【华为OD机考 统一考试机试C卷】查找接口成功率最优时间段（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

服务之间交换的接口成功率作为服务调用关键质量特性，某个时间段内的接口失败率使用一个数组表示，

数组中每个元素都是单位时间内失败率数值，数组中的数值为0~100的整数，

给定一个数值(minAverageLost)表示某个时间段内平均失败率容忍值，即平均失败率小于等于minAverageLost，

找出数组中最长时间段，如果未找到则直接返回NULL。

输入描述

输入有两行内容，第一行为{minAverageLost}，第二行为{数组}，数组元素通过空格(" ")分隔，

minAverageLost及数组中元素取值范围为0~100的整数，数组元素的个数不会超过100个。

输出描述

找出平均值小于等于minAverageLost的最长时间段，输出数组下标对，格式{beginIndex}-{endIndx}(下标从0开始)，

如果同时存在多个最长时间段，则输出多个下标对且下标对之间使用空格(" ")拼接，多个下标对按下标从小到大排序。

用例

输入	1 0 1 2 3 4
输出	0-2
说明	**输入解释：**minAverageLost=1，数组[0, 1, 2, 3, 4] 前3个元素的平均值为1，因此数组第一个至第三个数组下标，即0-2

输入	2 0 0 100 2 2 99 0 2
输出	0-1 3-4 6-7
说明	输入解释：minAverageLost=2，数组[0, 0, 100, 2, 2, 99, 0, 2] 通过计算小于等于2的最长时间段为： 数组下标为0-1即[0, 0]，数组下标为3-4即[2, 2]，数组下标为6-7即[0, 2]，这三个部分都满足平均值小于等于2的要求， 因此输出0-1 3-4 6-7

题目解析

解题思路如下：

1. 首先，我们需要读取输入的数据，包括容忍的平均失败率和失败率数组。
2. 然后，我们创建一个累积和数组，用于快速计算任意时间段的失败率总和。这个数组的每个元素都是从数组开始到当前位置的失败率的总和。
3. 接下来，我们遍历所有可能的时间段，包括所有可能的开始和结束索引。对于每个时间段，我们计算其失败率总和，然后计算其平均失败率。我们可以通过查找累积和数组来快速计算失败率总和。
4. 对于每个时间段，我们检查其平均失败率是否小于等于容忍的平均失败率。如果是，我们就找到了一个满足条件的时间段。

5. 我们需要找到最长的满足条件的时间段。因此，我们需要跟踪找到的最长时间段的长度。如果我们找到一个比当前最长时间段更长的时间段，我们就更新最长时间段的长度，并清空结果列表，然后将新的时间段添加到结果列表中。如果我们找到一个和当前最长时间段一样长的时间段，我们就将它添加到结果列表中。

6. 最后，我们检查结果列表。如果结果列表为空，说明我们没有找到任何满足条件的时间段，我们就输出"NULL"。否则，我们输出所有满足条件的时间段。如果有多个时间段，我们需要按照开始索引从小到大的顺序输出。

这个解题思路的关键是使用累积和数组来快速计算任意时间段的失败率总和，以及使用一个结果列表来跟踪所有满足条件的时间段。这样，我们可以在一次遍历中找到所有满足条件的时间段，并且可以快速找到最长的时间段。

C++

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

using namespace std;

int main() {
    // 容忍的平均失败率
    int toleratedAverageLoss;
    cin >> toleratedAverageLoss;

    // 读取失败率数组
    vector<int> failureRates;
    string line;
    getline(cin >> ws, line);
    istringstream iss(line);
    int num;
    while (iss >> num) {
        failureRates.push_back(num);
    }

    int arrayLength = failureRates.size();

    // 创建一个累积和数组，用于快速计算任意时间段的失败率总和
    vector<int> cumulativeSum(arrayLength);
    cumulativeSum[0] = failureRates[0];
    for (int i = 1; i < arrayLength; i++) cumulativeSum[i] = cumulativeSum[i - 1] + failureRates[i];
```

```
29
30      // 存储满足条件的时间段的开始和结束索引
31      vector<pair<int, int>> validPeriods;
32      int maxLength = 0;
33      for (int start = 0; start < arrayLength; start++) {
34          for (int end = start; end < arrayLength; end++) {
35              int sum = start == 0 ? cumulativeSum[end] : cumulativeSum[end] - cumulativeSum[start - 1];
36              int length = end - start + 1;
37              int toleratedLoss = length * toleratedAverageLoss;
38
39              // 如果这个时间段的平均失败率小于等于容忍的平均失败率
40              if (sum <= toleratedLoss) {
41                  // 如果这个时间段比之前找到的时间段更长，清空结果列表并添加这个时间段
42                  if (length > maxLength) {
43                      validPeriods.clear();
44                      validPeriods.push_back({start, end});
45                      maxLength = length;
46                  }
47                  // 如果这个时间段和之前找到的最长时间段一样长，添加这个时间段
48                  else if (length == maxLength) {
49                      validPeriods.push_back({start, end});
50                  }
51              }
52          }
53      }
54
55      // 如果没有找到满足条件的时间段，输出"NULL"
56      if (validPeriods.empty()) {
57          cout << "NULL" << endl;
58      }
59      // 否则，输出所有满足条件的时间段
60      else {
61          sort(validPeriods.begin(), validPeriods.end());
62
63          for (auto& period : validPeriods) {
64              cout << period.first << "-" << period.second << " ";
65          }
66          cout << endl;
67      }
68
69
```

```
    return 0;
}
```

JavaScript

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



readline.on('line', tolerated => {
 const toleratedAverageLoss = parseInt(tolerated);
  readline.on('line', rates => {a
    const failureRates = rates.split(' ').map(Number);

    const arrayLength = failureRates.length;

    // 创建一个累积和数组，用于快速计算任意时间段的失败率总和
    const cumulativeSum = new Array(arrayLength);
    cumulativeSum[0] = failureRates[0];
    for (let i = 1; i < arrayLength; i++) cumulativeSum[i] = cumulativeSum[i - 1] + failureRates[i];

    // 存储满足条件的时间段的开始和结束索引
    let validPeriods = [];
    let maxLength = 0;
    for (let start = 0; start < arrayLength; start++) {
      for (let end = start; end < arrayLength; end++) {
        const sum = start === 0 ? cumulativeSum[end] : cumulativeSum[end] - cumulativeSum[start - 1];
        const length = end - start + 1;
        const toleratedLoss = length * toleratedAverageLoss;

        // 如果这个时间段的平均失败率小于等于容忍的平均失败率
        if (sum <= toleratedLoss) {
          // 如果这个时间段比之前找到的时间段更长，清空结果列表并添加这个时间段
          if (length > maxLength) {
            validPeriods = [];
            validPeriods.push([start, end]);
```

```
          maxLength = length;
        }
        // 如果这个时间段和之前找到的最长时间段一样长，添加这个时间段
        else if (length === maxLength) {
          validPeriods.push([start, end]);
        }
      }
    }
  }

  // 如果没有找到满足条件的时间段，输出"NULL"
  if (validPeriods.length === 0) {
    console.log("NULL");
  }
  // 否则，输出所有满足条件的时间段
  else {
    validPeriods.sort((a, b) => a[0] - b[0]);

    console.log(validPeriods.map(period => period.join('-')).join(' '));
  }
});
});
```

Java

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

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

    // 容忍的平均失败率
    int toleratedAverageLoss = Integer.parseInt(scanner.nextLine());

    // 读取失败率数组
    Integer[] failureRates =
```

```
    Arrays.stream(scanner.nextLine().split(" ")).map(Integer::parseInt).toArray(Integer[]::new);

int arrayLength = failureRates.length;

// 创建一个累积和数组，用于快速计算任意时间段的失败率总和
int[] cumulativeSum = new int[arrayLength];
cumulativeSum[0] = failureRates[0];
for (int i = 1; i < arrayLength; i++) cumulativeSum[i] = cumulativeSum[i - 1] + failureRates[i];

// 存储满足条件的时间段的开始和结束索引
ArrayList<Integer[]> validPeriods = new ArrayList<>();
int maxLength = 0;
for (int start = 0; start < arrayLength; start++) {
  for (int end = start; end < arrayLength; end++) {
    int sum = start == 0 ? cumulativeSum[end] : cumulativeSum[end] - cumulativeSum[start - 1];
    int length = end - start + 1;
    int toleratedLoss = length * toleratedAverageLoss;

    // 如果这个时间段的平均失败率小于等于容忍的平均失败率
    if (sum <= toleratedLoss) {
      // 如果这个时间段比之前找到的时间段更长，清空结果列表并添加这个时间段
      if (length > maxLength) {
        validPeriods = new ArrayList<>();
        validPeriods.add(new Integer[] {start, end});
        maxLength = length;
      }
      // 如果这个时间段和之前找到的最长时间段一样长，添加这个时间段
      else if (length == maxLength) {
        validPeriods.add(new Integer[] {start, end});
      }
    }
  }
}

// 如果没有找到满足条件的时间段，输出"NULL"
if (validPeriods.size() == 0) {
  System.out.println("NULL");
}
// 否则，输出所有满足条件的时间段
else {
```

```
        validPeriods.sort((a, b) -> a[0] - b[0]);

        StringJoiner sj = new StringJoiner(" ");
        for (Integer[] period : validPeriods) sj.add(period[0] + "-" + period[1]);
        System.out.println(sj.toString());
    }
  }
}
```

Python

```python
# 容忍的平均失败率
toleratedAverageLoss = int(input())

# 读取失败率数组
failureRates = list(map(int, input().split()))

arrayLength = len(failureRates)

# 创建一个累积和数组，用于快速计算任意时间段的失败率总和
cumulativeSum = [0] * arrayLength
cumulativeSum[0] = failureRates[0]
for i in range(1, arrayLength):
    cumulativeSum[i] = cumulativeSum[i - 1] + failureRates[i]

# 存储满足条件的时间段的开始和结束索引
validPeriods = []
maxLength = 0
for start in range(arrayLength):
    for end in range(start, arrayLength):
        sum = cumulativeSum[end] if start == 0 else cumulativeSum[end] - cumulativeSum[start - 1]
        length = end - start + 1
        toleratedLoss = length * toleratedAverageLoss

        # 如果这个时间段的平均失败率小于等于容忍的平均失败率
        if sum <= toleratedLoss:
            # 如果这个时间段比之前找到的时间段更长，清空结果列表并添加这个时间段
            if length > maxLength:
                validPeriods = []
```

```
                validPeriods.append((start, end))
                maxLength = length
            # 如果这个时间段和之前找到的最长时间段一样长，添加这个时间段
            elif length == maxLength:
                validPeriods.append((start, end))

# 如果没有找到满足条件的时间段，输出"NULL"
if len(validPeriods) == 0:
    print("NULL")
# 否则，输出所有满足条件的时间段
else:
    validPeriods.sort()

    print(' '.join(f'{start}-{end}' for start, end in validPeriods))
```

C语言

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

int main() {
    // 容忍的平均失败率
    int toleratedAverageLoss;
    scanf("%d", &toleratedAverageLoss);

    // 读取失败率数组
    int failureRates[100];
    int arrayLength = 0;
    while (scanf("%d", &failureRates[arrayLength]) == 1) {
        arrayLength++;
    }

    // 创建一个累积和数组，用于快速计算任意时间段的失败率总和
    int cumulativeSum[100] = {0};
    cumulativeSum[0] = failureRates[0];
    for (int i = 1; i < arrayLength; i++) {
        cumulativeSum[i] = cumulativeSum[i - 1] + failureRates[i];
    }
```

```
    // 存储满足条件的时间段的开始和结束索引
    int validPeriods[100][2];
    int validPeriodCount = 0;
    int maxLength = 0;
    for (int start = 0; start < arrayLength; start++) {
        for (int end = start; end < arrayLength; end++) {
            int sum = start == 0 ? cumulativeSum[end] : cumulativeSum[end] - cumulativeSum[start - 1];
            int length = end - start + 1;
            int toleratedLoss = length * toleratedAverageLoss;

            // 如果这个时间段的平均失败率小于等于容忍的平均失败率
            if (sum <= toleratedLoss) {
                // 如果这个时间段比之前找到的时间段更长，清空结果列表并添加这个时间段
                if (length > maxLength) {
                    validPeriodCount = 0;
                    validPeriods[validPeriodCount][0] = start;
                    validPeriods[validPeriodCount][1] = end;
                    validPeriodCount++;
                    maxLength = length;
                }
                // 如果这个时间段和之前找到的最长时间段一样长，添加这个时间段
                else if (length == maxLength) {
                    validPeriods[validPeriodCount][0] = start;
                    validPeriods[validPeriodCount][1] = end;
                    validPeriodCount++;
                }
            }
        }
    }

    // 如果没有找到满足条件的时间段，输出"NULL"
    if (validPeriodCount == 0) {
        printf("NULL\n");
    }
    // 否则，输出所有满足条件的时间段
    else {
        for (int i = 0; i < validPeriodCount; i++) {
            if (i > 0) printf(" ");
            printf("%d-%d", validPeriods[i][0], validPeriods[i][1]);
```

```
        }
        printf("\n");
    }

    return 0;
}
```

完整用例

用例1

```
1
0 1 2 3 4
```

用例2

```
2
0 0 100 2 2 99 0 2
```

用例3

```
3
1 2 3 4 5 6 7 8 9 10
```

用例4

```
10
10 20 30 40 50 60 70 80 90 100
```

用例5

```
50
10 20 30 40 50 60 70 80 90 100
```

用例6

```
100

```

```
10 20 30 40 50 60 70 80 90 100
```

用例7

```
0
0 0 0 0 0 0 0 0 0 0
```

用例8

```
100
100 100 100 100 100 100 100 100 100 100
```

用例9

```
50
100 100 100 100 100 100 100 100 100 100
```

用例10

```
0
100 100 100 100 100 100 100 100 100 100
```

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
- 题目解析
- C++
- JavaScript
- Java
- Python
- C语言
- 完整用例

用例1
用例2
用例3
用例4
用例5
用例6
用例7
用例8
用例9
用例10