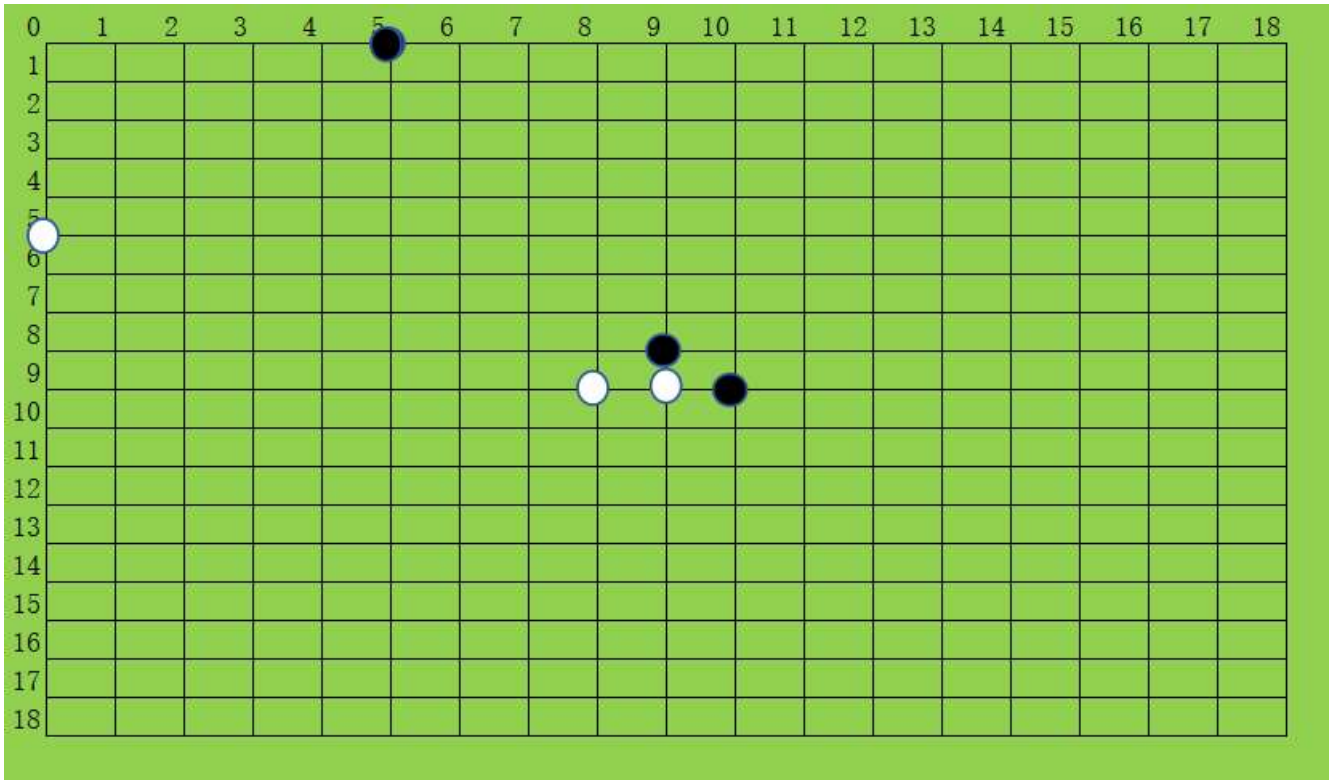
```
0 5 8 9 9 10
5 0 9 9 9 8
```

输出

```
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++

```cpp
#include <iostream>
#include <sstream>
#include <vector>
#include <unordered_set>

using namespace std;

// 最大的棋盘边界索引
const int maxSide = 18;

// 解析输入的坐标字符串，返回整数数组
vector<int> parseCoordinates(string input) {
    stringstream ss(input);
    int num;
    vector<int> coordinates;
    while (ss >> num) {
        coordinates.push_back(num);
    }
    return coordinates;
}

// 计算棋子的气数
int counting(vector<int>& alias, vector<int>& enemy) {
    // 创建一个新的unordered_set，用于存储棋子的气
    unordered_set<string> count;
    // 遍历己方棋子的坐标
    for (int i = 0; i < alias.size(); i += 2) {
        int x = alias[i];
        int y = alias[i + 1];
        string pos = to_string(x) + "_" + to_string(y);
        // 将己方棋子的位置添加到集合中
        count.insert(pos);
        // 分别检查上下左右四个方向，如果存在气则添加到集合中
```

```
        if (x > 0) {
            count.insert(to_string(x - 1) + "_" + to_string(y));
        }
        if (x < maxSide) {
            count.insert(to_string(x + 1) + "_" + to_string(y));
        }
        if (y > 0) {
            count.insert(to_string(x) + "_" + to_string(y - 1));
        }
        if (y < maxSide) {
            count.insert(to_string(x) + "_" + to_string(y + 1));
        }
    }
    // 计算得到的气数减去己方棋子的数量
    int res = count.size();
    // 减去敌方棋子占据的气点
    for (int i = 0; i < enemy.size(); i += 2) {
        string pos = to_string(enemy[i]) + "_" + to_string(enemy[i + 1]);
        if (count.find(pos) != count.end()) {
            res--;
        }
    }
    return res - alias.size() / 2;
}

int main() {
    string line;
    // 读取黑棋的坐标
    getline(cin, line);
    vector<int> locBlacks = parseCoordinates(line);
    // 读取白棋的坐标
    getline(cin, line);
    vector<int> locWhites = parseCoordinates(line);
    // 输出黑棋和白棋的气数，两个数字以空格分隔
    cout << counting(locBlacks, locWhites) << " " << counting(locWhites, locBlacks) << endl;
    return 0;
}
```

Java

```java
import java.util.Scanner;
import java.util.Set;
import java.util.HashSet;

public class Main {

    // 最大的棋盘边界索引
    static int maxSide = 18;

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        // 读取黑棋的坐标
        int[] locBlacks = parseCoordinates(in.nextLine());
        // 读取白棋的坐标
        int[] locWhites = parseCoordinates(in.nextLine());
        // 输出黑棋和白棋的气数，两个数字以空格分隔
        System.out.println(counting(locBlacks, locWhites) + " " + counting(locWhites, locBlacks));
    }

    // 计算棋子的气数
    static int counting(int[] alias, int[] enemy) {
        // 创建一个新的HashSet，用于存储棋子的气
        Set<String> count = new HashSet<>();
        // 遍历己方棋子的坐标
        for (int i = 0; i < alias.length; i += 2) {
            int x = alias[i];
            int y = alias[i + 1];
            String pos = x + "_" + y;
            // 将己方棋子的位置添加到集合中
            count.add(pos);
            // 分别检查上下左右四个方向，如果存在气则添加到集合中
            if (x > 0) {
                count.add((x - 1) + "_" + y);
            }
            if (x < maxSide) {
                count.add((x + 1) + "_" + y);
            }
            if (y > 0) {
                count.add(x + "_" + (y - 1));
            }

```

```
            if (y < maxSide) {
                count.add(x + "_" + (y + 1));
            }
        }
        // 计算得到的气数减去己方棋子的数量
        int res = count.size();
        // 减去敌方棋子占据的气点
        for (int i = 0; i < enemy.length; i += 2) {
            String pos = enemy[i] + "_" + enemy[i + 1];
            if (count.contains(pos)) {
                res--;
            }
        }
        return res - alias.length / 2;
    }

    // 解析输入的坐标字符串，返回整数数组
    static int[] parseCoordinates(String input) {
        String[] tokens = input.split(" ");
        int[] coordinates = new int[tokens.length];
        for (int i = 0; i < tokens.length; i++) {
            coordinates[i] = Integer.parseInt(tokens[i]);
        }
        return coordinates;
    }
}
```

javaScript

```
const readline = require('readline');

// 创建readline接口
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

// 最大的棋盘边界索引
const maxSide = 18;

```

```
// 解析输入的坐标字符串，返回整数数组
function parseCoordinates(input) {
    return input.split(' ').map(Number);
}

// 计算棋子的气数
function counting(alias, enemy) {
    // 创建一个新的Set，用于存储棋子的气
    let count = new Set();
    // 遍历己方棋子的坐标
    for (let i = 0; i < alias.length; i += 2) {
        let x = alias[i];
        let y = alias[i + 1];
        let pos = x + "_" + y;
        // 将己方棋子的位置添加到集合中
        count.add(pos);
        // 分别检查上下左右四个方向，如果存在气则添加到集合中
        if (x > 0) {
            count.add((x - 1) + "_" + y);
        }
        if (x < maxSide) {
            count.add((x + 1) + "_" + y);
        }
        if (y > 0) {
            count.add(x + "_" + (y - 1));
        }
        if (y < maxSide) {
            count.add(x + "_" + (y + 1));
        }
    }
    // 计算得到的气数减去己方棋子的数量
    let res = count.size;
    // 减去敌方棋子占据的气点
    for (let i = 0; i < enemy.length; i += 2) {
        let pos = enemy[i] + "_" + enemy[i + 1];
        if (count.has(pos)) {
            res--;
        }
    }
    return res - alias.length / 2;

```

```
}

// 读取黑棋的坐标
rl.on('line', (blackInput) => {
    let locBlacks = parseCoordinates(blackInput);
    // 读取白棋的坐标
    rl.on('line', (whiteInput) => {
        let locWhites = parseCoordinates(whiteInput);
        // 输出黑棋和白棋的气数，两个数字以空格分隔
        console.log(counting(locBlacks, locWhites) + " " + counting(locWhites, locBlacks));
        rl.close();
    });
});
```

Python

```
# 最大的棋盘边界索引
maxSide = 18

# 解析输入的坐标字符串，返回整数数组
def parseCoordinates(input):
    return list(map(int, input.split()))

# 计算棋子的气数
def counting(alias, enemy):
    # 创建一个新的set，用于存储棋子的气
    count = set()
    # 遍历己方棋子的坐标
    for i in range(0, len(alias), 2):
        x = alias[i]
        y = alias[i + 1]
        pos = str(x) + "_" + str(y)
        # 将己方棋子的位置添加到集合中
        count.add(pos)
        # 分别检查上下左右四个方向，如果存在气则添加到集合中
        if x > 0:
            count.add(str(x - 1) + "_" + str(y))
        if x < maxSide:
            count.add(str(x + 1) + "_" + str(y))
        if y > 0:
```

```
            count.add(str(x) + "_" + str(y - 1))
        if y < maxSide:
            count.add(str(x) + "_" + str(y + 1))
    # 计算得到的气数减去己方棋子的数量
    res = len(count)
    # 减去敌方棋子占据的气点
    for i in range(0, len(enemy), 2):
        pos = str(enemy[i]) + "_" + str(enemy[i + 1])
        if pos in count:
            res -= 1
    return res - len(alias) // 2

# 读取黑棋的坐标
locBlacks = parseCoordinates(input())
# 读取白棋的坐标
locWhites = parseCoordinates(input())
# 输出黑棋和白棋的气数，两个数字以空格分隔
print(counting(locBlacks, locWhites), counting(locWhites, locBlacks))
```

C语言

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

// 定义棋盘大小
#define MAX_SIDE 18
#define MAX_STONES 361 // 最多棋子数（19x19）

// 解析输入的坐标字符串，返回坐标数组
void parseCoordinates(char* input, int* coordinates) {
    char* token = strtok(input, " ");
    int index = 0;
    while (token != NULL) {
        coordinates[index++] = atoi(token);
        token = strtok(NULL, " ");
    }
}

```

```c
// 检查点是否在数组中
int contains(int x, int y, int* stones) {
    for (int i = 0; stones[i] != -1 && i < MAX_STONES * 2; i += 2) {
        if (stones[i] == x && stones[i + 1] == y) {
            return 1;
        }
    }
    return 0;
}

// 计算棋子的气数
int counting(int* alias, int* enemy) {
    int count = 0;
    for (int i = 0; alias[i] != -1 && i < MAX_STONES * 2; i += 2) {
        int x = alias[i];
        int y = alias[i + 1];
        int directions[4][2] = {{x-1, y}, {x+1, y}, {x, y-1}, {x, y+1}};

        for (int j = 0; j < 4; j++) {
            int dx = directions[j][0];
            int dy = directions[j][1];
            // 检查是否在棋盘内且不被任何棋子占据
            if (dx >= 0 && dx <= MAX_SIDE && dy >= 0 && dy <= MAX_SIDE &&
                !contains(dx, dy, alias) && !contains(dx, dy, enemy)) {
                count++;
            }
        }
    }
    return count;
}

int main() {
    char line[1024];
    int locBlacks[MAX_STONES * 2 + 1], locWhites[MAX_STONES * 2 + 1];

    // 初始化数组
    memset(locBlacks, -1, sizeof(locBlacks));
    memset(locWhites, -1, sizeof(locWhites));

    // 读取黑棋的坐标
```

```
    fgets(line, sizeof(line), stdin);
    parseCoordinates(line, locBlacks);
    // 读取白棋的坐标
    fgets(line, sizeof(line), stdin);
    parseCoordinates(line, locWhites);

    // 输出黑棋和白棋的气数，两个数字以空格分隔
    printf("%d %d\n", counting(locBlacks, locWhites), counting(locWhites, locBlacks));
    return 0;
}
```

解法2

一种骚操作，可忽略

C++

```
#include <iostream>
#include <sstream>
#include <set>
#include <vector>
using namespace std;
// 最大的棋盘边界索引
static int maxSide = 18;

// 读取棋子的坐标集合
set<int> readChessSet(string line) {
    istringstream iss(line);
    set<int> chessSet; // 创建一个set来存储坐标
    int x, y;
    while (iss >> x >> y) { // 解析行坐标和列坐标
        // 将坐标转换为一个唯一的整数表示，并添加到集合中
        chessSet.insert(x * 19 + y);
    }
    return chessSet; // 返回包含所有棋子坐标的集合
}

// 计算棋子的气数
int counting(set<int> alias, set<int> enemy) {
    // 创建一个新的set，用于存储棋子的气
```

```
    set<int> count(alias); // 初始包含所有己方棋子的位置
    for (int pos : alias) {
        int x = pos / 19; // 计算行坐标
        int y = pos % 19; // 计算列坐标
        // 分别检查上下左右四个方向，如果存在气则添加到集合中
        if (x > 0) count.insert((x - 1) * 19 + y);
        if (x < maxSide) count.insert((x + 1) * 19 + y);
        if (y > 0) count.insert(x * 19 + (y - 1));
        if (y < maxSide) count.insert(x * 19 + (y + 1));
    }
    // 计算得到的气数减去己方棋子的数量
    int res = count.size() - alias.size();
    // 减去敌方棋子占据的气点
    for (int pos : enemy) {
        if (count.find(pos) != count.end()) {
            res--;
        }
    }
    return res; // 返回最终的气数
}


int main() {
    string blackLine, whiteLine;
    getline(cin, blackLine); // 读取黑棋的坐标集合
    getline(cin, whiteLine); // 读取白棋的坐标集合
    set<int> blackSet = readChessSet(blackLine);
    set<int> whiteSet = readChessSet(whiteLine);
    // 输出黑棋和白棋的气数，两个数字以空格分隔
    cout << counting(blackSet, whiteSet) << " " << counting(whiteSet, blackSet) << endl;
    return 0;
}
```

Java

```
import java.util.Scanner;
import java.util.Set;
import java.util.HashSet;


```

```java
public class Main {

    // 最大的棋盘边界索引
    static int maxSide = 18;


    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        // 读取黑棋的坐标集合
        Set<Integer> blackSet = readChessSet(in.nextLine());
        // 读取白棋的坐标集合
        Set<Integer> whiteSet = readChessSet(in.nextLine());
        // 输出黑棋和白棋的气数，两个数字以空格分隔
        System.out.println(counting(blackSet, whiteSet) + " " + counting(whiteSet, blackSet));
    }

    // 读取棋子的坐标集合
    static Set<Integer> readChessSet(String line) {
        String[] parts = line.split(" "); // 将输入的行按空格分割
        Set<Integer> chessSet = new HashSet<>(); // 创建一个HashSet来存储坐标
        for (int i = 0; i < parts.length; i += 2) {
            int x = Integer.parseInt(parts[i]); // 解析行坐标
            int y = Integer.parseInt(parts[i + 1]); // 解析列坐标
            // 将坐标转换为一个唯一的整数表示，并添加到集合中
            chessSet.add(x * 19 + y);
        }
        return chessSet; // 返回包含所有棋子坐标的集合
    }

    // 计算棋子的气数
    static int counting(Set<Integer> alias, Set<Integer> enemy) {
        // 创建一个新的HashSet，用于存储棋子的气
        Set<Integer> count = new HashSet<>(alias); // 初始包含所有己方棋子的位置
        for (int pos : alias) {
            int x = pos / 19; // 计算行坐标
            int y = pos % 19; // 计算列坐标
            // 分别检查上下左右四个方向，如果存在气则添加到集合中
            if (x > 0) count.add((x - 1) * 19 + y);
            if (x < maxSide) count.add((x + 1) * 19 + y);
            if (y > 0) count.add(x * 19 + (y - 1));
```

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

javaScript

```
// 最大的棋盘边界索引
const maxSide = 18;

// 读取棋子的坐标集合
function readChessSet(line) {
    const parts = line.split(' '); // 将输入的行按空格分割
    const chessSet = new Set(); // 创建一个Set来存储坐标
    for (let i = 0; i < parts.length; i += 2) {
        const x = parseInt(parts[i]); // 解析行坐标
        const y = parseInt(parts[i + 1]); // 解析列坐标
        // 将坐标转换为一个唯一的整数表示，并添加到集合中
        chessSet.add(x * 19 + y);
    }
    return chessSet; // 返回包含所有棋子坐标的集合
}

// 计算棋子的气数
function counting(alias, enemy) {
    // 创建一个新的Set，用于存储棋子的气
    const count = new Set(alias); // 初始包含所有己方棋子的位置
    for (let pos of alias) {
        const x = Math.floor(pos / 19); // 计算行坐标
        const y = pos % 19; // 计算列坐标
        // 分别检查上下左右四个方向，如果存在气则添加到集合中
```

```
        if (x > 0) count.add((x - 1) * 19 + y);
        if (x < maxSide) count.add((x + 1) * 19 + y);
        if (y > 0) count.add(x * 19 + (y - 1));
        if (y < maxSide) count.add(x * 19 + (y + 1));
    }
    // 计算得到的气数减去己方棋子的数量
    let res = count.size - alias.size;
    // 减去敌方棋子占据的气点
    for (let pos of enemy) {
        if (count.has(pos)) {
            res--;
        }
    }
    return res; // 返回最终的气数
}

const readline = require('readline').createInterface({
    input: process.stdin,
    output: process.stdout
});

readline.on('line', blackLine => {
    readline.on('line', whiteLine => {
        const blackSet = readChessSet(blackLine); // 读取黑棋的坐标集合
        const whiteSet = readChessSet(whiteLine); // 读取白棋的坐标集合
        // 输出黑棋和白棋的气数，两个数字以空格分隔
        console.log(counting(blackSet, whiteSet) + " " + counting(whiteSet, blackSet));
        readline.close();
    });
});
```

Python

```
maxSide = 18


def readChessSet(line):
    parts = line.split(' ')
    chessSet = set()
```

```
    for i in range(0, len(parts), 2):
        x = int(parts[i])
        y = int(parts[i + 1])
        chessSet.add(x * 19 + y)
    return chessSet


def counting(alias, enemy):
    count = set(alias)
    for pos in alias:
        x = pos // 19
        y = pos % 19
        if x > 0: count.add((x - 1) * 19 + y)
        if x < maxSide: count.add((x + 1) * 19 + y)
        if y > 0: count.add(x * 19 + (y - 1))
        if y < maxSide: count.add(x * 19 + (y + 1))

    res = len(count) - len(alias)

    for pos in enemy:
        if pos in count:
            res -= 1
    return res

# 主方法，程序入口
blackLine = input()
whiteLine = input()
blackSet = readChessSet(blackLine)
whiteSet = readChessSet(whiteLine)
print(counting(blackSet, whiteSet), counting(whiteSet, blackSet))
```

文章目录

解法1

解题思路

C++

Java

javaScript

Python

C语言

解法2

C++

Java

javaScript

Python