

# 【华为OD机考 统一考试机试C卷】CPU算力分配 (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

现有两组服务器A和B，每组有多个算力不同的CPU，其中  $A[i]$  是 A 组第  $i$  个CPU的运算能力， $B[i]$  是 B组 第  $i$  个CPU的运算能力。

一组服务器的总算力是各CPU的算力之和。

为了让两组服务器的算力相等，允许从每组各选出一个CPU进行一次交换，

求两组服务器中，用于交换的CPU的算力，并且要求从A组服务器中选出的CPU，算力尽可能小。

## 输入描述

第一行输入为 $L1$ 和 $L2$ ，以空格分隔， $L1$ 表示A组服务器中的CPU数量， $L2$ 表示B组服务器中的CPU数量。

第二行输入为A组服务器中各个CPU的算力值，以空格分隔。

第三行输入为B组服务器中各个CPU的算力值，以空格分隔。

$1 \leq L1 \leq 10000$

$1 \leq L2 \leq 10000$

1 ≤ A[i] ≤ 100000  
1 ≤ B[i] ≤ 100000

输出描述

对于每组测试数据，输出两个整数，以空格分隔，依次表示A组选出的CPU算力，B组选出的CPU算力。

要求从A组选出的CPU的算力尽可能小。

备注

- 保证两组服务器的初始总算力不同。
- 答案肯定存在

用例1

输入	2 2 1 1 2 2
输出	1 2
说明	从A组中选出算力为1的CPU，与B组中算力为2的进行交换，使两组服务器的算力都等于3。

用例2

输入	2 2 1 2 2 3
输出	1 2

用例3

输入	1 2 2 1 3
输出	2 3

## 用例4

输入	3 2 1 2 5 2 4
输出	5 4

## 解题思路

这个问题的目标是通过交换A组和B组的服务器，使得两组的总算力尽可能接近。为了达到这个目标，我们需要找到一对服务器，使得交换后两组的总算力差最小。

假设A组的总算力为 $a$ ，B组的总算力为 $b$ ，且 $a > b$ 。我们希望找到A组的一个服务器，其算力为 $x$ ，和B组的一个服务器，其算力为 $y$ ，使得交换后的总算力差为 $(a - x + y) - (b - y + x) = a - b - 2(x - y)$ 尽可能小。也就是说，我们希望 $x - y$ 尽可能接近 $(a - b) / 2$ ，也就是两组总算力差的一半。

因此，我们需要计算两组总算力差的一半，然后在A组的服务器中找到一个算力，使得它减去这个值后的结果在B组的服务器中存在。这样，我们就找到了一对可以交换的服务器，使得交换后两组的总算力尽可能接近。

## C++

```

1  #include <iostream>
2  #include <vector>
3  #include <map>
4  #include <algorithm>
5
6  using namespace std;
7
8  int main() {
9      // 读取A组和B组的服务器数量
10     int serverCountGroupA, serverCountGroupB;
11     cin >> serverCountGroupA >> serverCountGroupB;

```

```

12
13 // 初始化A组的总算力为0
14 int totalPowerGroupA = 0;
15 // 创建数组存储A组的服务器算力
16 vector<int> powerGroupA(serverCountGroupA);
17 // 读取A组的服务器算力, 并计算总算力
18 for (int i = 0; i < serverCountGroupA; i++) {
19     cin >> powerGroupA[i];
20     totalPowerGroupA += powerGroupA[i];
21 }
22
23 // 初始化B组的总算力为0
24 int totalPowerGroupB = 0;
25 // 创建map存储B组的服务器算力和对应的数量
26 map<int, int> powerCountGroupB;
27 // 读取B组的服务器算力, 并计算总算力
28 for (int i = 0; i < serverCountGroupB; i++) {
29     int power;
30     cin >> power;
31     totalPowerGroupB += power;
32     powerCountGroupB[power]++;
33 }
34
35 // 计算两组总算力差的一半
36 int halfDifference = (totalPowerGroupA - totalPowerGroupB) / 2;
37
38 // 对A组的服务器算力进行排序
39 sort(powerGroupA.begin(), powerGroupA.end());
40 // 从小到大遍历A组的服务器
41 for (int powerA : powerGroupA) {
42     // 计算需要在B组中找到的服务器算力
43     int powerB = powerA - halfDifference;
44
45     // 如果B组中存在这样的服务器, 并且数量大于0
46     if (powerCountGroupB.count(powerB) && powerCountGroupB[powerB] > 0) {
47         // 输出A组和B组选出的服务器的算力
48         cout << powerA << " " << powerB << endl;
49         break;
50     }
51 }
52

```

```
53  
54     return 0;  
    }
```

## Java

```
1  import java.util.*;  
2  
3  public class Main {  
4      public static void main(String[] args) {  
5          // 创建扫描器读取输入  
6          Scanner scanner = new Scanner(System.in);  
7  
8          // 读取A组和B组的服务器数量  
9          int serverCountGroupA = scanner.nextInt();  
10         int serverCountGroupB = scanner.nextInt();  
11  
12         // 初始化A组的总算力为0  
13         int totalPowerGroupA = 0;  
14         // 创建数组存储A组的服务器算力  
15         int[] powerGroupA = new int[serverCountGroupA];  
16         // 读取A组的服务器算力, 并计算总算力  
17         for (int i = 0; i < serverCountGroupA; i++) {  
18             powerGroupA[i] = scanner.nextInt();  
19             totalPowerGroupA += powerGroupA[i];  
20         }  
21  
22         // 初始化B组的总算力为0  
23         int totalPowerGroupB = 0;  
24         // 创建HashMap存储B组的服务器算力和对应的数量  
25         HashMap<Integer, Integer> powerCountGroupB = new HashMap<>();  
26         // 读取B组的服务器算力, 并计算总算力  
27         for (int i = 0; i < serverCountGroupB; i++) {  
28             int power = scanner.nextInt();  
29             totalPowerGroupB += power;  
30             powerCountGroupB.put(power, powerCountGroupB.getOrDefault(power, 0) + 1);  
31         }  
32  
33         // 计算两组总算力差的一半, 四舍五入取整  
34         int halfDifference = (int) Math.round((totalPowerGroupA - totalPowerGroupB) / 2.0);  
35     }
```

```

35
36     // 对A组的服务器算力进行排序
37     Arrays.sort(powerGroupA);
38     // 从小到大遍历A组的服务器
39     for (int powerA : powerGroupA) {
40         // 计算需要在B组中找到的服务器算力
41         int powerB = powerA - halfDifference;
42
43         // 如果B组中存在这样的服务器，并且数量大于0
44         if (powerCountGroupB.containsKey(powerB) && powerCountGroupB.get(powerB) > 0) {
45             // 输出A组和B组选出的服务器的算力
46             System.out.println(powerA + " " + powerB);
47             break;
48         }
49     }
50 }
51 }

```

## javaScript

```

1  const readline = require('readline').createInterface({
2      input: process.stdin,
3      output: process.stdout
4  });
5
6  let lines = [];
7
8  // 将输入行存储在数组中
9  readline.on('line', line => {
10     lines.push(line);
11 });
12
13 // 在所有输入行都读取完毕后处理这些行
14 readline.on('close', () => {
15     // 读取A组和B组的服务器数量
16     const [serverCountGroupA, serverCountGroupB] = lines[0].split(' ').map(Number);
17
18     // 初始化A组的总算力为0
19     let totalPowerGroupA = 0;
20     // 创建数组存储A组的服务器算力
21

```

```

41 let powerGroupA = lines[1].split(' ').map(Number);
42 // 计算A组的服务器算力总和
43 totalPowerGroupA = powerGroupA.reduce((a, b) => a + b, 0);
44
45 // 初始化B组的总算力为0
46 let totalPowerGroupB = 0;
47 // 创建map存储B组的服务器算力和对应的数量
48 let powerCountGroupB = new Map();
49 // 读取B组的服务器算力, 并计算总算力
50 const powers = lines[2].split(' ').map(Number);
51 for (let power of powers) {
52     totalPowerGroupB += power;
53     powerCountGroupB.set(power, (powerCountGroupB.get(power) || 0) + 1);
54 }
55
56 // 计算两组总算力差的一半, 四舍五入取整
57 let halfDifference = Math.round((totalPowerGroupA - totalPowerGroupB) / 2);
58
59 // 对A组的服务器算力进行排序
60 powerGroupA.sort((a, b) => a - b);
61 // 从小到大遍历A组的服务器
62 for (let powerA of powerGroupA) {
63     // 计算需要在B组中找到的服务器算力
64     let powerB = powerA - halfDifference;
65
66     // 如果B组中存在这样的服务器, 并且数量大于0
67     if (powerCountGroupB.has(powerB) && powerCountGroupB.get(powerB) > 0) {
68         // 输出A组和B组选出的服务器的算力
69         console.log(powerA + " " + powerB);
70         break;
71     }
72 }
73 }
74 });

```

## Python

```

1 # 导入需要的库
2 import sys
3
4 # 读取A组和B组的服务器数量
5

```

```

5 serverCountGroupA, serverCountGroupB = map(int, sys.stdin.readline().split())
6
7 # 初始化A组的总算力为0
8 totalPowerGroupA = 0
9 # 读取A组的服务器算力，并计算总算力
10 powerGroupA = list(map(int, sys.stdin.readline().split()))
11 totalPowerGroupA = sum(powerGroupA)
12
13 # 初始化B组的总算力为0
14 totalPowerGroupB = 0
15 # 创建字典存储B组的服务器算力和对应的数量
16 powerCountGroupB = {}
17 # 读取B组的服务器算力，并计算总算力
18 powers = list(map(int, sys.stdin.readline().split()))
19 for power in powers:
20     totalPowerGroupB += power
21     powerCountGroupB[power] = powerCountGroupB.get(power, 0) + 1
22
23 # 计算两组总算力差的一半，四舍五入取整
24 halfDifference = round((totalPowerGroupA - totalPowerGroupB) / 2)
25
26 # 对A组的服务器算力进行排序
27 powerGroupA.sort()
28 # 从小到大遍历A组的服务器
29 for powerA in powerGroupA:
30     # 计算需要在B组中找到的服务器算力
31     powerB = powerA - halfDifference
32
33     # 如果B组中存在这样的服务器，并且数量大于0
34     if powerB in powerCountGroupB and powerCountGroupB[powerB] > 0:
35         # 输出A组和B组选出的服务器的算力
36         print(powerA, powerB)
37         break

```

## C语言

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 // 比较函数，用于整数数组排序
5

```



```

5  int compare(const void *a, const void *b) {
6      return (*(int *)a - *(int *)b);
7  }
8
9  int main() {
10     int L1, L2;
11     // 读取A组和B组的服务器数量
12     scanf("%d %d", &L1, &L2);
13
14     int A[L1], B[L2];
15     int totalPowerA = 0, totalPowerB = 0;
16
17     // 读取A组的服务器算力, 并计算总算力
18     for (int i = 0; i < L1; i++) {
19         scanf("%d", &A[i]);
20         totalPowerA += A[i];
21     }
22
23     // 读取B组的服务器算力, 并计算总算力
24     for (int i = 0; i < L2; i++) {
25         scanf("%d", &B[i]);
26         totalPowerB += B[i];
27     }
28
29     // 计算两组总算力差的一半
30     int halfDifference = (totalPowerA - totalPowerB) / 2;
31
32     // 对A组的服务器算力进行排序
33     qsort(A, L1, sizeof(int), compare);
34
35     // 使用哈希表记录B组的服务器算力
36     int hash[100001] = {0};
37     for (int i = 0; i < L2; i++) {
38         hash[B[i]] = 1;
39     }
40
41     // 从小到大遍历A组的服务器
42     for (int i = 0; i < L1; i++) {
43         int powerA = A[i];
44         // 计算需要在B组中找到的服务器算力
45

```

```
46         int powerB = powerA - halfDifference;
47
48         // 如果B组中存在这样的服务器
49         if (powerB >= 1 && powerB <= 100000 && hash[powerB]) {
50             // 输出A组和B组选出的服务器的算力
51             printf("%d %d\n", powerA, powerB);
52             break;
53         }
54     }
55
56     return 0;
}
```

## 完整用例

### 用例1

3 3  
1 2 3  
4 5 6

### 用例2

2 2  
10 20  
10 20

### 用例3

3 3  
1 2 3  
10 20 30

### 用例4

4 4  
5 10 15 20  
30 25 20 15

用例5

2 2  
99999 100000  
99998 99997

用例6

2 2  
1 2  
3 4

用例7

5 5  
17 23 42 50 60  
22 35 37 41 55

用例8

3 3  
20 20 20  
10 15 25

用例9

3 3  
10 15 25  
20 20 20

用例10

3 2  
1 2 5  
2 4

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

备注

用例1

用例2

用例3

用例4

解题思路

C++

Java

javaScript

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

# 机考真题 华为OD



CSDN @算法大师