【华为OD机考 统一考试机试C卷】 找出作弊的人（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

公司组织了一次考试,现在考试结果出来了，想看一下有没人存在作弊行为,但是员工太多了,需要先对员工进行一次过滤,再进一步确定是否存在作弊行为。

过滤的规则为:找到分差最小的员工ID对(p1,p2)列表,要求p1<p2

员工个数取值范国:O<n<100000

员工ID为整数,取值范围:0<=n<=100000

考试成绩为整数,取值范围:0<=score<=300

输入描述

员工的ID及考试分数

输出描述

分差最小的员工ID对(p1,p2)列表,要求p1<p2。每一行代表一个集合,每个集合内的员工ID按顺序排列,多行结果也以员工对中p1值大小升序排列(如果p1相同则p2升序)。

样例1

输入：

```
5
1 90
2 91
3 95
4 96
5 100
```

输出:

```
1 2
3 4
```

解释:

> 输入：第一行为员工个数n，后续的n行第一个数值为员工ID,第二个数值为员工考试分数
> 输出:员工1和员工2的分差为1,员工3和员工4的分差也为1,因此最终结果为
> 1 2
> 3 4

样例2

输入：

```
5
1 90
2 91
3 92
4 85
5 86
```

输出：

```
1 2
2 3
4 5
```

解题思路

C++

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <climits>

using namespace std;

int main() {
    // 读取员工的数量
    int n;
    cin >> n;

    // 创建一个vector用于存储员工的ID和分数
    vector<pair<int, int>> employees(n);

    // 读取每个员工的ID和分数，并将其添加到vector中
    for (int i = 0; i < n; i++) {
        cin >> employees[i].first >> employees[i].second;
    }

    // 对员工vector按照分数进行排序
    sort(employees.begin(), employees.end(), [](const pair<int, int>& a, const pair<int, int>& b) {
        return a.second < b.second;
    });

    // 初始化最小分差为INT_MAX
    int minDiff = INT_MAX;
    // 创建一个vector用于存储分差最小的员工ID对
    vector<pair<int, int>> result;

```

```
    // 遍历排序后的员工vector，计算相邻员工的分差
    for (int i = 1; i < n; i++) {
        int diff = employees[i].second - employees[i - 1].second;
        // 如果当前分差小于最小分差，则更新最小分差，并清空结果vector，将当前员工ID对添加到结果vector中
        if (diff < minDiff) {
            minDiff = diff;
            result.clear();
            result.push_back({employees[i - 1].first, employees[i].first});
        }
        // 如果当前分差等于最小分差，则将当前员工ID对添加到结果vector中
        else if (diff == minDiff) {
            result.push_back({employees[i - 1].first, employees[i].first});
        }
    }

    // 对结果vector按照员工ID进行排序
    sort(result.begin(), result.end());

    // 打印出分差最小的员工ID对
    for (const auto& pair : result) {
        cout << pair.first << " " << pair.second << endl;
    }

    return 0;
}
```

Java

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        // 创建一个Scanner对象用于读取输入
        Scanner scanner = new Scanner(System.in);
        // 读取员工的数量
        int n = scanner.nextInt();
```

```java
        // 创建一个List用于存储员工的ID和分数
        List<int[]> employees = new ArrayList<>();

        // 读取每个员工的ID和分数，并将其添加到list中
        for (int i = 0; i < n; i++) {
            int id = scanner.nextInt();
            int score = scanner.nextInt();
            employees.add(new int[]{id, score});
        }

        // 关闭Scanner对象
        scanner.close();

        // 对员工List按照分数进行排序
        employees.sort(Comparator.comparingInt(a -> a[1]));

        // 初始化最小分差为Integer的最大值
        int minDiff = Integer.MAX_VALUE;
        // 创建一个List用于存储分差最小的员工ID对
        List<int[]> result = new ArrayList<>();

        // 遍历排序后的员工List，计算相邻员工的分差
        for (int i = 1; i < n; i++) {
            int diff = employees.get(i)[1] - employees.get(i - 1)[1];
            // 如果当前分差小于最小分差，则更新最小分差，并清空结果List，将当前员工ID对添加到结果List中
            if (diff < minDiff) {
                minDiff = diff;
                result.clear();
                result.add(new int[]{employees.get(i - 1)[0], employees.get(i)[0]});
            }
            // 如果当前分差等于最小分差，则将当前员工ID对添加到结果List中
            else if (diff == minDiff) {
                result.add(new int[]{employees.get(i - 1)[0], employees.get(i)[0]});
            }
        }

        // 对结果List按照员工ID进行排序
        result.sort(Comparator.comparingInt(a -> a[0]));

        // 打印出分差最小的员工ID对
```

```
        for (int[] pair : result) {
            System.out.println(pair[0] + " " + pair[1]);
        }
    }
}
```

javaScript

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

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

let input = [];

rl.on('line', (line) => {
    input.push(line.trim());
}).on('close', () => {
    // 读取员工的数量
    const n = parseInt(input.shift());
    // 创建一个数组用于存储员工的ID和分数
    const employees = input.map(line => line.split(' ').map(Number));

    // 对员工数组按照分数进行排序
    employees.sort((a, b) => a[1] - b[1]);

    // 初始化最小分差为Number的最大值
    let minDiff = Number.MAX_SAFE_INTEGER;
    // 创建一个数组用于存储分差最小的员工ID对
    let result = [];

    // 遍历排序后的员工数组，计算相邻员工的分差
    for (let i = 1; i < n; i++) {
        const diff = employees[i][1] - employees[i - 1][1];
        // 如果当前分差小于最小分差，则更新最小分差，并清空结果数组，将当前员工ID对添加到结果数组中
        if (diff < minDiff) {
            minDiff = diff;
            result = [[employees[i - 1][0], employees[i][0]]];
```

```
        }
        // 如果当前分差等于最小分差，则将当前员工ID对添加到结果数组中
        else if (diff === minDiff) {
            result.push([employees[i - 1][0], employees[i][0]]);
        }
    }

    // 对结果数组按照员工ID进行排序
    result.sort((a, b) => a[0] - b[0]);

    // 打印出分差最小的员工ID对
    for (const pair of result) {
        console.log(pair.join(' '));
    }
});
```

Python

```
# 读取员工的数量
n = int(input())

# 创建一个list用于存储员工的ID和分数
employees = []

# 读取每个员工的ID和分数，并将其添加到list中
for _ in range(n):
    id, score = map(int, input().split())
    employees.append((id, score))

# 对员工list按照分数进行排序
employees.sort(key=lambda x: x[1])

# 初始化最小分差为无穷大
min_diff = float('inf')

# 创建一个list用于存储分差最小的员工ID对
result = []

# 遍历排序后的员工list，计算相邻员工的分差
for i in range(1, n):
```

```
    diff = employees[i][1] - employees[i - 1][1]
    # 如果当前分差小于最小分差，则更新最小分差，并清空结果list，将当前员工ID对添加到结果list中
    if diff < min_diff:
        min_diff = diff
        result = [(employees[i - 1][0], employees[i][0])]
    # 如果当前分差等于最小分差，则将当前员工ID对添加到结果list中
    elif diff == min_diff:
        result.append((employees[i - 1][0], employees[i][0]))

# 对结果list按照员工ID进行排序
result.sort()

# 打印出分差最小的员工ID对
for pair in result:
    print(pair[0], pair[1])
```

C语言

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

// 定义结构体，用于存储员工的ID和分数
typedef struct {
    int id;
    int score;
} Employee;

// 定义比较函数，用于qsort函数对员工按照分数进行排序
int compareByScore(const void* a, const void* b) {
    return ((Employee*)a)->score - ((Employee*)b)->score;
}

// 定义比较函数，用于qsort函数对员工ID对按照ID进行排序
int compareById(const void* a, const void* b) {
    int id1 = ((Employee*)a)->id;
    int id2 = ((Employee*)b)->id;
    return id1 - id2;
}

int main() {
```

```
    // 读取员工的数量
    int n;
    scanf("%d", &n);

    // 创建一个动态数组用于存储员工的ID和分数
    Employee* employees = (Employee*)malloc(n * sizeof(Employee));

    // 读取每个员工的ID和分数，并将其添加到动态数组中
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &employees[i].id, &employees[i].score);
    }

    // 对员工动态数组按照分数进行排序
    qsort(employees, n, sizeof(Employee), compareByScore);

    // 初始化最小分差为INT_MAX
    int minDiff = 2147483647;
    // 创建一个动态数组用于存储分差最小的员工ID对
    Employee* result = (Employee*)malloc(n * sizeof(Employee));
    int resultSize = 0;

    // 遍历排序后的员工动态数组，计算相邻员工的分差
    for (int i = 1; i < n; i++) {
        int diff = employees[i].score - employees[i - 1].score;
        // 如果当前分差小于最小分差，则更新最小分差，并清空结果动态数组，将当前员工ID对添加到结果动态数组中
        if (diff < minDiff) {
            minDiff = diff;
            resultSize = 0;
            result[resultSize].id = employees[i - 1].id;
            result[resultSize].score = employees[i].id;
            resultSize++;
        }
        // 如果当前分差等于最小分差，则将当前员工ID对添加到结果动态数组中
        else if (diff == minDiff) {
            result[resultSize].id = employees[i - 1].id;
            result[resultSize].score = employees[i].id;
            resultSize++;
        }
    }
```

```
    // 对结果动态数组按照员工ID进行排序
    qsort(result, resultSize, sizeof(Employee), compareById);

    // 打印出分差最小的员工ID对
    for (int i = 0; i < resultSize; i++) {
        printf("%d %d\n", result[i].id, result[i].score);
    }

    // 释放动态分配的内存
    free(employees);
    free(result);

    return 0;
}
```

文章目录


机考真题
华为OD
No No No
CSDN @算法大师