

【华为OD机考 统一考试机试C卷】员工派遣 (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

某公司部门需要派遣员工去国外做项目。

现在，代号为 x 的国家和代号为 y 的国家分别需要 $cntx$ 名和 $cnty$ 名员工。

部门每个员工有一个员工号 $(1, 2, 3, \dots)$ ，工号连续，从1开始。

部长派遣员工的规则：

- 规则1：从 $[1, k]$ 中选择员工派遣出去
- 规则2：编号为 x 的倍数的员工不能去 x 国，编号为 y 的倍数的员工不能去 y 国。

问题：

找到最小的 k ，使得可以将编号在 $[1, k]$ 中的员工分配给 x 国和 y 国，且满足 x 国和 y 国的需求。

输入描述

四个整数 $x, y, cntx, cnty$ 。

- $2 \leq x < y \leq 30000$

- x 和 y 一定是质数
- $1 \leq cntx, cnty < 10^9$
- $cntx + cnty \leq 10^9$

输出描述

满足条件的最小的k

用例

输入	2 3 3 1
输出	5
说明	输入说明：2 表示国家代号23 表示国家代号33 表示国家2需要3个人1 表示国家3需要1个人

解题思路

题目的问题是：要找到一个最小的k，使得在遵守上述规则的情况下，可以从编号在[1, k]中的员工中选择足够的员工派遣到x国和y国，满足他们的需求。

输入数据的解释如下：

- 2 表示员工的ID如果是2的倍数，就不能派遣到国家X。
- 3 表示员工的ID如果是3的倍数，就不能派遣到国家Y。
- 3 表示国家X需要3个员工。
- 1 表示国家Y需要1个员工。

我们需要找到一个最小的员工ID，使得在[1, ID]的范围内，能够找到至少3个可以派遣到国家X的员工和至少1个可以派遣到国家Y的员工。

在这个例子中，员工ID为5是满足条件的最小值。因为在[1, 5]的范围内，可以找到3个可以派遣到国家X的员工（他们的ID是1, 3, 5），以及1个可以派遣到国家Y的员工（他的ID是1）。所以，输出结果是5。

这段代码使用了二分查找法（Binary Search）来寻找满足特定条件的最小员工ID。二分查找法是一种在有序集合中查找特定元素的搜索算法，通过每次比较中间元素来缩小搜索范围，从而提高查找效率。

这里的特定条件是：在排除不能同时为两个国家工作的员工后，剩余的员工数量能满足两个国家的需求。

1. 二分查找法:

- 初始化搜索范围，下限为两国员工需求之和，上限为一个的大数（例如10亿）。
- 在每次循环中，计算搜索范围的中间值 `midStaffID`。
- 根据 `midStaffID` 判断是否满足条件，然后调整搜索范围。如果满足条件，缩小上限；否则，增大下限。
- 重复上述过程，直到找到满足条件的最小 `midStaffID`。

2. 排除法:

- 计算在1到 `midStaffID` 范围内，不能同时为两个国家工作的员工数量。
- 这些员工的ID是国家X或国家Y的倍数，或者两者的公倍数。
- 计算排除这些员工后，每个国家实际上还需要多少员工。
- 如果剩余的员工数量能满足两个国家的需求，那么 `midStaffID` 就满足条件。

C++

```
1  #include <iostream>
2  #include <algorithm>
3
4  using namespace std;
5
6  int main() {
7      // 定义x, y, cntX, cntY用于存储输入四个数值
8      long x, y, cntX, cntY;
9      // 从标准输入读取这四个值
10     cin >> x >> y >> cntX >> cntY;
11
12     // minID是满足条件的最小员工ID, 初始值设置为两个国家需要的员工总数
13     long minID = cntX + cntY;
14     // maxID是员工ID的可能的最大值
15     long maxID = 1000000000L;
16
17     // 使用二分查找算法
18     while (minID <= maxID) {
19         // 计算中间值midID
20         long midID = minID + (maxID - minID) / 2;
21     }
```

```
22
23 // 计算在[1, midID]范围内不能去国家X的员工数量
24 long excludedX = midID / x;
25 // 计算在[1, midID]范围内不能去国家Y的员工数量
26 long excludedY = midID / y;
27 // 计算在[1, midID]范围内既不能去X国也不能去Y国的员工数量
28 long excludedBoth = midID / (x * y);
29
30 // 计算国家X实际需要的员工数量
31 long neededX = max(0L, cntX - (excludedY - excludedBoth));
32 // 计算国家Y实际需要的员工数量
33 long neededY = max(0L, cntY - (excludedX - excludedBoth));
34 // 计算总共不能使用的员工数量
35 long totalExcluded = midID - excludedX - excludedY + excludedBoth;
36
37 // 判断当前的中间值是否满足条件
38 if (neededX + neededY <= totalExcluded) {
39     // 如果满足条件, 则减小最大的搜索范围
40     maxID = midID - 1;
41 } else {
42     // 如果不满足条件, 则增加最小的搜索范围
43     minID = midID + 1;
44 }
45 }
46
47 // 输出满足条件的最小员工ID
48 cout << minID << endl;
49
50 return 0;
}
```

Java

```
1 import java.util.Scanner;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7         long x, y, cntX, cntY; // 定义静态变量x, y, cntX, cntY
8     }
9 }
```

```
8
9
10     x = sc.nextLong(); // 读取国家X的倍数限制
11     y = sc.nextLong(); // 读取国家Y的倍数限制
12     cntX = sc.nextLong(); // 读取国家X需要的员工数量
13     cntY = sc.nextLong(); // 读取国家Y需要的员工数量
14
15     long minID = cntX + cntY; // 设置员工ID的最小值, 初值为两国需要的员工总数
16     long maxID = 1000000000L; // 设置员工ID的最大值
17
18     // 通过二分查找算法找到满足条件的最小员工ID
19     while (minID <= maxID) {
20         long midID = minID + (maxID - minID) / 2; // 计算中间值midID
21
22         long excludedX = midID / x; // 计算在[1, midID]范围内不能去X国的员工数
23         long excludedY = midID / y; // 计算在[1, midID]范围内不能去Y国的员工数
24         long excludedBoth = midID / (x * y); // 计算在[1, midID]范围内同时不能去X国和Y国的员工数
25
26         long neededX = Math.max(0, cntX - (excludedY - excludedBoth)); // 计算X国实际需要的员工数
27         long neededY = Math.max(0, cntY - (excludedX - excludedBoth)); // 计算Y国实际需要的员工数
28         long totalExcluded = midID - excludedX - excludedY + excludedBoth; // 计算总共不能使用的员工数
29
30         // 判断当前midID是否满足条件
31         if (neededX + neededY <= totalExcluded) {
32             maxID = midID - 1; // 如果满足条件, 降低最大ID的搜索范围
33         } else {
34             minID = midID + 1; // 如果不满足条件, 提高最小ID的搜索范围
35         }
36     }
37
38     System.out.println(minID); // 输出满足条件的最小员工ID
39     sc.close(); // 关闭扫描器
40 }
```

JavaScript

```
1 // 引入readline模块, 用于从标准输入读取数据
2 const readline = require('readline');
3
4 // 创建readline接口实例
5
```

```
5  const rl = readline.createInterface({
6      input: process.stdin, // 标准输入作为输入源
7      output: process.stdout // 标准输出作为输出源
8  });
9
10 // 当从标准输入中读取到一行数据时触发
11 rl.on('line', (line) => {
12     // 将读取到的行分割成数组, 并将其元素转换为数字
13     const [x, y, cntX, cntY] = line.split(' ').map(Number);
14
15     // minID是满足条件的最小员工ID, 初始值设置为两个国家需要的员工总数
16     let minID = cntX + cntY;
17     // maxID是员工ID的可能的最大值
18     let maxID = 1000000000;
19
20     // 使用二分查找算法
21     while (minID <= maxID) {
22         // 计算中间值midID
23         const midID = minID + Math.floor((maxID - minID) / 2);
24
25         // 计算在[1, midID]范围内不能去国家X的员工数量
26         const excludedX = Math.floor(midID / x);
27         // 计算在[1, midID]范围内不能去国家Y的员工数量
28         const excludedY = Math.floor(midID / y);
29         // 计算在[1, midID]范围内既不能去X国也不能去Y国的员工数量
30         const excludedBoth = Math.floor(midID / (x * y));
31
32         // 计算国家X实际需要的员工数量
33         const neededX = Math.max(0, cntX - (excludedY - excludedBoth));
34         // 计算国家Y实际需要的员工数量
35         const neededY = Math.max(0, cntY - (excludedX - excludedBoth));
36         // 计算总共不能使用的员工数量
37         const totalExcluded = midID - excludedX - excludedY + excludedBoth;
38
39         // 判断当前的中间值是否满足条件
40         if (neededX + neededY <= totalExcluded) {
41             // 如果满足条件, 则减小最大的搜索范围
42             maxID = midID - 1;
43         } else {
44             // 如果不满足条件, 则增加最小的搜索范围
45         }
```

```
46         minID = midID + 1;
47     }
48 }
49
50 // 输出满足条件的最小员工ID
51 console.log(minID);
});
```

Python

```
1  # 使用map函数和int函数从标准输入读取四个整数
2  x, y, cntX, cntY = map(int, input().split())
3
4  # minID是满足条件的最小员工ID, 初始值设置为两个国家需要的员工总数
5  minID = cntX + cntY
6  # maxID是员工ID的可能的最大值
7  maxID = 1000000000
8
9  # 使用二分查找算法
10 while minID <= maxID:
11     # 计算中间值midID
12     midID = minID + (maxID - minID) // 2
13
14     # 计算在[1, midID]范围内不能去国家X的员工数量
15     excludedX = midID // x
16     # 计算在[1, midID]范围内不能去国家Y的员工数量
17     excludedY = midID // y
18     # 计算在[1, midID]范围内既不能去X国也不能去Y国的员工数量
19     excludedBoth = midID // (x * y)
20
21     # 计算国家X实际需要的员工数量
22     neededX = max(0, cntX - (excludedY - excludedBoth))
23     # 计算国家Y实际需要的员工数量
24     neededY = max(0, cntY - (excludedX - excludedBoth))
25     # 计算总共不能使用的员工数量
26     totalExcluded = midID - excludedX - excludedY + excludedBoth
27
28     # 判断当前的中间值是否满足条件
29     if neededX + neededY <= totalExcluded:
30         # 如果满足条件, 则减小最大的搜索范围
31         maxID = midID
```



```
31     maxID = midID - 1
32 else:
33     # 如果不满足条件, 则增加最小的搜索范围
34     minID = midID + 1
35
36 # 输出满足条件的最小员工ID
37 print(minID)
```

C语言

```
1  #include <stdio.h>
2
3  // 计算最大值的函数, 用于确定实际需要的员工数量
4  long max(long a, long b) {
5      return a > b ? a : b;
6  }
7
8  int main() {
9      long x, y, cntX, cntY;
10     // 从标准输入读取x, y, cntX, cntY
11     scanf("%ld %ld %ld %ld", &x, &y, &cntX, &cntY);
12
13     // 初始化最小和最大员工编号
14     long minID = cntX + cntY; // 最小编号为两国需要的员工总数
15     long maxID = 1000000000L; // 假定的最大员工编号
16
17     // 使用二分查找
18     while (minID <= maxID) {
19         long midID = minID + (maxID - minID) / 2; // 计算中间编号
20
21         // 计算不能去x国和y国的员工数量
22         long excludedX = midID / x;
23         long excludedY = midID / y;
24         long excludedBoth = midID / (x * y);
25
26         // 计算x国和y国实际需要的员工数量
27         long neededX = max(0, cntX - (excludedY - excludedBoth));
28         long neededY = max(0, cntY - (excludedX - excludedBoth));
29
30         // 计算总共可用的员工数量
31         long total = neededX + neededY - excludedBoth;
```

```
31     long totalAvailable = midID - excludedX - excludedY + excludedBoth;
32
33     // 根据可用员工数量更新minID和maxID
34     if (neededX + neededY <= totalAvailable) {
35         maxID = midID - 1;
36     } else {
37         minID = midID + 1;
38     }
39 }
40
41 // 输出满足条件的最小员工编号
42 printf("%ld\n", minID);
43
44 return 0;
45 }
```

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](#)

[解题思路](#)

[C++](#)

[Java](#)

[javaScript](#)

[Python](#)

[C语言](#)

机考真题 华为OD



CSDN @算法大师