

【华为OD机考 统一考试机试C卷】最优结果的a数组数量/数组比较 (C++ Java JavaScript Python)

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

2023年11月份，华为官方已经将 华为OD机考：OD统一考试（A卷 / B卷）切换到 OD统一考试（C卷）和 OD统一考试（D卷）。根据考友反馈：目前抽到的试卷为B卷或C卷/D卷，其中C卷居多，按照之前的经验C卷D卷部分考题会复用A卷/B卷题，博主正积极从考过的同学收集C卷和D卷真题，可以查看下面的真题目录。

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](#) [华为OD机考B卷C卷](#) [华为OD机考华为OD机考B卷](#) [华为OD机试B卷](#) [华为OD机试C卷](#) [华为OD机考C卷](#) [华为OD机考D卷题目](#) [华为OD机考C卷/D卷答案](#) [华为OD机考C卷/D卷解析](#) [华为OD机考C卷和D卷真题](#) [华为OD机考C卷和D卷题解](#)

题目描述

给定两个只包含数字的数组a, b, 调整数组a里面数字的顺序，使得尽可能多的 $a[i] > b[i]$ 。数组a和b中的数字各不相同。
输出所有可以达到最优结果的a数组数量

输入描述

输入的第一行是数组a中的数字，其中只包含数字，每两个数字之间相隔一个空格，a数组大小不超过10

输入的第一行是数组b中的数字，其中只包含数字，每两个数字之间相隔一个空格，b数组大小不超过10

输出描述

输出所有可以达到最优结果的a数组数量

用例1

输入输出示例仅供调试，后台判题数据一般不包含示例

输入

```
1 | 11 8 20
2 |
3 | 10 13 7
```

输出

```
1 | 1
```

说明

最优结果只有一个，a = [11, 20, 8]，故输出1

用例2

输入

```
1 | 11 12 20
2 |
3 | 10 13 7
```

输出

```
1 | 2
```

说明

有两个a数组的排列可以达到最优结果，[12, 20, 11]和[11, 20, 12]，故输出2

用例3

输入

```
1 | 1 2 3
2 |
3 | 4 5 6
```

输出

1 | 6

说明

a无论如何都会全输，故a任意排列都行，输出所有a数组的排列，6种排法

解题思路

C++

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <sstream>
5  #include <iterator>
6  using namespace std;
7  // 存储最大匹配对的数量
8  int maxMatch = 0;
9  // 存储达到最大匹配对数量的排列数量
10 int optimalArrangements = 0;
11
12 // 辅助函数，用于交换数组中的两个元素
13 void swap(vector<int>& array, int i, int j) {
14     int temp = array[i];
15     array[i] = array[j];
16     array[j] = temp;
17 }
18
19 // 递归函数，用于找出数组a的任何排列中，与数组b的最大匹配对数量
20 void findMaxMatch(vector<int>& a, vector<int>& b, int index) {
21     // 递归终止条件，当索引等于数组长度时，计算当前排列的匹配对数量
22     if (index == a.size()) {
23         int match = 0;
24         for (int i = 0; i < a.size(); i++) {
25             if (a[i] > b[i]) {
26                 match++;
```

```
27     }
28 }
29 // 更新最大匹配对数量
30 maxMatch = max(maxMatch, match);
31 return;
32 }
33
34 // 递归产生a的所有排列
35 for (int i = index; i < a.size(); i++) {
36     swap(a, index, i);
37     findMaxMatch(a, b, index + 1);
38     swap(a, index, i); // 回溯, 恢复数组状态
39 }
40 }
41
42 // 递归函数, 用于生成数组a的所有排列, 并计算每个排列的匹配对数量
43 void permute(vector<int>& a, int index, vector<int>& b) {
44     // 递归终止条件, 当索引等于数组长度时, 检查当前排列是否达到最大匹配对数量
45     if (index == a.size()) {
46         int match = 0;
47         for (int i = 0; i < a.size(); i++) {
48             if (a[i] > b[i]) {
49                 match++;
50             }
51         }
52         // 如果达到最大匹配对数量, 增加计数
53         if (match == maxMatch) {
54             optimalArrangements++;
55         }
56         return;
57     }
58
59     // 递归产生a的所有排列
60     for (int i = index; i < a.size(); i++) {
61         swap(a, index, i);
62         permute(a, index + 1, b);
63         swap(a, index, i); // 回溯, 恢复数组状态
64     }
65 }
66
67 }
```

```
68 // 主函数, 读取输入, 调用解决方案并输出最优排列数量
69 int main() {
70     string line;
71     getline(cin, line);
72     istringstream iss(line);
73     vector<int> a((istream_iterator<int>(iss)), istream_iterator<int>());
74
75     getline(cin, line);
76     iss.str(line);
77     iss.clear();
78     vector<int> b((istream_iterator<int>(iss)), istream_iterator<int>());
79
80     // 对数组b进行排序, 以便后续比较
81     sort(b.begin(), b.end());
82
83     // 首先计算最大匹配对的数量
84     findMaxMatch(a, b, 0);
85
86     // 然后计算所有可能的a的排列, 并筛选出达到最大匹配对数量的排列
87     permute(a, 0, b);
88
89     cout << optimalArrangements << endl;
90
91     return 0;
}
```

Java

```
1 import java.util.Arrays;
2 import java.util.Scanner;
3
4 public class Main {
5     // 存储最大匹配对的数量
6     private static int maxMatch = 0;
7     // 存储达到最大匹配对数量的排列数量
8     private static int optimalArrangements = 0;
9
10    public static void main(String[] args) {
11        Scanner scanner = new Scanner(System.in);
12
13    }
```

```
13 // 从标准输入读取两行, 将每行的字符串分割并转换成整数数组
14 int[] a = Arrays.stream(scanner.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
15 int[] b = Arrays.stream(scanner.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
16
17 // 调用解决方案并输出最优排列数量
18 solve(a, b);
19 System.out.println(optimalArrangements);
20 }
21
22 private static void solve(int[] a, int[] b) {
23     // 对数组b进行排序, 以便后续比较
24     Arrays.sort(b);
25
26     // 首先计算最大匹配对的数量
27     findMaxMatch(a, b, 0);
28
29     // 然后计算所有可能的a的排列, 并筛选出达到最大匹配对数量的排列
30     permute(a, 0, b);
31 }
32
33 // 递归函数, 用于找出数组a的任何排列中, 与数组b的最大匹配对数量
34 private static void findMaxMatch(int[] a, int[] b, int index) {
35     // 递归终止条件, 当索引等于数组长度时, 计算当前排列的匹配对数量
36     if (index == a.length) {
37         int match = 0;
38         for (int i = 0; i < a.length; i++) {
39             if (a[i] > b[i]) {
40                 match++;
41             }
42         }
43         // 更新最大匹配对数量
44         maxMatch = Math.max(maxMatch, match);
45         return;
46     }
47
48     // 递归产生a的所有排列
49     for (int i = index; i < a.length; i++) {
50         swap(a, index, i);
51         findMaxMatch(a, b, index + 1);
52         swap(a, index, i); // 回溯, 恢复数组状态
53     }
```

```
54     }
55 }
56
57 // 递归函数, 用于生成数组a的所有排列, 并计算每个排列的匹配对数量
58 private static void permute(int[] a, int index, int[] b) {
59     // 递归终止条件, 当索引等于数组长度时, 检查当前排列是否达到最大匹配对数量
60     if (index == a.length) {
61         int match = 0;
62         for (int i = 0; i < a.length; i++) {
63             if (a[i] > b[i]) {
64                 match++;
65             }
66         }
67         // 如果达到最大匹配对数量, 增加计数
68         if (match == maxMatch) {
69             optimalArrangements++;
70         }
71         return;
72     }
73
74     // 递归产生a的所有排列
75     for (int i = index; i < a.length; i++) {
76         swap(a, index, i);
77         permute(a, index + 1, b);
78         swap(a, index, i); // 回溯, 恢复数组状态
79     }
80 }
81
82 // 辅助函数, 用于交换数组中的两个元素
83 private static void swap(int[] array, int i, int j) {
84     int temp = array[i];
85     array[i] = array[j];
86     array[j] = temp;
87 }
}
```

javaScript

```
1 const readline = require('readline');
2
~
```



```
3 // 创建readline接口实例
4 const rl = readline.createInterface({
5   input: process.stdin,
6   output: process.stdout
7 });
8
9 // 存储最大匹配对的数量
10 let maxMatch = 0;
11 // 存储达到最大匹配对数量的排列数量
12 let optimalArrangements = 0;
13
14 // 辅助函数, 用于交换数组中的两个元素
15 function swap(array, i, j) {
16   const temp = array[i];
17   array[i] = array[j];
18   array[j] = temp;
19 }
20
21 // 递归函数, 用于找出数组a的任何排列中, 与数组b的最大匹配对数量
22 function findMaxMatch(a, b, index) {
23   if (index === a.length) {
24     let match = 0;
25     for (let i = 0; i < a.length; i++) {
26       if (a[i] > b[i]) {
27         match++;
28       }
29     }
30     maxMatch = Math.max(maxMatch, match);
31     return;
32   }
33
34   for (let i = index; i < a.length; i++) {
35     swap(a, index, i);
36     findMaxMatch(a, b, index + 1);
37     swap(a, index, i); // 回溯, 恢复数组状态
38   }
39 }
40
41 // 递归函数, 用于生成数组a的所有排列, 并计算每个排列的匹配对数量
42 function permute(a, index, b) {
43   ...
```

```
44     if (index === a.length) {
45         let match = 0;
46         for (let i = 0; i < a.length; i++) {
47             if (a[i] > b[i]) {
48                 match++;
49             }
50         }
51         if (match === maxMatch) {
52             optimalArrangements++;
53         }
54         return;
55     }
56
57     for (let i = index; i < a.length; i++) {
58         swap(a, index, i);
59         permute(a, index + 1, b);
60         swap(a, index, i); // 回溯, 恢复数组状态
61     }
62 }
63 let a;
64 let b;
65 // 从标准输入读取数据
66 rl.on('line', (line) => {
67     const inputs = line.split(' ').map(Number);
68     if (!a) {
69         a = inputs;
70     } else {
71         b = inputs;
72         rl.close();
73     }
74 }).on('close', () => {
75     // 对数组b进行排序, 以便后续比较
76     b.sort((x, y) => x - y);
77
78     // 调用解决方案并输出最优排列数量
79     findMaxMatch(a, b, 0);
80     permute(a, 0, b);
81     console.log(optimalArrangements);
82 });
```

Python

```
1 import itertools
2
3 # 存储最大匹配对的数量
4 max_match = 0
5 # 存储达到最大匹配对数量的排列数量
6 optimal_arrangements = 0
7
8 def swap(array, i, j):
9     """
10     辅助函数，用于交换数组中的两个元素
11     """
12     array[i], array[j] = array[j], array[i]
13
14 def find_max_match(a, b, index):
15     """
16     递归函数，用于找出数组a的任何排列中，与数组b的最大匹配对数量
17     """
18     global max_match
19     if index == len(a):
20         match = sum(1 for i in range(len(a)) if a[i] > b[i])
21         max_match = max(max_match, match)
22         return
23
24     for i in range(index, len(a)):
25         swap(a, index, i)
26         find_max_match(a, b, index + 1)
27         swap(a, index, i) # 回溯，恢复数组状态
28
29 def permute(a, b):
30     """
31     函数，用于生成数组a的所有排列，并计算每个排列的匹配对数量
32     """
33     global optimal_arrangements
34     for p in itertools.permutations(a):
35         if sum(1 for i in range(len(a)) if p[i] > b[i]) == max_match:
36             optimal_arrangements += 1
37
38 # 从标准输入读取两行数据
39
```

```
40 a = list(map(int, input().split()))
41 b = list(map(int, input().split()))
42
43 # 对数组b进行排序, 以便后续比较
44 b.sort()
45
46 # 调用解决方案并输出最优排列数量
47 find_max_match(a, b, 0)
48 permute(a, b)
   print(optimal_arrangements)
```

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1](#)

[用例2](#)

[用例3](#)

[解题思路](#)

[C++](#)

[Java](#)

[javaScript](#)

[Python](#)

机考真题 华为OD



CSDN @算法大师