【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

```
11 8 20

10 13 7
```

输出

```
1
```

说明

> 最优结果只有一个，a = [11, 20, 8]，故输出1

用例2

输入

```
11 12 20

10 13 7
```

输出

```
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 2 3

4 5 6
```

输出

```
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++

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <sstream>
#include <iterator>
using namespace std;
// 存储最大匹配对的数量
int maxMatch = 0;
// 存储达到最大匹配对数量的排列数量
int optimalArrangements = 0;

// 辅助函数，用于交换数组中的两个元素
void swap(vector<int>& array, int i, int j) {
    int temp = array[i];
    array[i] = array[j];
    array[j] = temp;
}

// 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
void findMaxMatch(vector<int>& a, vector<int>& b, int index) {
    // 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
    if (index == a.size()) {
        int match = 0;
        for (int i = 0; i < a.size(); i++) {
            if (a[i] > b[i]) {
                match++;
```

```
            }
        }
        // 更新最大匹配对数量
        maxMatch = max(maxMatch, match);
        return;
    }

    // 递归产生a的所有排列
    for (int i = index; i < a.size(); i++) {
        swap(a, index, i);
        findMaxMatch(a, b, index + 1);
        swap(a, index, i); // 回溯，恢复数组状态
    }
}

// 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
void permute(vector<int>& a, int index, vector<int>& b) {
    // 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
    if (index == a.size()) {
        int match = 0;
        for (int i = 0; i < a.size(); i++) {
            if (a[i] > b[i]) {
                match++;
            }
        }
        // 如果达到最大匹配对数量，增加计数
        if (match == maxMatch) {
            optimalArrangements++;
        }
        return;
    }

    // 递归产生a的所有排列
    for (int i = index; i < a.size(); i++) {
        swap(a, index, i);
        permute(a, index + 1, b);
        swap(a, index, i); // 回溯，恢复数组状态
    }
}


```

```cpp
// 主函数，读取输入，调用解决方案并输出最优排列数量
int main() {
    string line;
    getline(cin, line);
    istringstream iss(line);
    vector<int> a((istream_iterator<int>(iss)), istream_iterator<int>());

    getline(cin, line);
    iss.str(line);
    iss.clear();
    vector<int> b((istream_iterator<int>(iss)), istream_iterator<int>());

    // 对数组b进行排序，以便后续比较
    sort(b.begin(), b.end());

    // 首先计算最大匹配对的数量
    findMaxMatch(a, b, 0);

    // 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
    permute(a, 0, b);

    cout << optimalArrangements << endl;

    return 0;
}
```

Java

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main {
    // 存储最大匹配对的数量
    private static int maxMatch = 0;
    // 存储达到最大匹配对数量的排列数量
    private static int optimalArrangements = 0;

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

```

```
        // 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
        int[] a = Arrays.stream(scanner.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
        int[] b = Arrays.stream(scanner.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();

        // 调用解决方案并输出最优排列数量
        solve(a, b);
        System.out.println(optimalArrangements);
    }

    private static void solve(int[] a, int[] b) {
        // 对数组b进行排序，以便后续比较
        Arrays.sort(b);

        // 首先计算最大匹配对的数量
        findMaxMatch(a, b, 0);

        // 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
        permute(a, 0, b);
    }

    // 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
    private static void findMaxMatch(int[] a, int[] b, int index) {
        // 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
        if (index == a.length) {
            int match = 0;
            for (int i = 0; i < a.length; i++) {
                if (a[i] > b[i]) {
                    match++;
                }
            }
            // 更新最大匹配对数量
            maxMatch = Math.max(maxMatch, match);
            return;
        }

        // 递归产生a的所有排列
        for (int i = index; i < a.length; i++) {
            swap(a, index, i);
            findMaxMatch(a, b, index + 1);
            swap(a, index, i); // 回溯，恢复数组状态
```

```
        }
    }

    // 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
    private static void permute(int[] a, int index, int[] b) {
        // 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
        if (index == a.length) {
            int match = 0;
            for (int i = 0; i < a.length; i++) {
                if (a[i] > b[i]) {
                    match++;
                }
            }
            // 如果达到最大匹配对数量，增加计数
            if (match == maxMatch) {
                optimalArrangements++;
            }
            return;
        }

        // 递归产生a的所有排列
        for (int i = index; i < a.length; i++) {
            swap(a, index, i);
            permute(a, index + 1, b);
            swap(a, index, i); // 回溯，恢复数组状态
        }
    }

    // 辅助函数，用于交换数组中的两个元素
    private static void swap(int[] array, int i, int j) {
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }
}
```

javaScript

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

```

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

// 存储最大匹配对的数量
let maxMatch = 0;
// 存储达到最大匹配对数量的排列数量
let optimalArrangements = 0;

// 辅助函数，用于交换数组中的两个元素
function swap(array, i, j) {
    const temp = array[i];
    array[i] = array[j];
    array[j] = temp;
}

// 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
function findMaxMatch(a, b, index) {
    if (index === a.length) {
        let match = 0;
        for (let i = 0; i < a.length; i++) {
            if (a[i] > b[i]) {
                match++;
            }
        }
        maxMatch = Math.max(maxMatch, match);
        return;
    }

    for (let i = index; i < a.length; i++) {
        swap(a, index, i);
        findMaxMatch(a, b, index + 1);
        swap(a, index, i); // 回溯，恢复数组状态
    }
}

// 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
function permute(a, index, b) {
```

```
    if (index === a.length) {
        let match = 0;
        for (let i = 0; i < a.length; i++) {
            if (a[i] > b[i]) {
                match++;
            }
        }
        if (match === maxMatch) {
            optimalArrangements++;
        }
        return;
    }

    for (let i = index; i < a.length; i++) {
        swap(a, index, i);
        permute(a, index + 1, b);
        swap(a, index, i); // 回溯，恢复数组状态
    }
}
let a;
let b;
// 从标准输入读取数据
rl.on('line', (line) => {
    const inputs = line.split(' ').map(Number);
    if (!a) {
        a = inputs;
    } else {
        b = inputs;
         rl.close();
    }
}).on('close', () => {
    // 对数组b进行排序，以便后续比较
    b.sort((x, y) => x - y);

    // 调用解决方案并输出最优排列数量
    findMaxMatch(a, b, 0);
    permute(a, 0, b);
    console.log(optimalArrangements);
});
```

Python

```python
import itertools

# 存储最大匹配对的数量
max_match = 0
# 存储达到最大匹配对数量的排列数量
optimal_arrangements = 0

def swap(array, i, j):
    """
    辅助函数，用于交换数组中的两个元素
    """
    array[i], array[j] = array[j], array[i]

def find_max_match(a, b, index):
    """
    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
    """
    global max_match
    if index == len(a):
        match = sum(1 for i in range(len(a)) if a[i] > b[i])
        max_match = max(max_match, match)
        return

    for i in range(index, len(a)):
        swap(a, index, i)
        find_max_match(a, b, index + 1)
        swap(a, index, i)  # 回溯，恢复数组状态

def permute(a, b):
    """
    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
    """
    global optimal_arrangements
    for p in itertools.permutations(a):
        if sum(1 for i in range(len(a)) if p[i] > b[i]) == max_match:
            optimal_arrangements += 1

# 从标准输入读取两行数据

```

```python
a = list(map(int, input().split()))
b = list(map(int, input().split()))

# 对数组b进行排序，以便后续比较
b.sort()

# 调用解决方案并输出最优排列数量
find_max_match(a, b, 0)
permute(a, b)
print(optimal_arrangements)
```

文章目录


机考真题
华为OD
No No No
CSDN @算法大师