【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,…），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 ≤ x < y ≤ 30000

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

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

using namespace std;

int main() {
    // 定义x, y, cntX, cntY用于存储输入的四个数值
    long x, y, cntX, cntY;
    // 从标准输入读取这四个值
    cin >> x >> y >> cntX >> cntY;

    // 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
    long minID = cntX + cntY;
    // maxID是员工ID的可能的最大值
    long maxID = 1000000000L;

    // 使用二分查找算法
    while (minID <= maxID) {
        // 计算中间值midID
        long midID = minID + (maxID - minID) / 2;

```

```
        // 计算在[1, midID]范围内不能去国家X的员工数量
        long excludedX = midID / x;
        // 计算在[1, midID]范围内不能去国家Y的员工数量
        long excludedY = midID / y;
        // 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
        long excludedBoth = midID / (x * y);

        // 计算国家X实际需要的员工数量
        long neededX = max(0L, cntX - (excludedY - excludedBoth));
        // 计算国家Y实际需要的员工数量
        long neededY = max(0L, cntY - (excludedX - excludedBoth));
        // 计算总共不能使用的员工数量
        long totalExcluded = midID - excludedX - excludedY + excludedBoth;

        // 判断当前的中间值是否满足条件
        if (neededX + neededY <= totalExcluded) {
            // 如果满足条件，则减小最大的搜索范围
            maxID = midID - 1;
        } else {
            // 如果不满足条件，则增加最小的搜索范围
            minID = midID + 1;
        }
    }

    // 输出满足条件的最小员工ID
    cout << minID << endl;

    return 0;
}
```

Java

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        long x, y, cntX, cntY; // 定义静态变量x, y, cntX, cntY
```

```
        x = sc.nextLong(); // 读取国家X的倍数限制
        y = sc.nextLong(); // 读取国家Y的倍数限制
        cntX = sc.nextLong(); // 读取国家X需要的员工数量
        cntY = sc.nextLong(); // 读取国家Y需要的员工数量

        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
        long maxID = 1000000000L; // 设置员工ID的最大值

        // 通过二分查找算法找到满足条件的最小员工ID
        while (minID <= maxID) {
            long midID = minID + (maxID - minID) / 2; // 计算中间值midID

            long excludedX = midID / x; // 计算在[1, midID]范围内不能去X国的员工数
            long excludedY = midID / y; // 计算在[1, midID]范围内不能去Y国的员工数
            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

            long neededX = Math.max(0, cntX - (excludedY - excludedBoth)); // 计算X国实际需要的员工数
            long neededY = Math.max(0, cntY - (excludedX - excludedBoth)); // 计算Y国实际需要的员工数
            long totalExcluded = midID - excludedX - excludedY + excludedBoth; // 计算总共不能使用的员工数

            // 判断当前midID是否满足条件
            if (neededX + neededY <= totalExcluded) {
                maxID = midID - 1; // 如果满足条件，降低最大ID的搜索范围
            } else {
                minID = midID + 1; // 如果不满足条件，提高最小ID的搜索范围
            }
        }

        System.out.println(minID); // 输出满足条件的最小员工ID
        sc.close(); // 关闭扫描器
    }
}
```

javaScript

```
// 引入readline模块，用于从标准输入读取数据
const readline = require('readline');

// 创建readline接口实例
```

```
const rl = readline.createInterface({
    input: process.stdin,  // 标准输入作为输入源
    output: process.stdout // 标准输出作为输出源
});

// 当从标准输入中读取到一行数据时触发
rl.on('line', (line) => {
    // 将读取到的行分割成数组，并将其元素转换为数字
    const [x, y, cntX, cntY] = line.split(' ').map(Number);

    // 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
    let minID = cntX + cntY;
    // maxID是员工ID的可能的最大值
    let maxID = 1000000000;

    // 使用二分查找算法
    while (minID <= maxID) {
        // 计算中间值midID
        const midID = minID + Math.floor((maxID - minID) / 2);

        // 计算在[1, midID]范围内不能去国家X的员工数量
        const excludedX = Math.floor(midID / x);
        // 计算在[1, midID]范围内不能去国家Y的员工数量
        const excludedY = Math.floor(midID / y);
        // 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
        const excludedBoth = Math.floor(midID / (x * y));

        // 计算国家X实际需要的员工数量
        const neededX = Math.max(0, cntX - (excludedY - excludedBoth));
        // 计算国家Y实际需要的员工数量
        const neededY = Math.max(0, cntY - (excludedX - excludedBoth));
        // 计算总共不能使用的员工数量
        const totalExcluded = midID - excludedX - excludedY + excludedBoth;

        // 判断当前的中间值是否满足条件
        if (neededX + neededY <= totalExcluded) {
            // 如果满足条件，则减小最大的搜索范围
            maxID = midID - 1;
        } else {
            // 如果不满足条件，则增加最小的搜索范围
```

```
            minID = midID + 1;
        }
    }

    // 输出满足条件的最小员工ID
    console.log(minID);
});
```

Python

```python
# 使用map函数和int函数从标准输入读取四个整数
x, y, cntX, cntY = map(int, input().split())

# 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
minID = cntX + cntY
# maxID是员工ID的可能的最大值
maxID = 1000000000

# 使用二分查找算法
while minID <= maxID:
    # 计算中间值midID
    midID = minID + (maxID - minID) // 2

    # 计算在[1, midID]范围内不能去国家X的员工数量
    excludedX = midID // x
    # 计算在[1, midID]范围内不能去国家Y的员工数量
    excludedY = midID // y
    # 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
    excludedBoth = midID // (x * y)

    # 计算国家X实际需要的员工数量
    neededX = max(0, cntX - (excludedY - excludedBoth))
    # 计算国家Y实际需要的员工数量
    neededY = max(0, cntY - (excludedX - excludedBoth))
    # 计算总共不能使用的员工数量
    totalExcluded = midID - excludedX - excludedY + excludedBoth

    # 判断当前的中间值是否满足条件
    if neededX + neededY <= totalExcluded:
        # 如果满足条件，则减小最大的搜索范围
```

```
        maxID = midID - 1
    else:
        # 如果不满足条件，则增加最小的搜索范围
        minID = midID + 1

# 输出满足条件的最小员工ID
print(minID)
```

C语言

```c
#include <stdio.h>

// 计算最大值的函数，用于确定实际需要的员工数量
long max(long a, long b) {
    return a > b ? a : b;
}

int main() {
    long x, y, cntX, cntY;
    // 从标准输入读取x, y, cntX, cntY
    scanf("%ld %ld %ld %ld", &x, &y, &cntX, &cntY);

    // 初始化最小和最大员工编号
    long minID = cntX + cntY; // 最小编号为两国需要的员工总数
    long maxID = 1000000000L; // 假定的最大员工编号

    // 使用二分查找
    while (minID <= maxID) {
        long midID = minID + (maxID - minID) / 2; // 计算中间编号

        // 计算不能去x国和y国的员工数量
        long excludedX = midID / x;
        long excludedY = midID / y;
        long excludedBoth = midID / (x * y);

        // 计算x国和y国实际需要的员工数量
        long neededX = max(0, cntX - (excludedY - excludedBoth));
        long neededY = max(0, cntY - (excludedX - excludedBoth));

        // 计算总共可用的员工数量
```

```
        long totalAvailable = midID - excludedX - excludedY + excludedBoth;

        // 根据可用员工数量更新minID和maxID
        if (neededX + neededY <= totalAvailable) {
            maxID = midID - 1;
        } else {
            minID = midID + 1;
        }
    }

    // 输出满足条件的最小员工编号
    printf("%ld\n", minID);

    return 0;
}
```

文章目录


机考真题
华为OD
No No No
CSDN @算法大师