

=====

文件夹: class062\_GCDCalculationAndLCM

=====

[Markdown 文件]

=====

文件: ADDITIONAL\_PROBLEMS.md

=====

# Class041 - 额外 GCD 和 LCM 相关问题

## ## 概述

本文档总结了从各大算法平台收集的额外 GCD 和 LCM 相关问题，包含详细的题目描述、解题思路、复杂度分析以及 Java、C++、Python 三种语言的实现。

## ## 额外题目列表

### #### 1. SPOJ LCMSUM - LCM Sum

\*\*题目来源\*\*: [SPOJ LCMSUM] (<https://www.spoj.com/problems/LCMSUM/>)

\*\*问题描述\*\*: 给定 n, 计算  $\sum_{i=1 \text{ to } n} \text{lcm}(i, n)$

\*\*解题思路\*\*:

利用数学公式进行优化。我们知道:

$$\sum_{i=1 \text{ to } n} \text{lcm}(i, n) = \sum_{i=1 \text{ to } n} (i * n) / \text{gcd}(i, n) = n * \sum_{i=1 \text{ to } n} i / \text{gcd}(i, n)$$

我们可以将这个和式按 gcd 值分组:

$$\sum_{d|n} \sum_{i=1 \text{ to } n, \text{gcd}(i, n)=d} i / d$$

对于  $\text{gcd}(i, n)=d$  的情况, 设  $i=d*j, n=d*k$ , 则  $\text{gcd}(j, k)=1$

所以  $\sum_{i=1 \text{ to } n, \text{gcd}(i, n)=d} i = d * \sum_{j=1 \text{ to } k, \text{gcd}(j, k)=1} j$

$$\sum_{j=1 \text{ to } k, \text{gcd}(j, k)=1} j = k * \phi(k) / 2 \quad (\text{当 } k>1 \text{ 时})$$

其中  $\phi$  是欧拉函数

因此,  $\sum_{i=1 \text{ to } n} \text{lcm}(i, n) = n * \sum_{d|n} \phi(n/d) * (n/d) / 2 = (n/2) * \sum_{d|n} \phi(d) * d + n$   
(当  $d=n$  时需要特殊处理)

\*\*时间复杂度\*\*:  $O(\sqrt{n})$

\*\*空间复杂度\*\*:  $O(1)$

\*\*是否最优解\*\*: 是, 这是解决该问题的最优方法。

### ### 2. SPOJ GCDEX – GCD Extreme

\*\*题目来源\*\*: [SPOJ GCDEX] (<https://www.spoj.com/problems/GCDEX/>)

\*\*问题描述\*\*: 计算  $G(n) = \sum_{i=1}^n \sum_{j=i+1}^n \gcd(i, j)$

\*\*解题思路\*\*: 使用欧拉函数优化计算

\*\*时间复杂度\*\*:  $O(n \log n)$

\*\*空间复杂度\*\*:  $O(n)$

\*\*是否最优解\*\*: 是, 这是解决该问题的最优方法。

### ### 3. UVa 10892 – LCM Cardinality

\*\*题目来源\*\*: [UVa 10892] ([https://onlinejudge.org/index.php?option=com\\_onlinejudge&Itemid=8&page=show\\_problem&problem=1833](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1833))

\*\*问题描述\*\*: 给定一个正整数  $n$ , 找出有多少对不同的整数对  $(a, b)$ , 使得  $\text{lcm}(a, b) = n$ 。

\*\*解题思路\*\*: 枚举  $n$  的所有因子, 对于每个因子  $d$ , 如果  $\gcd(d, n/d) = 1$ , 则  $(d, n/d)$  是一对解。

\*\*时间复杂度\*\*:  $O(\sqrt{n})$

\*\*空间复杂度\*\*:  $O(1)$

\*\*是否最优解\*\*: 是, 这是解决该问题的最优方法。

### ### 4. POJ 2429 – GCD & LCM Inverse

\*\*题目来源\*\*: [POJ 2429] (<http://poj.org/problem?id=2429>)

\*\*问题描述\*\*: 给定两个正整数  $a$  和  $b$  的最大公约数和最小公倍数, 反过来求这两个数, 要求这两个数的和最小。

\*\*解题思路\*\*:

设  $\gcd$  为最大公约数,  $\text{lcm}$  为最小公倍数, 则  $a*b = \gcd*\text{lcm}$ 。设  $a = \gcd*x$ ,  $b = \gcd*y$ , 则  $x*y = \text{lcm}/\gcd$ , 且  $\gcd(x, y) = 1$ 。问题转化为找到两个互质的数  $x$  和  $y$ , 使得  $x*y = \text{lcm}/\gcd$ , 并且  $x+y$  最小。

\*\*时间复杂度\*\*:  $O(\sqrt{(\text{lcm}/\gcd)})$

\*\*空间复杂度\*\*:  $O(1)$

\*\*是否最优解\*\*: 是, 这是解决该问题的最优方法。

### ### 5. Codeforces 1034A – Enlarge GCD

**\*\*题目来源\*\*:** [Codeforces 1034A] (<https://codeforces.com/problemset/problem/1034/A>)

**\*\*问题描述\*\*:** 给定  $n$  个正整数，通过删除最少的数来增大这些数的最大公约数。返回需要删除的最少数字个数，如果无法增大 GCD 则返回-1。

**\*\*解题思路\*\*:**

首先计算所有数的 GCD，然后将所有数除以这个 GCD，问题转化为找到一个大于 1 的因子，使得尽可能多的数是这个因子的倍数。枚举所有质数，统计是其倍数的数的个数，答案就是  $n$  减去最大个数。

**\*\*时间复杂度\*\*:**  $O(n \log(\max\_value) + \max\_value \log(\log(\max\_value)))$

**\*\*空间复杂度\*\*:**  $O(\max\_value)$

**\*\*是否最优解\*\*:** 是，这是解决该问题的最优方法。

#### #### 6. AtCoder ABC150D – Semi Common Multiple

**\*\*题目描述\*\*:** 给定一个由偶数组成的数组  $a$  和一个整数  $M$ ，求  $[1, M]$  中有多少个数  $X$  满足  $X = a_i * (p+0.5)$  对所有  $i$  成立，其中  $p$  是非负整数

**\*\*来源\*\*:** [AtCoder ABC150D] ([https://atcoder.jp/contests/abc150/tasks/abc150\\_d](https://atcoder.jp/contests/abc150/tasks/abc150_d))

**\*\*解题思路\*\*:**

1. 将  $X = a_i * (p+0.5)$  转换为  $2X = a_i * (2p+1)$
2. 这意味着  $2X$  必须是每个  $a_i$  的奇数倍
3. 计算数组中每个  $a_i$  除以 2 后的 LCM，记为  $L$
4. 然后需要计算有多少个  $X \leq M$  满足  $X = kL$ ，其中  $k$  是奇数

**\*\*时间复杂度\*\*:**  $O(n \log \max(a_i))$

**\*\*空间复杂度\*\*:**  $O(1)$

#### #### 7. 三元组 GCD 和 LCM 计数问题

**\*\*题目描述\*\*:** 给定  $G$  和  $L$ ，计算满足  $\gcd(x, y, z) = G$  且  $\text{lcm}(x, y, z) = L$  的三元组  $(x, y, z)$  的个数

**\*\*来源\*\*:** 数论经典问题

**\*\*解题思路\*\*:**

1. 首先检查  $L$  是否能被  $G$  整除，如果不能则没有解
2. 对  $L/G$  进行质因数分解
3. 对于每个质因子  $p$ ，分析其在  $x, y, z$  中的指数分布
4. 对于每个质因子  $p$ ，要求：
  - 至少有一个数的指数等于  $g$  ( $G$  中  $p$  的指数)

- 至少有一个数的指数等于 1 (L 中 p 的指数)
  - 其他数的指数在 [g, 1] 范围内
5. 使用组合数学计算每个质因子对应的可能性，最后相乘

**\*\*时间复杂度\*\*:**  $O(\sqrt{L/G})$  用于质因数分解

**\*\*空间复杂度\*\*:**  $O(\log(L/G))$  用于存储质因数分解结果

### ### 8. HackerRank GCD Product

**\*\*题目来源\*\*:** [HackerRank GCD Product] (<https://www.hackerrank.com/challenges/gcd-product/problem>)

**\*\*问题描述\*\*:** 给定 N 和 M，计算  $\prod_{i=1}^N \prod_{j=1}^M \gcd(i, j) \bmod (10^{9+7})$

**\*\*解题思路\*\*:**

对于每个质数 p，计算它在结果中的指数。对于质数 p，它在  $\gcd(i, j)$  中的指数等于  $\min(vp(i), vp(j))$ ，其中  $vp(x)$  表示 x 中质因子 p 的指数。

我们可以枚举所有质数 p，计算  $\sum_{i=1}^N \sum_{j=1}^M \min(vp(i), vp(j))$ 。

为了优化计算，我们可以使用以下方法：

对于每个质数 p，计算有多少个数 i 满足  $vp(i)=k$ ，记为  $\text{count\_p}(k)$ 。

然后计算  $\sum_{k=1}^{\max} \sum_{l=1}^{\max} \min(k, l) * \text{count\_p}(k) * \text{count\_p}(l)$ 。

**\*\*时间复杂度\*\*:**  $O(N \log(\log(N)) + M \log(\log(M)))$

**\*\*空间复杂度\*\*:**  $O(N + M)$

**\*\*是否最优解\*\*:** 是，这是解决该问题的最优方法。

## ## 📈 复杂度分析汇总

题目	时间复杂度	空间复杂度	是否最优解
SPOJ LCMSUM	$O(\sqrt{n})$	$O(1)$	是
SPOJ GCDEX	$O(n \log n)$	$O(n)$	是
UVa 10892	$O(\sqrt{n})$	$O(1)$	是
POJ 2429	$O(\sqrt{1cm/gcd})$	$O(1)$	是
Codeforces 1034A	$O(n \log(\max_value) + \max_value * \log(\log(\max_value)))$	$O(\max_value)$	是
AtCoder ABC150D	$O(n \log \max(a_i))$	$O(1)$	是
三元组 GCD 和 LCM 计数	$O(\sqrt{L/G})$	$O(\log(L/G))$	是
HackerRank GCD Product	$O(N \log(\log(N)) + M \log(\log(M)))$	$O(N + M)$	是

## ## 📁 文件列表

本目录包含以下实现文件：

- `AdditionalGcdLcmProblems.java` - 额外 GCD/LCM 问题集合 (Java 版本)
- `AdditionalGcdLcmProblems.cpp` - 额外 GCD/LCM 问题集合 (C++ 版本)

- `AdditionalGcdLcmProblems.py` - 额外 GCD/LCM 问题集合 (Python 版本)
- `ADDITIONAL\_PROBLEMS.md` - 本说明文档

每个文件都包含详细的注释说明、复杂度分析和测试用例，确保代码的正确性和可读性。

=====

文件: README.md

=====

# Class041 - GCD 和 LCM 算法详解与扩展

## ## 📚 概述

本模块主要讲解最大公约数 (GCD) 和最小公倍数 (LCM) 的算法实现，以及它们在各类算法问题中的应用。GCD 和 LCM 是数论中的基础概念，在算法竞赛和实际工程中都有广泛应用。

## ## 🎯 核心知识点

### ### 1. 欧几里得算法（辗转相除法）

欧几里得算法是计算两个数最大公约数的经典算法，基于以下数学原理：

```

$$\gcd(a, b) = \gcd(b, a \% b)$$

```

\*\*时间复杂度\*\*:  $O(\log(\min(a, b)))$

\*\*空间复杂度\*\*:  $O(\log(\min(a, b)))$  (递归) 或  $O(1)$  (迭代)

### ### 2. GCD 与 LCM 的关系

```

$$\gcd(a, b) * \text{lcm}(a, b) = |a * b|$$

```

因此，可以通过 GCD 计算 LCM：

```

$$\text{lcm}(a, b) = |a * b| / \gcd(a, b)$$

```

### ### 3. 扩展欧几里得算法

用于求解线性丢番图方程  $ax + by = \gcd(a, b)$  的整数解。

## ## 📈 算法实现

### ### Java 实现

```
```java
// 计算最大公约数（递归）
public static long gcd(long a, long b) {
    return b == 0 ? a : gcd(b, a % b);
}
```

```
// 计算最小公倍数
public static long lcm(long a, long b) {
    return a / gcd(a, b) * b;
}
```
```

### ### C++实现

```
```cpp
// 计算最大公约数（递归）
static long long gcd(long long a, long long b) {
    return b == 0 ? a : gcd(b, a % b);
}
```

```
// 计算最小公倍数
static long long lcm(long long a, long long b) {
    return a / gcd(a, b) * b;
}
```
```

### ### Python 实现

```
```python
import math

# Python 3.5+ 内置函数
math.gcd(a, b) # 计算最大公约数
math.lcm(a, b) # 计算最小公倍数 (Python 3.9+)
```
```
```

## ## 💭 经典问题与解法

#### #### 1. 基础 GCD/LCM 计算

\*\*问题\*\*: 给定两个正整数，计算它们的最大公约数和最小公倍数

\*\*解法\*\*: 直接应用欧几里得算法和公式

#### #### 2. 第 N 个神奇数字 (LeetCode 878)

\*\*问题\*\*: 一个正整数如果能被  $a$  或  $b$  整除，那么它是神奇的。给定三个整数  $n$ ,  $a$ ,  $b$ , 返回第  $n$  个神奇的数字。

\*\*解法\*\*: 二分查找 + 容斥原理

\*\*关键公式\*\*: 在  $[1, x]$  范围内神奇数字的个数 =  $x/a + x/b - x/\text{lcm}(a, b)$

#### #### 3. 丑数 III (LeetCode 1201)

\*\*问题\*\*: 编写一个程序，找出第  $n$  个丑数，丑数是可以被  $a$  或  $b$  或  $c$  整除的正整数。

\*\*解法\*\*: 二分查找 + 容斥原理 (三元组版本)

\*\*关键公式\*\*: 在  $[1, x]$  范围内丑数的个数 =  $x/a + x/b + x/c - x/\text{lcm}(a, b) - x/\text{lcm}(a, c) - x/\text{lcm}(b, c) + x/\text{lcm}(a, b, c)$

#### #### 4. 字符串的最大公因子 (LeetCode 1071)

\*\*问题\*\*: 对于字符串  $s$  和  $t$ ，只有在  $s=t+t+t+\dots+t$  时，才认为  $t$  能除尽  $s$ 。给定两个字符串  $\text{str1}$  和  $\text{str2}$ ，返回最长字符串  $x$ ，使得  $x$  能除尽  $\text{str1}$  和  $\text{str2}$ 。

\*\*解法\*\*: 利用字符串连接的性质 + GCD

\*\*关键洞察\*\*: 如果存在公因子字符串，则  $\text{str1}+\text{str2} == \text{str2}+\text{str1}$ ，且最大公因子字符串长度为  $\text{gcd}(\text{len}(\text{str1}), \text{len}(\text{str2}))$

#### #### 5. 最大公因数等于 K 的子数组数目 (LeetCode 2447)

\*\*问题\*\*: 给定一个数组和一个正整数  $k$ ，返回最大公因数等于  $k$  的子数组数目。

\*\*解法\*\*: 枚举所有子数组 + GCD 计算 + 优化剪枝

\*\*优化点\*\*:

1. 如果当前元素不能被  $k$  整除，跳过该子数组
2. 如果当前 GCD 小于  $k$ ，不可能再变大，提前终止

#### #### 6. 最小公倍数为 K 的子数组数目 (LeetCode 2470)

\*\*问题\*\*: 给定一个数组和一个正整数  $k$ ，返回最小公倍数等于  $k$  的子数组数目。

\*\*解法\*\*: 枚举所有子数组 + LCM 计算 + 优化剪枝

\*\*优化点\*\*:

1. 如果当前元素不能整除  $k$ ，跳过该子数组
2. 如果当前 LCM 大于  $k$ ，不可能再变小，提前终止

#### #### 7. 链表中插入最大公约数 (LeetCode 2807)

**\*\*问题\*\*:** 给定一个链表，在每对相邻节点之间插入一个值为它们最大公约数的新节点。

**\*\*解法\*\*:** 遍历链表，对每对相邻节点，计算它们的最大公约数并插入新节点。

**\*\*关键点\*\*:**

1. 遍历链表时注意处理指针
2. 插入新节点后要正确更新指针

#### #### 8. 数组中最小数和最大数的最大公约数 (LeetCode 1979)

**\*\*问题\*\*:** 给定一个整数数组  $\text{nums}$ ，返回数组中最小数和最大数的最大公约数。

**\*\*解法\*\*:** 首先找到数组中的最小值和最大值，然后计算它们的最大公约数。

**\*\*关键点\*\*:**

1. 遍历数组找到最小值和最大值
2. 应用欧几里得算法计算 GCD

#### #### 9. GCD Extreme (SPOJ GCDEX)

**\*\*问题\*\*:** 计算  $G(n) = \sum_{i=1}^n \sum_{j=i+1}^n \gcd(i, j)$

**\*\*解法\*\*:** 使用欧拉函数优化计算

**\*\*关键点\*\*:**

1. 利用欧拉函数的性质进行优化
2. 预处理欧拉函数值

#### #### 10. LCM Cardinality (UVa 10892)

**\*\*问题\*\*:** 给定一个正整数  $n$ ，找出有多少对不同的整数对  $(a, b)$ ，使得  $\text{lcm}(a, b) = n$ 。

**\*\*解法\*\*:** 枚举  $n$  的所有因子，对于每个因子  $d$ ，如果  $\gcd(d, n/d) = 1$ ，则  $(d, n/d)$  是一对解。

**\*\*关键点\*\*:**

1. 找到  $n$  的所有因子
2. 检查因子对是否互质

#### #### 11. GCD & LCM Inverse (POJ 2429)

**\*\*问题\*\*:** 给定两个正整数  $a$  和  $b$  的最大公约数和最小公倍数，反过来求这两个数，要求这两个数的和最小。

**\*\*解法\*\*:** 设  $\text{gcd}$  为最大公约数， $\text{lcm}$  为最小公倍数，则  $a*b = \text{gcd}*\text{lcm}$ 。设  $a = \text{gcd}*x$ ,  $b = \text{gcd}*y$ ，则  $x*y = \text{lcm}/\text{gcd}$ ，且  $\gcd(x, y) = 1$ 。问题转化为找到两个互质的数  $x$  和  $y$ ，使得  $x*y = \text{lcm}/\text{gcd}$ ，并且  $x+y$  最小。

**\*\*关键点\*\*:**

1. 将问题转化为寻找互质因子对
2. 枚举所有可能的因子对并找到和最小的

#### #### 12. Enlarge GCD (Codeforces 1034A)

**\*\*问题\*\*:** 给定  $n$  个正整数，通过删除最少的数来增大这些数的最大公约数。返回需要删除的最少数字个数，如果无法增大 GCD 则返回-1。

**\*\*解法\*\*:** 首先计算所有数的 GCD，然后将所有数除以这个 GCD，问题转化为找到一个大于 1 的因子，使得尽可能多的数是这个因子的倍数。枚举所有质数，统计是其倍数的数的个数，答案就是  $n$  减去最大个数。

**\*\*关键点\*\*:**

1. 线性筛法预处理质数
2. 统计每个质数的倍数个数

#### ### 13. GCD Product (HackerRank)

**\*\*问题\*\*:** 给定  $N$  和  $M$ ，计算  $\prod_{i=1}^N \prod_{j=1}^M \gcd(i, j) \bmod (10^{9+7})$

**\*\*解法\*\*:** 对于每个质数  $p$ ，计算它在结果中的指数。对于质数  $p$ ，它在  $\gcd(i, j)$  中的指数等于  $\min(v_p(i), v_p(j))$ ，其中  $v_p(x)$  表示  $x$  中质因子  $p$  的指数。

**\*\*关键点\*\*:**

1. 质因子分解
2. 使用费马小定理进行模运算

#### ### 14. LCM Sum (SPOJ LCMSUM)

**\*\*问题\*\*:** 给定  $n$ ，计算  $\sum_{i=1}^n \text{lcm}(i, n)$

**\*\*解法\*\*:** 利用数学公式进行优化。我们知道： $\sum_{i=1}^n \text{lcm}(i, n) = \sum_{i=1}^n (i * n) / \gcd(i, n) = n * \sum_{i=1}^n i / \gcd(i, n)$ 。可以将这个和式按 gcd 值分组，并利用欧拉函数进行计算。

**\*\*关键点\*\*:**

1. 按 gcd 值分组求和
2. 利用欧拉函数的性质

## ## 复杂度分析

算法	时间复杂度	空间复杂度	说明
欧几里得算法	$O(\log(\min(a, b)))$	$O(\log(\min(a, b)))$	递归实现
欧几里得算法	$O(\log(\min(a, b)))$	$O(1)$	迭代实现
二分查找第 $N$ 个神奇数字	$O(\log(n * \min(a, b)))$	$O(1)$	
子数组 GCD 计数	$O(n^2 * \log(\max))$	$O(1)$	优化后实际更快
子数组 LCM 计数	$O(n^2 * \log(\max))$	$O(1)$	优化后实际更快
GCD Extreme	$O(n \log n)$	$O(n)$	使用欧拉函数优化
LCM Cardinality	$O(\sqrt{n})$	$O(1)$	枚举因子
GCD & LCM Inverse	$O(\sqrt{(\text{lcm}/\gcd)})$	$O(1)$	枚举因子对
Enlarge GCD	$O(n * \log(\max\_value) + \max\_value * \log(\log(\max\_value)))$	$O(\max\_value)$	线性筛法
GCD Product	$O(N * \log(\log(N)) + M * \log(\log(M)))$	$O(N + M)$	质因子分解
LCM Sum	$O(\sqrt{n})$	$O(1)$	按因子分组

## ## 🔧 工程化考量

### #### 1. 溢出处理

- 使用 long 类型避免整数溢出
- 在计算 LCM 时先除后乘: `a / gcd(a, b) \* b`

### #### 2. 边界情况

- 处理其中一个数为 0 的情况
- 处理负数输入 (取绝对值)

### #### 3. 性能优化

- 提前终止条件判断
- 利用 GCD 和 LCM 的单调性进行剪枝
- 使用欧拉函数等数学工具优化计算

### #### 4. 代码质量

- 详细的注释说明题目来源、解题思路、复杂度分析
- 完整的测试用例覆盖各种情况
- 清晰的代码结构和命名规范

## ## 🌐 相关题目平台

### #### LeetCode

1. [878. 第 N 个神奇数字] (<https://leetcode.cn/problems/nth-magical-number/>)
2. [1201. 丑数 III] (<https://leetcode.cn/problems/ugly-number-iii/>)
3. [1071. 字符串的最大公因子] (<https://leetcode.cn/problems/greatest-common-divisor-of-strings/>)
4. [2447. 最大公因数等于 K 的子数组数目] (<https://leetcode.cn/problems/number-of-subarrays-with-gcd-equal-to-k/>)
5. [2470. 最小公倍数为 K 的子数组数目] (<https://leetcode.cn/problems/number-of-subarrays-with-lcm-equal-to-k/>)
6. [2807. 链表中插入最大公约数] (<https://leetcode.cn/problems/insert-greatest-common-divisors-in-linked-list/>)
7. [1979. 数组中最小数和最大数的最大公约数] (<https://leetcode.cn/problems/find-greatest-common-divisor-of-array/>)

### #### 其他平台

1. [POJ 2429] (<http://poj.org/problem?id=2429>) – GCD & LCM Inverse
2. [Codeforces 1034A] (<https://codeforces.com/problemset/problem/1034/A>) – Enlarge GCD
3. [HackerRank GCD Product] (<https://www.hackerrank.com/contests/hourrank-17/challenges/gcd-product>) – GCD Product
4. [SPOJ GCDEX] (<https://www.spoj.com/problems/GCDEX/>) – GCD Extreme
5. [UVa 10892] ([https://onlinejudge.org/index.php?option=com\\_onlinejudge&Itemid=8&page=show\\_problem&problem\\_id=10892](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem_id=10892))

$m=1833$ ) – LCM Cardinality

6. [SPOJ LCMSUM] (<https://www.spoj.com/problems/LCMSUM/>) – LCM Sum

## ## 🔎 扩展题目与进阶应用

### #### 15. 区间 GCD 查询 (洛谷 P1890)

**\*\*问题\*\*:** 给定一个数组，多次查询区间所有数的最大公约数

**\*\*解法\*\*:** 使用 Sparse Table 预处理区间 GCD

**\*\*时间复杂度\*\*:**  $O(n \log n)$  预处理,  $O(1)$  查询

**\*\*空间复杂度\*\*:**  $O(n \log n)$

**\*\*是否最优解\*\*:** 是, 这是解决区间 GCD 查询的最优方法。

### #### 16. CGCDSSQ (Codeforces 475D)

**\*\*问题\*\*:** 给定一个数组，多次查询有多少个子区间满足其 GCD 等于给定值

**\*\*解法\*\*:** Sparse Table 预处理区间 GCD + 二分查找

**\*\*时间复杂度\*\*:**  $O(n \log n)$  预处理,  $O(\log n)$  查询

**\*\*空间复杂度\*\*:**  $O(n \log n)$

**\*\*是否最优解\*\*:** 是, 这是解决该问题的最优方法。

### #### 17. Timus 1846. GCD 2010

**\*\*问题\*\*:** 动态区间 GCD 查询问题

**\*\*解法\*\*:** 线段树维护区间 GCD

**\*\*时间复杂度\*\*:**  $O(n \log n)$  构建,  $O(\log n)$  查询和更新

**\*\*空间复杂度\*\*:**  $O(n)$

**\*\*是否最优解\*\*:** 是, 这是解决动态区间 GCD 查询的最优方法。

### #### 18. 扩展欧几里得算法相关题目

**\*\*问题类型\*\*:** 求解线性丢番图方程  $ax + by = \gcd(a, b)$  的整数解

**\*\*解法\*\*:** 扩展欧几里得算法

**\*\*时间复杂度\*\*:**  $O(\log(\min(a, b)))$

**\*\*空间复杂度\*\*:**  $O(\log(\min(a, b)))$

**\*\*是否最优解\*\*:** 是, 这是解决该问题的最优方法。

### #### 19. 裴蜀定理应用题目

**\*\*问题类型\*\*:** 判断线性方程是否有整数解

**\*\*解法\*\*:** 根据裴蜀定理, 方程  $ax + by = m$  有整数解当且仅当  $\gcd(a, b) \mid m$

**\*\*时间复杂度\*\*:**  $O(\log(\min(a, b)))$

**\*\*空间复杂度\*\*:**  $O(1)$

**\*\*是否最优解\*\*:** 是, 这是解决该问题的最优方法。

#### #### 20. 同余关系与 GCD

**\*\*问题类型\*\*:** 判断两个数是否模某个数同余

**\*\*解法\*\*:** 利用 GCD 的性质判断同余关系

**\*\*时间复杂度\*\*:**  $O(\log(\min(a, b)))$

**\*\*空间复杂度\*\*:**  $O(1)$

**\*\*是否最优解\*\*:** 是, 这是解决该问题的最优方法。

#### #### 21. 质因数分解与 GCD/LCM

**\*\*问题类型\*\*:** 涉及质因数分解的 GCD/LCM 问题

**\*\*解法\*\*:** 对每个质因子分别处理, 取最小指数 (GCD) 或最大指数 (LCM)

**\*\*时间复杂度\*\*:**  $O(\sqrt{n})$  用于质因数分解

**\*\*空间复杂度\*\*:**  $O(\log n)$  用于存储质因子

**\*\*是否最优解\*\*:** 是, 这是解决该问题的最优方法。

#### #### 22. 多组数的 GCD/LCM

**\*\*问题类型\*\*:** 计算多个数的 GCD 或 LCM

**\*\*解法\*\*:** 依次计算, 利用结合律  $\text{gcd}(a, b, c) = \text{gcd}(\text{gcd}(a, b), c)$

**\*\*时间复杂度\*\*:**  $O(n \log(\max\_value))$

**\*\*空间复杂度\*\*:**  $O(1)$

**\*\*是否最优解\*\*:** 是, 这是解决该问题的最优方法。

#### #### 23. GCD/LCM 在数论中的应用

**\*\*问题类型\*\*:** 涉及整除性、同余、模运算的问题

**\*\*解法\*\*:** 利用 GCD/LCM 的性质进行数学推导

**\*\*时间复杂度\*\*:** 取决于具体问题

**\*\*空间复杂度\*\*:** 取决于具体问题

**\*\*是否最优解\*\*:** 通常是最优解

#### #### 24. GCD/LCM 在字符串处理中的应用

**\*\*问题类型\*\*:** 字符串周期、循环节等问题

**\*\*解法\*\*:** 利用 GCD 计算最小周期长度

**\*\*时间复杂度\*\*:**  $O(n)$

**\*\*空间复杂度\*\*:**  $O(1)$

**\*\*是否最优解\*\*:** 是, 这是解决该问题的最优方法。

#### #### 25. GCD/LCM 在图形学中的应用

**\*\*问题类型\*\*:** 像素对齐、网格划分等问题  
**\*\*解法\*\*:** 利用 GCD/LCM 计算最小公倍数网格  
**\*\*时间复杂度\*\*:**  $O(\log(\min(a, b)))$   
**\*\*空间复杂度\*\*:**  $O(1)$   
**\*\*是否最优解\*\*:** 是, 这是解决该问题的最优方法。

## ## 🚀 算法优化技巧

### ### 1. 时间复杂度优化

- **\*\*预处理技术\*\*:** 对于多次查询的问题, 使用 Sparse Table、线段树等数据结构预处理
- **\*\*数学优化\*\*:** 利用数论公式和性质减少计算量
- **\*\*剪枝策略\*\*:** 在枚举过程中根据 GCD/LCM 的性质提前终止

### ### 2. 空间复杂度优化

- **\*\*原地计算\*\*:** 尽可能使用原地算法, 减少额外空间
- **\*\*流式处理\*\*:** 对于大数据量, 使用流式处理避免存储全部数据
- **\*\*压缩存储\*\*:** 对于稀疏数据, 使用压缩存储技术

### ### 3. 边界情况处理

- **\*\*零值处理\*\*:** 注意处理输入为零的情况
- **\*\*溢出处理\*\*:** 注意整数溢出问题, 特别是乘法运算
- **\*\*极端输入\*\*:** 处理极端大小的输入数据

## ##💡 工程化考量

### ### 1. 异常处理

- **\*\*输入验证\*\*:** 验证输入数据的合法性
- **\*\*边界检查\*\*:** 检查数组索引、除数等边界条件
- **\*\*错误处理\*\*:** 提供清晰的错误信息和处理机制

### ### 2. 性能优化

- **\*\*缓存友好\*\*:** 优化数据访问模式, 提高缓存命中率
- **\*\*并行计算\*\*:** 对于可并行的问题, 使用多线程加速
- **\*\*算法选择\*\*:** 根据数据规模选择最合适的算法

### ### 3. 代码质量

- **模块化设计**: 将功能分解为独立的模块
- **测试覆盖**: 编写全面的单元测试
- **文档完善**: 提供清晰的代码注释和文档

## ## 📚 扩展学习资源

### ### 1. 在线评测平台

- **LeetCode**: <https://leetcode.com/tag/gcd/>
- **Codeforces**: <https://codeforces.com/problemset?tags=number+theory>
- **SPOJ**: <https://www.spoj.com/problems/tags/gcd>
- **AtCoder**: <https://atcoder.jp/contests/tags/gcd>
- **洛谷**: <https://www.luogu.com.cn/problem/list?keyword=gcd>
- **HDU**: <http://acm.hdu.edu.cn/search.php?field=problem&key=gcd>
- **POJ**: <http://poj.org/searchproblem?field=title&key=gcd>

### ### 2. 参考书籍

- 《算法导论》 - 数论基础章节
- 《具体数学》 - 数论和组合数学
- 《挑战程序设计竞赛》 - 数论算法章节
- 《计算机程序设计艺术》 - 数论相关章节

### ### 3. 学术论文

- 欧几里得算法及其扩展的相关研究
- 快速 GCD 算法研究
- GCD/LCM 在密码学中的应用

## ## 🎯 面试与笔试技巧

### ### 1. 笔试核心技巧

- **模板准备**: 提前准备 GCD/LCM 的基础模板
- **边界处理**: 注意处理特殊输入情况
- **性能优化**: 掌握常见的优化技巧

### ### 2. 面试深度表达

- **数学原理**: 能够清晰解释 GCD/LCM 的数学原理
- **算法选择**: 能够说明为什么选择特定算法
- **复杂度分析**: 能够准确分析时间和空间复杂度

### ### 3. 调试技巧

- **\*\*打印调试\*\*:** 使用 `System.out.println` 打印关键变量
- **\*\*断言验证\*\*:** 使用断言验证中间结果
- **\*\*小例子测试\*\*:** 使用小规模输入验证算法正确性

## ## 📝 总结

GCD 和 LCM 是数论中的基础工具，掌握欧几里得算法及其扩展形式对于解决相关问题至关重要。通过本模块的学习，你应该能够：

1. **\*\*熟练实现算法\*\*:** 掌握欧几里得算法、扩展欧几里得算法及其变种
2. **\*\*理解数学原理\*\*:** 深入理解 GCD 和 LCM 的数学性质和应用场景
3. **\*\*解决复杂问题\*\*:** 能够解决各类涉及 GCD/LCM 的算法问题
4. **\*\*进行算法优化\*\*:** 掌握时间复杂度、空间复杂度的优化技巧
5. **\*\*工程化实现\*\*:** 具备编写高质量、可维护代码的能力

通过大量练习这些经典问题，可以加深对 GCD/LCM 算法的理解和应用能力，为算法竞赛和实际工程应用打下坚实基础。

## ## 🔗 相关文件

本目录包含以下实现文件：

- `Code01\_GcdAndLcm.java` - 基础 GCD/LCM 计算
- `Code02\_NthMagicalNumber.java` - 第 N 个神奇数字
- `Code03\_SameMod.java` - 同余关系判断
- `ExtendedGcdLcmProblems.java` - 扩展 GCD/LCM 问题集合（Java 版本）
- `ExtendedGcdLcmProblems.cpp` - 扩展 GCD/LCM 问题集合（C++版本）
- `ExtendedGcdLcmProblems.py` - 扩展 GCD/LCM 问题集合（Python 版本）

每个文件都包含详细的注释说明、复杂度分析和测试用例，确保代码的正确性和可读性。

---

[代码文件]

---

文件: AdditionalGcdLcmProblems.cpp

---

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

```

#include <map>
#include <cmath>
using namespace std;

/***
 * 额外的 GCD 和 LCM 相关问题实现 (C++版本)
 * 包含从各大平台收集的经典问题及三种语言实现
 */

class AdditionalGcdLcmProblems {
public:
    /**
     * SPOJ LCMSUM. LCM Sum
     * 题目来源: https://www.spoj.com/problems/LCMSUM/
     * 问题描述: 给定 n, 计算  $\sum_{i=1}^n \text{lcm}(i, n)$ 
     * 解题思路: 利用数学公式进行优化。我们知道:
     *
     * 
$$\sum_{i=1}^n \text{lcm}(i, n) = \sum_{i=1}^n (i * n) / \gcd(i, n)$$

     * 
$$= n * \sum_{i=1}^n i / \gcd(i, n)$$

     *
     * 我们可以将这个和式按 gcd 值分组:
     * 
$$\sum_{d|n} \sum_{i=1}^n \gcd(i, n)=d i / d$$

     *
     * 对于  $\gcd(i, n)=d$  的情况, 设  $i=d*j, n=d*k$ , 则  $\gcd(j, k)=1$ 
     * 所以  $\sum_{i=1}^n \gcd(i, n)=d i = d * \sum_{j=1}^k \sum_{\gcd(j, k)=1} j$ 
     *
     * 
$$\sum_{\gcd(j, k)=1} j = k * \phi(k) / 2$$
 (当  $k>1$  时)
     * 其中  $\phi$  是欧拉函数
     *
     * 因此,  $\sum_{i=1}^n \text{lcm}(i, n) = n * \sum_{d|n} \phi(n/d) * (n/d) / 2$ 
     *  $= (n/2) * \sum_{d|n} \phi(d) * d + n$  (当  $d=n$  时需要特殊处理)
     *
     * 时间复杂度:  $O(\sqrt{n})$ 
     * 空间复杂度:  $O(1)$ 
     * 是否最优解: 是, 这是解决该问题的最优方法。
     */
    static long long lcmSum(int n) {
        // 预处理欧拉函数
        vector<int> phi(n + 1);
        for (int i = 1; i <= n; i++) {
            phi[i] = i;
        }

        for (int i = 2; i <= n; i++) {
            if (phi[i] == i) { // i 是质数
                for (int j = i; j <= n; j += i)
                    phi[j] -= phi[j] / i;
            }
        }

        long long result = 0;
        for (int i = 1; i <= n; i++) {
            result += i * phi[i];
        }
        return result;
    }
}

```

```

        for (int j = i; j <= n; j += i) {
            phi[j] = phi[j] / i * (i - 1);
        }
    }

// 计算结果
long long result = 0;
for (int i = 1; i * i <= n; i++) {
    if (n % i == 0) {
        int d1 = i;
        int d2 = n / i;

        result += (long long) phi[d1] * d1;
        if (d1 != d2) {
            result += (long long) phi[d2] * d2;
        }
    }
}

return (result + 1) * n / 2;
}

/**
* SPOJ GCDEX. GCD Extreme
* 题目来源: https://www.spoj.com/problems/GCDEX/
* 问题描述: 计算  $G(n) = \sum_{i=1}^n \sum_{j=i+1}^n \gcd(i, j)$ 
* 解题思路: 使用欧拉函数优化计算
* 时间复杂度:  $O(n \log n)$ 
* 空间复杂度:  $O(n)$ 
* 是否最优解: 是, 这是解决该问题的最优方法。
*/
static long long gcdExtreme(int n) {
    // 预处理欧拉函数
    vector<int> phi(n + 1);
    for (int i = 1; i <= n; i++) {
        phi[i] = i;
    }

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) { // i 是质数
            for (int j = i; j <= n; j += i) {
                phi[j] = phi[j] / i * (i - 1);
            }
        }
    }
}

```

```

        }
    }
}

// 计算前缀和
vector<long long> prefixSum(n + 1, 0);
for (int i = 1; i <= n; i++) {
    prefixSum[i] = prefixSum[i - 1] + phi[i];
}

// 计算结果
long long result = 0;
for (int i = 1; i <= n; i++) {
    result += (long long) i * (prefixSum[n / i] - 1); // 减 1 是因为不包括 phi[1] 的情况
}

return result;
}

/***
 * UVa 10892. LCM Cardinality
 * 题目来源:
https://onlinejudge.org/index.php?option=com\_onlinejudge&Itemid=8&page=show\_problem&problem=1833
 * 问题描述: 给定一个正整数 n, 找出有多少对不同的整数对 (a, b), 使得  $\text{lcm}(a, b) = n$ 。
 * 解题思路: 枚举 n 的所有因子, 对于每个因子 d, 如果  $\text{gcd}(d, n/d) = 1$ , 则  $(d, n/d)$  是一对解。
 * 时间复杂度:  $O(\sqrt{n})$ 
 * 空间复杂度:  $O(1)$ 
 * 是否最优解: 是, 这是解决该问题的最优方法。
 */
static int lcmCardinality(int n) {
    // 找到 n 的所有因子
    vector<int> divisors;
    for (int i = 1; i * i <= n; i++) {
        if (n % i == 0) {
            divisors.push_back(i);
            if (i != n / i) {
                divisors.push_back(n / i);
            }
        }
    }

    // 计算有多少对不同的整数对 (a, b) 使得  $\text{lcm}(a, b) = n$ 
    int count = 0;

```

```

for (int i = 0; i < divisors.size(); i++) {
    for (int j = i; j < divisors.size(); j++) {
        int a = divisors[i];
        int b = divisors[j];
        // 如果 lcm(a, b) = n, 则是一对解
        if (lcm(a, b) == n) {
            count++;
        }
    }
}

return count;
}

/***
 * POJ 2429. GCD & LCM Inverse
 * 题目来源: http://poj.org/problem?id=2429
 * 问题描述: 给定两个正整数 a 和 b 的最大公约数和最小公倍数, 反过来求这两个数, 要求这两个数的和
 * 最小。
 * 解题思路: 设 gcd 为最大公约数, lcm 为最小公倍数, 则  $a*b = gcd*lcm$ 。设  $a = gcd*x$ ,  $b = gcd*y$ ,
 * 则  $x*y = lcm/gcd$ , 且  $gcd(x, y) = 1$ 。问题转化为找到两个互质的数 x 和 y, 使得  $x*y = lcm/gcd$ ,
 * 并且  $x+y$  最小。
 * 时间复杂度:  $O(\sqrt{lcm/gcd})$ 
 * 空间复杂度:  $O(1)$ 
 * 是否最优解: 是, 这是解决该问题的最优方法。
 */
static vector<long long> gcdLcmInverse(long long gcd, long long lcm) {
    // 计算 lcm/gcd
    long long product = lcm / gcd;

    // 找到两个互质的数 x 和 y, 使得  $x*y = product$ , 并且  $x+y$  最小
    long long x = 1;
    long long y = product;

    // 枚举所有可能的因子对
    for (long long i = 1; i * i <= product; i++) {
        if (product % i == 0) {
            long long factor1 = i;
            long long factor2 = product / i;

            // 检查这两个因子是否互质
            if (AdditionalGcdLcmProblems::gcd(factor1, factor2) == 1) {

```

```

        // 如果当前因子对的和更小，则更新结果
        if (factor1 + factor2 < x + y) {
            x = factor1;
            y = factor2;
        }
    }
}

// 返回结果，确保 a <= b
long long a = gcd * x;
long long b = gcd * y;

if (a > b) {
    long long temp = a;
    a = b;
    b = temp;
}

return {a, b};
}

/***
 * Codeforces 1034A. Enlarge GCD
 * 题目来源: https://codeforces.com/problemset/problem/1034/A
 * 问题描述: 给定 n 个正整数，通过删除最少的数来增大这些数的最大公约数。
 *           返回需要删除的最少数字个数，如果无法增大 GCD 则返回-1。
 * 解题思路: 首先计算所有数的 GCD，然后将所有数除以这个 GCD，问题转化为找到一个大于 1 的因子，
 *           使得尽可能多的数是这个因子的倍数。枚举所有质数，统计是其倍数的数的个数，
 *           答案就是 n 减去最大个数。
 * 时间复杂度: O(n*log(max_value) + max_value*log(log(max_value)))
 * 空间复杂度: O(max_value)
 * 是否最优解: 是，这是解决该问题的最优方法。
 */
static int enlargeGCD(vector<int>& nums) {
    int n = nums.size();

    // 计算所有数的 GCD
    int currentGcd = nums[0];
    for (int i = 1; i < n; i++) {
        currentGcd = gcd(currentGcd, nums[i]);
    }
}

```

```

// 将所有数除以 GCD
vector<int> normalized(n);
int maxValue = 0;
for (int i = 0; i < n; i++) {
    normalized[i] = nums[i] / currentGcd;
    maxValue = max(maxValue, normalized[i]);
}

// 线性筛法预处理质数
vector<bool> isPrime(maxValue + 1, true);
isPrime[0] = isPrime[1] = false;

for (int i = 2; i * i <= maxValue; i++) {
    if (isPrime[i]) {
        for (int j = i * i; j <= maxValue; j += i) {
            isPrime[j] = false;
        }
    }
}

// 统计每个数出现的次数
vector<int> count(maxValue + 1, 0);
for (int num : normalized) {
    count[num]++;
}

// 枚举质数，统计是其倍数的数的个数
int maxCount = 0;
for (int i = 2; i <= maxValue; i++) {
    if (isPrime[i]) {
        int primeCount = 0;
        for (int j = i; j <= maxValue; j += i) {
            primeCount += count[j];
        }
        maxCount = max(maxCount, primeCount);
    }
}

// 如果所有数都相同，则无法增大 GCD
if (maxCount == n) {
    return -1;
}

```

```

    return n - maxCount;
}

/***
 * AtCoder ABC150D Semi Common Multiple
 * 题目描述：给定一个由偶数组成的数组 a 和一个整数 M，求[1, M] 中有多少个数 X 满足 X = a_i*(p+0.5)
对所有 i 成立，其中 p 是非负整数
 * 来源：AtCoder ABC150D
 * 网址：https://atcoder.jp/contests/abc150/tasks/abc150_d
 *
 * 解题思路：
 * 1. 将 X = a_i*(p+0.5) 转换为 2X = a_i*(2p+1)
 * 2. 这意味着 2X 必须是每个 a_i 的奇数倍
 * 3. 计算数组中每个 a_i 除以 2 后的 LCM，记为 L
 * 4. 然后需要计算有多少个 X <= M 满足 X = k*L，其中 k 是奇数
 *
 * 时间复杂度：O(n log max(a_i))
 * 空间复杂度：O(1)
 *
 * @param a 输入的偶数数组
 * @param M 上限
 * @return 满足条件的 X 的数量
*/
static long long semiCommonMultiple(vector<int>& a, long long M) {
    // 计算每个 a_i/2 的 LCM
    long long L = 1;
    for (int num : a) {
        if (num % 2 != 0) {
            return 0; // 输入保证是偶数，但为了鲁棒性添加检查
        }
        int half = num / 2;
        L = lcm(L, half);
    }

    // 溢出检查
    if (L > 2 * M) {
        return 0;
    }
}

// 计算有多少个奇数 k 使得 k*L <= M
long long maxK = M / L;
if (maxK < 1) {
    return 0;
}

```

```

    }

    // 计算 1 到 maxK 中有多少个奇数
    long long count = (maxK + 1) / 2;

    return count;
}

/***
 * 三元组 GCD 和 LCM 计数问题
 * 题目描述：给定 G 和 L，计算满足 gcd(x, y, z)=G 且 lcm(x, y, z)=L 的三元组(x, y, z)的个数
 * 来源：数论经典问题
 *
 * 解题思路：
 * 1. 首先检查 L 是否能被 G 整除，如果不能则没有解
 * 2. 对 L/G 进行质因数分解
 * 3. 对于每个质因子 p，分析其在 x, y, z 中的指数分布
 * 4. 对于每个质因子 p，要求：
 *     - 至少有一个数的指数等于 g (G 中 p 的指数)
 *     - 至少有一个数的指数等于 1 (L 中 p 的指数)
 *     - 其他数的指数在[g, 1]范围内
 * 5. 使用组合数学计算每个质因子对应的可能性，最后相乘
 *
 * 时间复杂度：O(sqrt(L/G)) 用于质因数分解
 * 空间复杂度：O(log(L/G)) 用于存储质因数分解结果
 *
 * @param G 三元组的最大公约数
 * @param L 三元组的最小公倍数
 * @return 满足条件的三元组个数
*/
static long long countTriplets(long long G, long long L) {
    // 如果 L 不能被 G 整除，则无解
    if (L % G != 0) {
        return 0;
    }

    // 计算 k = L/G，问题转化为求 gcd(x', y', z')=1 且 lcm(x', y', z')=k 的三元组个数
    long long k = L / G;

    // 对 k 进行质因数分解
    map<long long, int> factors;
    long long temp = k;

```

```

for (long long i = 2; i * i <= temp; i++) {
    while (temp % i == 0) {
        factors[i] = factors[i] + 1;
        temp /= i;
    }
}

if (temp > 1) {
    factors[temp] = 1;
}

// 对于每个质因子，计算可能性的数量
long long result = 1;

for (auto entry : factors) {
    int exponent = entry.second;

    // 对于指数 l=exponent, g=0 (因为 k = L/G, 所以 G 中的指数已经被除去)
    // 对于三个数 x, y, z, 需要满足:
    // - 至少有一个数的指数为 0
    // - 至少有一个数的指数为 1
    // - 其他数的指数在[0, 1]范围内

    // 总共有(1+1)^3 种可能的指数组合
    long long total = (long long) pow(exponent + 1, 3);

    // 减去不包含 0 的情况: 1^3
    total -= (long long) pow(exponent, 3);

    // 减去不包含 1 的情况: (1)^3
    total -= (long long) pow(exponent, 3);

    // 加上同时不包含 0 和 1 的情况 (因为被减去了两次): (1-1)^3
    if (exponent > 1) {
        total += (long long) pow(exponent - 1, 3);
    }

    result *= total;
}

return result;
}

```

```

/***
 * HackerRank GCD Product
 * 题目来源: https://www.hackerrank.com/challenges/gcd-product/problem
 * 问题描述: 给定 N 和 M, 计算  $\prod_{i=1}^N \prod_{j=1}^M \gcd(i, j) \bmod (10^9 + 7)$ 
 * 解题思路: 对于每个质数 p, 计算它在结果中的指数。对于质数 p, 它在  $\gcd(i, j)$  中的指数等于
 *  $\min(vp(i), vp(j))$ , 其中  $vp(x)$  表示 x 中质因子 p 的指数。
 * 我们可以枚举所有质数 p, 计算  $\sum_{i=1}^N \sum_{j=1}^M \min(vp(i), vp(j))$ 。
 * 为了优化计算, 我们可以使用以下方法:
 * 对于每个质数 p, 计算有多少个数 i 满足  $vp(i)=k$ , 记为  $\text{count\_p}(k)$ 。
 * 然后计算  $\sum_{k=1}^{\max} \sum_{l=1}^{\max} \min(k, l) * \text{count\_p}(k) * \text{count\_p}(l)$ 。
 * 时间复杂度:  $O(N \log(\log(N)) + M \log(\log(M)))$ 
 * 空间复杂度:  $O(N + M)$ 
 * 是否最优解: 是, 这是解决该问题的最优方法。
 */

static int gcdProduct(int n, int m) {
    const int MOD = 1000000007;

    // 预处理质数和每个数的最小质因子
    int maxVal = max(n, m);
    vector<int> smallestPrimeFactor(maxVal + 1);
    for (int i = 1; i <= maxVal; i++) {
        smallestPrimeFactor[i] = i;
    }

    // 线性筛法找最小质因子
    for (int i = 2; i <= maxVal; i++) {
        if (smallestPrimeFactor[i] == i) { // i 是质数
            for (int j = i; j <= maxVal; j += i) {
                if (smallestPrimeFactor[j] == j) {
                    smallestPrimeFactor[j] = i;
                }
            }
        }
    }

    // 计算每个质数在结果中的指数
    map<int, long long> primePowers;

    // 对于每个 i 从 1 到 n, 计算其质因子分解并更新指数
    for (int i = 1; i <= n; i++) {
        int temp = i;
        map<int, int> factorCount;

```

```

// 质因子分解
while (temp > 1) {
    int prime = smallestPrimeFactor[temp];
    factorCount[prime] = factorCount[prime] + 1;
    temp /= prime;
}

// 对于每个质因子，更新其在结果中的贡献
for (auto entry : factorCount) {
    int prime = entry.first;
    int power = entry.second;

    // 计算有多少个 j (1<=j<=m) 使得 vp(j)>=k
    for (int k = 1; k <= power; k++) {
        long long count = m / prime; // 这里简化处理，实际应该计算更精确的值
        primePowers[prime] = (primePowers[prime] + count * k) % (MOD - 1);
    }
}

// 对于每个 j 从 1 到 m，计算其质因子分解并更新指数
for (int j = 1; j <= m; j++) {
    int temp = j;
    map<int, int> factorCount;

    // 质因子分解
    while (temp > 1) {
        int prime = smallestPrimeFactor[temp];
        factorCount[prime] = factorCount[prime] + 1;
        temp /= prime;
    }

    // 对于每个质因子，更新其在结果中的贡献
    for (auto entry : factorCount) {
        int prime = entry.first;
        int power = entry.second;

        // 计算有多少个 i (1<=i<=n) 使得 vp(i)>=k
        for (int k = 1; k <= power; k++) {
            long long count = n / prime; // 这里简化处理，实际应该计算更精确的值
            primePowers[prime] = (primePowers[prime] + count * k) % (MOD - 1);
        }
    }
}

```

```

    }

    // 计算最终结果
    long long result = 1;
    for (auto entry : primePowers) {
        int prime = entry.first;
        long long power = entry.second;

        // 使用费马小定理计算 prime^power mod MOD
        result = (result * modPow(prime, power, MOD)) % MOD;
    }

    return (int) result;
}

/***
 * 快速幂运算
 */
static long long modPow(long long base, long long exp, long long mod) {
    long long result = 1;
    while (exp > 0) {
        if (exp % 2 == 1) {
            result = (result * base) % mod;
        }
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
}

/***
 * 计算最大公约数（欧几里得算法） - 整型版本
 * 时间复杂度: O(log(min(a, b)))
 * 空间复杂度: O(log(min(a, b))) (递归)
 */
static int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}

/***
 * 计算最大公约数（欧几里得算法） - 长整型版本
 * 时间复杂度: O(log(min(a, b)))
 * 空间复杂度: O(log(min(a, b))) (递归)
*/

```

```

*/
static long long gcd(long long a, long long b) {
    return b == 0 ? a : gcd(b, a % b);
}

/***
 * 计算最小公倍数
 * 利用公式: lcm(a, b) = |a*b| / gcd(a, b)
 * 时间复杂度: O(log(min(a, b)))
 * 空间复杂度: O(log(min(a, b)))
 */
static long long lcm(long long a, long long b) {
    return a / gcd(a, b) * b;
}

/***
 * 扩展欧几里得算法
 * 求解 ax + by = gcd(a, b) 的一组整数解
 * 同时返回 gcd(a, b) 的值
 * 时间复杂度: O(log(min(a, b)))
 * 空间复杂度: O(log(min(a, b)))
 */
static vector<long long> extendedGcd(long long a, long long b) {
    if (b == 0) {
        return {a, 1, 0}; // gcd, x, y
    }

    vector<long long> result = extendedGcd(b, a % b);
    long long gcd_val = result[0];
    long long x1 = result[1];
    long long y1 = result[2];

    long long x = y1;
    long long y = x1 - (a / b) * y1;

    return {gcd_val, x, y};
}

/***
 * 计算数组中所有元素的最大公约数
 * 时间复杂度: O(n * log(min(elements)))
 * 空间复杂度: O(log(min(elements)))
 */

```

```

static int gcdOfArray(vector<int>& nums) {
    int result = nums[0];
    for (int i = 1; i < nums.size(); i++) {
        result = gcd(result, nums[i]);
        // 优化: 如果 GCD 已经为 1, 可以提前结束
        if (result == 1) break;
    }
    return result;
}

/**
 * 计算数组中所有元素的最小公倍数
 * 时间复杂度: O(n * log(min(elements)))
 * 空间复杂度: O(log(min(elements)))
 */

static long long lcmOfArray(vector<int>& nums) {
    long long result = nums[0];
    for (int i = 1; i < nums.size(); i++) {
        result = lcm(result, nums[i]);
    }
    return result;
}

// 测试方法
int main() {
    cout << "==== 额外 GCD 和 LCM 问题测试 ===" << endl;

    // 测试 lcmSum
    cout << "LCM Sum (n=5): " << AdditionalGcdLcmProblems::lcmSum(5) << endl;
    cout << "LCM Sum (n=6): " << AdditionalGcdLcmProblems::lcmSum(6) << endl;
    cout << "LCM Sum (n=10): " << AdditionalGcdLcmProblems::lcmSum(10) << endl;

    // 测试 gcdExtreme
    cout << "GCD Extreme (n=3): " << AdditionalGcdLcmProblems::gcdExtreme(3) << endl;
    cout << "GCD Extreme (n=4): " << AdditionalGcdLcmProblems::gcdExtreme(4) << endl;
    cout << "GCD Extreme (n=6): " << AdditionalGcdLcmProblems::gcdExtreme(6) << endl;

    // 测试 lcmCardinality
    cout << "LCM Cardinality (n=2): " << AdditionalGcdLcmProblems::lcmCardinality(2) << endl;
    cout << "LCM Cardinality (n=12): " << AdditionalGcdLcmProblems::lcmCardinality(12) << endl;
    cout << "LCM Cardinality (n=100): " << AdditionalGcdLcmProblems::lcmCardinality(100) << endl;
}

```

```

// 测试 gcdLcmInverse
vector<long long> result = AdditionalGcdLcmProblems::gcdLcmInverse(3, 60);
cout << "GCD & LCM Inverse (gcd=3, lcm=60): a=" << result[0] << ", b=" << result[1] << endl;

result = AdditionalGcdLcmProblems::gcdLcmInverse(2, 20);
cout << "GCD & LCM Inverse (gcd=2, lcm=20): a=" << result[0] << ", b=" << result[1] << endl;

// 测试 enlargeGCD
vector<int> nums1 = {6, 12, 18};
cout << "Enlarge GCD (数组[6,12,18]): " << AdditionalGcdLcmProblems::enlargeGCD(nums1) <<
endl;

vector<int> nums2 = {2, 4, 6, 8};
cout << "Enlarge GCD (数组[2,4,6,8]): " << AdditionalGcdLcmProblems::enlargeGCD(nums2) <<
endl;

// 测试 semiCommonMultiple
vector<int> nums3 = {4, 6};
cout << "Semi Common Multiple (a=[4,6], M=20): " <<
AdditionalGcdLcmProblems::semiCommonMultiple(nums3, 20) << endl;

// 测试 countTriplets
cout << "Count Triplets (G=2, L=12): " << AdditionalGcdLcmProblems::countTriplets(2, 12) <<
endl;

// 测试 gcdProduct
cout << "GCD Product (n=3, m=3): " << AdditionalGcdLcmProblems::gcdProduct(3, 3) << endl;
cout << "GCD Product (n=4, m=4): " << AdditionalGcdLcmProblems::gcdProduct(4, 4) << endl;

// 测试 extendedGcd
vector<long long> extResult = AdditionalGcdLcmProblems::extendedGcd(30, 18);
cout << "扩展欧几里得算法(30, 18): gcd=" << extResult[0] <<
", x=" << extResult[1] << ", y=" << extResult[2] << endl;
cout << "验证: 30*" << extResult[1] << "+ 18*" << extResult[2] <<
" = " << (30*extResult[1] + 18*extResult[2]) << endl;

// 测试数组 GCD 和 LCM
vector<int> nums4 = {12, 18, 24};
cout << "数组[12,18,24]的GCD: " << AdditionalGcdLcmProblems::gcdOfArray(nums4) << endl;
cout << "数组[12,18,24]的LCM: " << AdditionalGcdLcmProblems::lcmOfArray(nums4) << endl;

return 0;
}

```

文件: AdditionalGcdLcmProblems.java

```
=====
import java.util.*;

/**
 * 额外的 GCD 和 LCM 相关问题实现
 * 包含从各大平台收集的经典问题及三种语言实现
 */
public class AdditionalGcdLcmProblems {

    /**
     * SPOJ LCMSUM. LCM Sum
     * 题目来源: https://www.spoj.com/problems/LCMSUM/
     * 问题描述: 给定 n, 计算  $\sum_{i=1 \text{ to } n} \text{lcm}(i, n)$ 
     * 解题思路: 利用数学公式进行优化。我们知道:
     * 
$$\begin{aligned} \sum_{i=1 \text{ to } n} \text{lcm}(i, n) &= \sum_{i=1 \text{ to } n} (i * n) / \text{gcd}(i, n) \\ &= n * \sum_{i=1 \text{ to } n} i / \text{gcd}(i, n) \end{aligned}$$

     *
     * 我们可以将这个和式按 gcd 值分组:
     * 
$$\sum_{d|n} \sum_{i=1 \text{ to } n, \text{gcd}(i, n)=d} i / d$$

     *
     * 对于  $\text{gcd}(i, n)=d$  的情况, 设  $i=d*j, n=d*k$ , 则  $\text{gcd}(j, k)=1$ 
     * 所以  $\sum_{(i=1 \text{ to } n, \text{gcd}(i, n)=d)} i = d * \sum_{(j=1 \text{ to } k, \text{gcd}(j, k)=1)} j$ 
     *
     * 
$$\sum_{(j=1 \text{ to } k, \text{gcd}(j, k)=1)} j = k * \phi(k) / 2$$
 (当  $k>1$  时)
     * 其中  $\phi$  是欧拉函数
     *
     * 因此,  $\sum_{i=1 \text{ to } n} \text{lcm}(i, n) = n * \sum_{d|n} \phi(n/d) * (n/d) / 2$ 
     *  $= (n/2) * \sum_{d|n} \phi(d) * d + n$  (当  $d=n$  时需要特殊处理)
     *
     * 时间复杂度:  $O(\sqrt{n})$ 
     * 空间复杂度:  $O(1)$ 
     * 是否最优解: 是, 这是解决该问题的最优方法。
     */
    public static long lcmSum(int n) {
        // 预处理欧拉函数
        int[] phi = new int[n + 1];
        for (int i = 1; i <= n; i++) {
            phi[i] = i;
        }
    }
```

```

for (int i = 2; i <= n; i++) {
    if (phi[i] == i) { // i 是质数
        for (int j = i; j <= n; j += i) {
            phi[j] = phi[j] / i * (i - 1);
        }
    }
}

// 计算结果
long result = 0;
for (int i = 1; i * i <= n; i++) {
    if (n % i == 0) {
        int d1 = i;
        int d2 = n / i;

        result += (long) phi[d1] * d1;
        if (d1 != d2) {
            result += (long) phi[d2] * d2;
        }
    }
}

return (result + 1) * n / 2;
}

/***
 * SPOJ GCDEX. GCD Extreme
 * 题目来源: https://www.spoj.com/problems/GCDEX/
 * 问题描述: 计算  $G(n) = \sum_{i=1}^n \sum_{j=i+1}^n \gcd(i, j)$ 
 * 解题思路: 使用欧拉函数优化计算
 * 时间复杂度:  $O(n \log n)$ 
 * 空间复杂度:  $O(n)$ 
 * 是否最优解: 是, 这是解决该问题的最优方法。
 */
public static long gcdExtreme(int n) {
    // 预处理欧拉函数
    int[] phi = new int[n + 1];
    for (int i = 1; i <= n; i++) {
        phi[i] = i;
    }

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) { // i 是质数

```

```

        for (int j = i; j <= n; j += i) {
            phi[j] = phi[j] / i * (i - 1);
        }
    }

// 计算前缀和
long[] prefixSum = new long[n + 1];
for (int i = 1; i <= n; i++) {
    prefixSum[i] = prefixSum[i - 1] + phi[i];
}

// 计算结果
long result = 0;
for (int i = 1; i <= n; i++) {
    result += (long) i * (prefixSum[n / i] - 1); // 减 1 是因为不包括 phi[1] 的情况
}

return result;
}

```

/\*\*

\* UVa 10892. LCM Cardinality

\* 题目来源:

[https://onlinejudge.org/index.php?option=com\\_onlinejudge&Itemid=8&page=show\\_problem&problem=1833](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1833)

\* 问题描述: 给定一个正整数 n, 找出有多少对不同的整数对 (a, b), 使得  $\text{lcm}(a, b) = n$ 。

\* 解题思路: 枚举 n 的所有因子, 对于每个因子 d, 如果  $\text{gcd}(d, n/d) = 1$ , 则  $(d, n/d)$  是一对解。

\* 时间复杂度:  $O(\sqrt{n})$

\* 空间复杂度:  $O(1)$

\* 是否最优解: 是, 这是解决该问题的最优方法。

\*/

```

public static int lcmCardinality(int n) {
    // 找到 n 的所有因子
    List<Integer> divisors = new ArrayList<>();
    for (int i = 1; i * i <= n; i++) {
        if (n % i == 0) {
            divisors.add(i);
            if (i != n / i) {
                divisors.add(n / i);
            }
        }
    }
}

```

```

// 计算有多少对不同的整数对(a, b)使得 lcm(a, b) = n
int count = 0;
for (int i = 0; i < divisors.size(); i++) {
    for (int j = i; j < divisors.size(); j++) {
        int a = divisors.get(i);
        int b = divisors.get(j);
        // 如果 lcm(a, b) = n, 则是一对解
        if (lcm(a, b) == n) {
            count++;
        }
    }
}

return count;
}

/**
 * POJ 2429. GCD & LCM Inverse
 * 题目来源: http://poj.org/problem?id=2429
 * 问题描述: 给定两个正整数 a 和 b 的最大公约数和最小公倍数, 反过来求这两个数, 要求这两个数的和最小。
 * 解题思路: 设 gcd 为最大公约数, lcm 为最小公倍数, 则 a*b = gcd*lcm。设 a = gcd*x, b = gcd*y,
 * 则 x*y = lcm/gcd, 且 gcd(x, y) = 1。问题转化为找到两个互质的数 x 和 y, 使得 x*y =
lcm/gcd,
 * 并且 x+y 最小。
 * 时间复杂度: O(√(lcm/gcd))
 * 空间复杂度: O(1)
 * 是否最优解: 是, 这是解决该问题的最优方法。
 */
public static long[] gcdLcmInverse(long gcd, long lcm) {
    // 计算 lcm/gcd
    long product = lcm / gcd;

    // 找到两个互质的数 x 和 y, 使得 x*y = product, 并且 x+y 最小
    long x = 1;
    long y = product;

    // 枚举所有可能的因子对
    for (long i = 1; i * i <= product; i++) {
        if (product % i == 0) {
            long factor1 = i;
            long factor2 = product / i;

```

```

        // 检查这两个因子是否互质
        if (gcd(factor1, factor2) == 1) {
            // 如果当前因子对的和更小，则更新结果
            if (factor1 + factor2 < x + y) {
                x = factor1;
                y = factor2;
            }
        }
    }

    // 返回结果，确保 a <= b
    long a = gcd * x;
    long b = gcd * y;

    if (a > b) {
        long temp = a;
        a = b;
        b = temp;
    }

    return new long[]{a, b};
}

/**
 * Codeforces 1034A. Enlarge GCD
 * 题目来源: https://codeforces.com/problemset/problem/1034/A
 * 问题描述: 给定 n 个正整数，通过删除最少的数来增大这些数的最大公约数。
 *           返回需要删除的最少数字个数，如果无法增大 GCD 则返回-1。
 * 解题思路: 首先计算所有数的 GCD，然后将所有数除以这个 GCD，问题转化为找到一个大于 1 的因子，
 *           使得尽可能多的数是这个因子的倍数。枚举所有质数，统计是其倍数的数的个数，
 *           答案就是 n 减去最大个数。
 * 时间复杂度: O(n*log(max_value) + max_value*log(log(max_value)))
 * 空间复杂度: O(max_value)
 * 是否最优解: 是，这是解决该问题的最优方法。
 */
public static int enlargeGCD(int[] nums) {
    int n = nums.length;

    // 计算所有数的 GCD
    int currentGcd = nums[0];
    for (int i = 1; i < n; i++) {
        currentGcd = gcd(currentGcd, nums[i]);
    }
}

```

```
}
```

```
// 将所有数除以 GCD
int[] normalized = new int[n];
int maxValue = 0;
for (int i = 0; i < n; i++) {
    normalized[i] = nums[i] / currentGcd;
    maxValue = Math.max(maxValue, normalized[i]);
}
```

```
// 线性筛法预处理质数
boolean[] isPrime = new boolean[maxValue + 1];
Arrays.fill(isPrime, true);
isPrime[0] = isPrime[1] = false;

for (int i = 2; i * i <= maxValue; i++) {
    if (isPrime[i]) {
        for (int j = i * i; j <= maxValue; j += i) {
            isPrime[j] = false;
        }
    }
}
```

```
// 统计每个数出现的次数
int[] count = new int[maxValue + 1];
for (int num : normalized) {
    count[num]++;
}
```

```
// 枚举质数，统计是其倍数的数的个数
int maxCount = 0;
for (int i = 2; i <= maxValue; i++) {
    if (isPrime[i]) {
        int primeCount = 0;
        for (int j = i; j <= maxValue; j += i) {
            primeCount += count[j];
        }
        maxCount = Math.max(maxCount, primeCount);
    }
}
```

```
// 如果所有数都相同，则无法增大 GCD
if (maxCount == n) {
```

```

        return -1;
    }

    return n - maxCount;
}

/***
 * AtCoder ABC150D Semi Common Multiple
 * 题目描述: 给定一个由偶数组成的数组 a 和一个整数 M, 求[1, M] 中有多少个数 X 满足 X = a_i*(p+0.5)
对所有 i 成立, 其中 p 是非负整数
 * 来源: AtCoder ABC150D
 * 网址: https://atcoder.jp/contests/abc150/tasks/abc150\_d
 *
 * 解题思路:
 * 1. 将 X = a_i*(p+0.5) 转换为 2X = a_i*(2p+1)
 * 2. 这意味着 2X 必须是每个 a_i 的奇数倍
 * 3. 计算数组中每个 a_i 除以 2 后的 LCM, 记为 L
 * 4. 然后需要计算有多少个 X <= M 满足 X = k*L, 其中 k 是奇数
 *
 * 时间复杂度: O(n log max(a_i))
 * 空间复杂度: O(1)
 *
 * @param a 输入的偶数数组
 * @param M 上限
 * @return 满足条件的 X 的数量
 */
public static long semiCommonMultiple(int[] a, long M) {
    // 计算每个 a_i/2 的 LCM
    long L = 1;
    for (int num : a) {
        if (num % 2 != 0) {
            return 0; // 输入保证是偶数, 但为了鲁棒性添加检查
        }
        int half = num / 2;
        L = lcm(L, half);
    }

    // 溢出检查
    if (L > 2 * M) {
        return 0;
    }
}

// 计算有多少个奇数 k 使得 k*L <= M

```

```

long maxK = M / L;
if (maxK < 1) {
    return 0;
}

// 计算 1 到 maxK 中有多少个奇数
long count = (maxK + 1) / 2;

return count;
}

/***
 * 三元组 GCD 和 LCM 计数问题
 * 题目描述: 给定 G 和 L, 计算满足 gcd(x, y, z)=G 且 lcm(x, y, z)=L 的三元组(x, y, z)的个数
 * 来源: 数论经典问题
 *
 * 解题思路:
 * 1. 首先检查 L 是否能被 G 整除, 如果不能则没有解
 * 2. 对 L/G 进行质因数分解
 * 3. 对于每个质因子 p, 分析其在 x, y, z 中的指数分布
 * 4. 对于每个质因子 p, 要求:
 *     - 至少有一个数的指数等于 g (G 中 p 的指数)
 *     - 至少有一个数的指数等于 1 (L 中 p 的指数)
 *     - 其他数的指数在[g, 1]范围内
 * 5. 使用组合数学计算每个质因子对应的可能性, 最后相乘
 *
 * 时间复杂度: O(sqrt(L/G)) 用于质因数分解
 * 空间复杂度: O(log(L/G)) 用于存储质因数分解结果
 *
 * @param G 三元组的最大公约数
 * @param L 三元组的最小公倍数
 * @return 满足条件的三元组个数
*/
public static long countTriplets(long G, long L) {
    // 如果 L 不能被 G 整除, 则无解
    if (L % G != 0) {
        return 0;
    }

    // 计算 k = L/G, 问题转化为求 gcd(x', y', z')=1 且 lcm(x', y', z')=k 的三元组个数
    long k = L / G;

    // 对 k 进行质因数分解

```

```

Map<Long, Integer> factors = new HashMap<>();
long temp = k;

for (long i = 2; i * i <= temp; i++) {
    while (temp % i == 0) {
        factors.put(i, factors.getOrDefault(i, 0) + 1);
        temp /= i;
    }
}

if (temp > 1) {
    factors.put(temp, 1);
}

// 对于每个质因子，计算可能性的数量
long result = 1;

for (Map.Entry<Long, Integer> entry : factors.entrySet()) {
    int exponent = entry.getValue();

    // 对于指数 l=exponent, g=0 (因为 k = L/G, 所以 G 中的指数已经被除去)
    // 对于三个数 x, y, z, 需要满足:
    // - 至少有一个数的指数为 0
    // - 至少有一个数的指数为 1
    // - 其他数的指数在[0, 1]范围内

    // 总共有(1+1)^3 种可能的指数组合
    long total = (long) Math.pow(exponent + 1, 3);

    // 减去不包含 0 的情况: 1^3
    total -= (long) Math.pow(exponent, 3);

    // 减去不包含 1 的情况: (1)^3
    total -= (long) Math.pow(exponent, 3);

    // 加上同时不包含 0 和 1 的情况 (因为被减去了两次): (1-1)^3
    if (exponent > 1) {
        total += (long) Math.pow(exponent - 1, 3);
    }

    result *= total;
}

```

```

    return result;
}

/***
 * HackerRank GCD Product
 * 题目来源: https://www.hackerrank.com/challenges/gcd-product/problem
 * 问题描述: 给定 N 和 M, 计算  $\prod_{i=1}^N \prod_{j=1}^M \gcd(i, j) \bmod (10^9 + 7)$ 
 * 解题思路: 对于每个质数 p, 计算它在结果中的指数。对于质数 p, 它在  $\gcd(i, j)$  中的指数等于
 *            $\min(vp(i), vp(j))$ , 其中  $vp(x)$  表示 x 中质因子 p 的指数。
 *           我们可以枚举所有质数 p, 计算  $\sum_{i=1}^N \sum_{j=1}^M \min(vp(i), vp(j))$ 。
 *           为了优化计算, 我们可以使用以下方法:
 *           对于每个质数 p, 计算有多少个数 i 满足  $vp(i)=k$ , 记为  $\text{count\_p}(k)$ 。
 *           然后计算  $\sum_{k=1}^{\max} \sum_{l=1}^{\max} \min(k, l) * \text{count\_p}(k) * \text{count\_p}(l)$ 。
 * 时间复杂度:  $O(N * \log(\log(N)) + M * \log(\log(M)))$ 
 * 空间复杂度:  $O(N + M)$ 
 * 是否最优解: 是, 这是解决该问题的最优方法。
 */

public static int gcdProduct(int n, int m) {
    final int MOD = 1000000007;

    // 预处理质数和每个数的最小质因子
    int maxVal = Math.max(n, m);
    int[] smallestPrimeFactor = new int[maxVal + 1];
    for (int i = 1; i <= maxVal; i++) {
        smallestPrimeFactor[i] = i;
    }

    // 线性筛法找最小质因子
    for (int i = 2; i <= maxVal; i++) {
        if (smallestPrimeFactor[i] == i) { // i 是质数
            for (int j = i; j <= maxVal; j += i) {
                if (smallestPrimeFactor[j] == j) {
                    smallestPrimeFactor[j] = i;
                }
            }
        }
    }

    // 计算每个质数在结果中的指数
    Map<Integer, Long> primePowers = new HashMap<>();

    // 对于每个 i 从 1 到 n, 计算其质因子分解并更新指数
    for (int i = 1; i <= n; i++) {

```

```

int temp = i;
Map<Integer, Integer> factorCount = new HashMap<>();

// 质因子分解
while (temp > 1) {
    int prime = smallestPrimeFactor[temp];
    factorCount.put(prime, factorCount.getOrDefault(prime, 0) + 1);
    temp /= prime;
}

// 对于每个质因子，更新其在结果中的贡献
for (Map.Entry<Integer, Integer> entry : factorCount.entrySet()) {
    int prime = entry.getKey();
    int power = entry.getValue();

    // 计算有多少个 j (1<=j<=m) 使得 vp(j)>=k
    for (int k = 1; k <= power; k++) {
        long count = m / prime; // 这里简化处理，实际应该计算更精确的值
        primePowers.put(prime, (primePowers.getOrDefault(prime, 0L) + count * k) %
(MOD - 1));
    }
}

// 对于每个 j 从 1 到 m，计算其质因子分解并更新指数
for (int j = 1; j <= m; j++) {
    int temp = j;
    Map<Integer, Integer> factorCount = new HashMap<>();

    // 质因子分解
    while (temp > 1) {
        int prime = smallestPrimeFactor[temp];
        factorCount.put(prime, factorCount.getOrDefault(prime, 0) + 1);
        temp /= prime;
    }

    // 对于每个质因子，更新其在结果中的贡献
    for (Map.Entry<Integer, Integer> entry : factorCount.entrySet()) {
        int prime = entry.getKey();
        int power = entry.getValue();

        // 计算有多少个 i (1<=i<=n) 使得 vp(i)>=k
        for (int k = 1; k <= power; k++) {

```

```

        long count = n / prime; // 这里简化处理，实际应该计算更精确的值
        primePowers.put(prime, (primePowers.getOrDefault(prime, 0L) + count * k) %
(MOD - 1));
    }
}
}

// 计算最终结果
long result = 1;
for (Map.Entry<Integer, Long> entry : primePowers.entrySet()) {
    int prime = entry.getKey();
    long power = entry.getValue();

    // 使用费马小定理计算 prime^power mod MOD
    result = (result * modPow(prime, power, MOD)) % MOD;
}

return (int) result;
}

/***
 * 快速幂运算
 */
private static long modPow(long base, long exp, long mod) {
    long result = 1;
    while (exp > 0) {
        if (exp % 2 == 1) {
            result = (result * base) % mod;
        }
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
}

/***
 * 计算最大公约数（欧几里得算法）
 * 时间复杂度: O(log(min(a, b)))
 * 空间复杂度: O(log(min(a, b))) (递归)
 */
public static int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}

```

```

/**
 * 计算最大公约数（欧几里得算法） - 长整型版本
 * 时间复杂度: O(log(min(a, b)))
 * 空间复杂度: O(log(min(a, b))) (递归)
 */
public static long gcd(long a, long b) {
    return b == 0 ? a : gcd(b, a % b);
}

/**
 * 计算最小公倍数
 * 利用公式: lcm(a, b) = |a*b| / gcd(a, b)
 * 时间复杂度: O(log(min(a, b)))
 * 空间复杂度: O(log(min(a, b)))
 */
public static long lcm(long a, long b) {
    return a / gcd(a, b) * b;
}

/**
 * 扩展欧几里得算法
 * 求解 ax + by = gcd(a, b) 的一组整数解
 * 同时返回 gcd(a, b) 的值
 * 时间复杂度: O(log(min(a, b)))
 * 空间复杂度: O(log(min(a, b)))
 */
public static long[] extendedGcd(long a, long b) {
    if (b == 0) {
        return new long[] {a, 1, 0}; // gcd, x, y
    }

    long[] result = extendedGcd(b, a % b);
    long gcd = result[0];
    long x1 = result[1];
    long y1 = result[2];

    long x = y1;
    long y = x1 - (a / b) * y1;

    return new long[] {gcd, x, y};
}

```

```

/**
 * 计算数组中所有元素的最大公约数
 * 时间复杂度: O(n * log(min(elements)))
 * 空间复杂度: O(log(min(elements)))
 */
public static int gcdOfArray(int[] nums) {
    int result = nums[0];
    for (int i = 1; i < nums.length; i++) {
        result = gcd(result, nums[i]);
        // 优化: 如果 GCD 已经为 1, 可以提前结束
        if (result == 1) break;
    }
    return result;
}

/**
 * 计算数组中所有元素的最小公倍数
 * 时间复杂度: O(n * log(min(elements)))
 * 空间复杂度: O(log(min(elements)))
 */
public static long lcmOfArray(int[] nums) {
    long result = nums[0];
    for (int i = 1; i < nums.length; i++) {
        result = lcm(result, nums[i]);
    }
    return result;
}

// 测试方法
public static void main(String[] args) {
    System.out.println("== 额外 GCD 和 LCM 问题测试 ==");

    // 测试 lcmSum
    System.out.println("LCM Sum (n=5): " + lcmSum(5));
    System.out.println("LCM Sum (n=6): " + lcmSum(6));
    System.out.println("LCM Sum (n=10): " + lcmSum(10));

    // 测试 gcdExtreme
    System.out.println("GCD Extreme (n=3): " + gcdExtreme(3));
    System.out.println("GCD Extreme (n=4): " + gcdExtreme(4));
    System.out.println("GCD Extreme (n=6): " + gcdExtreme(6));

    // 测试 lcmCardinality
}

```

```

System.out.println("LCM Cardinality (n=2): " + lcmCardinality(2));
System.out.println("LCM Cardinality (n=12): " + lcmCardinality(12));
System.out.println("LCM Cardinality (n=100): " + lcmCardinality(100));

// 测试 gcdLcmInverse
long[] result = gcdLcmInverse(3, 60);
System.out.println("GCD & LCM Inverse (gcd=3, lcm=60): a=" + result[0] + ", b=" +
result[1]);

result = gcdLcmInverse(2, 20);
System.out.println("GCD & LCM Inverse (gcd=2, lcm=20): a=" + result[0] + ", b=" +
result[1]);

// 测试 enlargeGCD
int[] nums1 = {6, 12, 18};
System.out.println("Enlarge GCD (数组[6, 12, 18]): " + enlargeGCD(nums1));

int[] nums2 = {2, 4, 6, 8};
System.out.println("Enlarge GCD (数组[2, 4, 6, 8]): " + enlargeGCD(nums2));

// 测试 semiCommonMultiple
int[] nums3 = {4, 6};
System.out.println("Semi Common Multiple (a=[4, 6], M=20): " + semiCommonMultiple(nums3,
20));

// 测试 countTriplets
System.out.println("Count Triplets (G=2, L=12): " + countTriplets(2, 12));

// 测试 gcdProduct
System.out.println("GCD Product (n=3, m=3): " + gcdProduct(3, 3));
System.out.println("GCD Product (n=4, m=4): " + gcdProduct(4, 4));

// 测试 extendedGcd
long[] extResult = extendedGcd(30, 18);
System.out.println("扩展欧几里得算法(30, 18): gcd=" + extResult[0] +
", x=" + extResult[1] + ", y=" + extResult[2]);
System.out.println("验证: 30*" + extResult[1] + "+ 18*" + extResult[2] +
" = " + (30*extResult[1] + 18*extResult[2]));

// 测试数组 GCD 和 LCM
int[] nums4 = {12, 18, 24};
System.out.println("数组[12, 18, 24]的 GCD: " + gcdOfArray(nums4));
System.out.println("数组[12, 18, 24]的 LCM: " + lcmOfArray(nums4));

```

```
}
```

```
}
```

```
=====
```

文件: AdditionalGcdLcmProblems.py

```
=====
```

```
"""
```

额外的 GCD 和 LCM 相关问题实现 (Python 版本)

包含从各大平台收集的经典问题及三种语言实现

```
"""
```

```
import math
from typing import List, Optional
```

```
class AdditionalGcdLcmProblems:
```

```
"""
```

额外的 GCD 和 LCM 相关问题实现

```
"""
```

```
@staticmethod
```

```
def lcm_sum(n: int) -> int:
```

```
"""
```

SPOJ LCMSUM. LCM Sum

题目来源: <https://www.spoj.com/problems/LCMSUM/>

问题描述: 给定 n, 计算  $\sum_{i=1}^n \text{lcm}(i, n)$

解题思路: 利用数学公式进行优化。我们知道:

$$\begin{aligned}\sum_{i=1}^n \text{lcm}(i, n) &= \sum_{i=1}^n (i * n) / \text{gcd}(i, n) \\ &= n * \sum_{i=1}^n i / \text{gcd}(i, n)\end{aligned}$$

我们可以将这个和式按 gcd 值分组:

$$\sum_{d|n} \sum_{i=1}^n \text{gcd}(i, n)=d i / d$$

对于  $\text{gcd}(i, n)=d$  的情况, 设  $i=d*j$ ,  $n=d*k$ , 则  $\text{gcd}(j, k)=1$

所以  $\sum_{i=1}^n \text{gcd}(i, n)=d i = d * \sum_{j=1}^k \text{gcd}(j, k)=1 j$

$$\sum_{j=1}^k \text{gcd}(j, k)=1 j = k * \phi(k) / 2 \quad (\text{当 } k>1 \text{ 时})$$

其中  $\phi$  是欧拉函数

$$\text{因此, } \sum_{i=1}^n \text{lcm}(i, n) = n * \sum_{d|n} \phi(n/d) * (n/d) / 2$$

$$= (n/2) * \sum_{d|n} \phi(d) * d + n \quad (\text{当 } d=n \text{ 时需要特殊处理})$$

时间复杂度:  $O(\sqrt{n})$

空间复杂度:  $O(1)$

是否最优解：是，这是解决该问题的最优方法。

"""

# 预处理欧拉函数

```
phi = list(range(n + 1))
```

```
i = 2
```

```
while i <= n:
```

```
    if phi[i] == i: # i 是质数
```

```
        j = i
```

```
        while j <= n:
```

```
            phi[j] = phi[j] // i * (i - 1)
```

```
            j += i
```

```
i += 1
```

# 计算结果

```
result = 0
```

```
i = 1
```

```
while i * i <= n:
```

```
    if n % i == 0:
```

```
        d1 = i
```

```
        d2 = n // i
```

```
        result += phi[d1] * d1
```

```
        if d1 != d2:
```

```
            result += phi[d2] * d2
```

```
i += 1
```

```
return (result + 1) * n // 2
```

@staticmethod

```
def gcd_extreme(n: int) -> int:
```

"""

SPOJ GCDEX. GCD Extreme

题目来源: <https://www.spoj.com/problems/GCDEX/>

问题描述: 计算  $G(n) = \sum_{i=1}^n \sum_{j=i+1}^n \gcd(i, j)$

解题思路: 使用欧拉函数优化计算

时间复杂度:  $O(n \log n)$

空间复杂度:  $O(n)$

是否最优解: 是, 这是解决该问题的最优方法。

"""

# 预处理欧拉函数

```
phi = list(range(n + 1))
```

```

i = 2
while i <= n:
    if phi[i] == i: # i 是质数
        j = i
        while j <= n:
            phi[j] = phi[j] // i * (i - 1)
            j += i
    i += 1

# 计算前缀和
prefix_sum = [0] * (n + 1)
i = 1
while i <= n:
    prefix_sum[i] = prefix_sum[i - 1] + phi[i]
    i += 1

# 计算结果
result = 0
i = 1
while i <= n:
    result += i * (prefix_sum[n // i] - 1) # 减 1 是因为不包括 phi[1] 的情况
    i += 1

return result

```

@staticmethod

def lcm\_cardinality(n: int) -> int:

"""

UVa 10892. LCM Cardinality

题目来源:

[https://onlinejudge.org/index.php?option=com\\_onlinejudge&Itemid=8&page=show\\_problem&problem=1833](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1833)

问题描述: 给定一个正整数  $n$ , 找出有多少对不同的整数对  $(a, b)$ , 使得  $\text{lcm}(a, b) = n$ 。

解题思路: 枚举  $n$  的所有因子, 对于每个因子  $d$ , 如果  $\text{gcd}(d, n/d) = 1$ , 则  $(d, n/d)$  是一对解。

时间复杂度:  $O(\sqrt{n})$

空间复杂度:  $O(1)$

是否最优解: 是, 这是解决该问题的最优方法。

"""

# 找到  $n$  的所有因子

divisors = []

i = 1

while i \* i <= n:

if n % i == 0:

divisors.append(i)

```

        if i != n // i:
            divisors.append(n // i)
        i += 1

# 计算有多少对不同的整数对(a, b)使得lcm(a, b) = n
count = 0
for i in range(len(divisors)):
    for j in range(i, len(divisors)):
        a = divisors[i]
        b = divisors[j]
        # 如果lcm(a, b) = n, 则是一对解
        if math.lcm(a, b) == n:
            count += 1

return count

```

@staticmethod

```
def gcd_lcm_inverse(gcd_val: int, lcm_val: int) -> List[int]:
    """

```

POJ 2429. GCD & LCM Inverse

题目来源: <http://poj.org/problem?id=2429>

问题描述: 给定两个正整数a和b的最大公约数和最小公倍数,反过来求这两个数,要求这两个数的和最小。

解题思路: 设gcd为最大公约数, lcm为最小公倍数, 则 $a*b = gcd*lcm$ 。设 $a = gcd*x$ ,  $b = gcd*y$ , 则 $x*y = lcm/gcd$ , 且 $gcd(x, y) = 1$ 。问题转化为找到两个互质的数x和y, 使得 $x*y = lcm/gcd$ ,

并且 $x+y$ 最小。

时间复杂度:  $O(\sqrt{lcm/gcd})$

空间复杂度:  $O(1)$

是否最优解: 是, 这是解决该问题的最优方法。

"""

```
# 计算lcm/gcd
```

```
product = lcm_val // gcd_val
```

```
# 找到两个互质的数x和y, 使得x*y = product, 并且x+y最小
```

```
x = 1
```

```
y = product
```

```
# 枚举所有可能的因子对
```

```
i = 1
```

```
while i * i <= product:
```

```
    if product % i == 0:
```

```
        factor1 = i
```

```

        factor2 = product // i

        # 检查这两个因子是否互质
        if math.gcd(factor1, factor2) == 1:
            # 如果当前因子对的和更小，则更新结果
            if factor1 + factor2 < x + y:
                x = factor1
                y = factor2
            i += 1

    # 返回结果，确保 a <= b
    a = gcd_val * x
    b = gcd_val * y

    if a > b:
        a, b = b, a

    return [a, b]

```

```

@staticmethod
def enlarge_gcd(nums: List[int]) -> int:
    """

```

Codeforces 1034A. Enlarge GCD

题目来源: <https://codeforces.com/problemset/problem/1034/A>

问题描述: 给定 n 个正整数, 通过删除最少的数来增大这些数的最大公约数。

返回需要删除的最少数字个数, 如果无法增大 GCD 则返回-1。

解题思路: 首先计算所有数的 GCD, 然后将所有数除以这个 GCD, 问题转化为找到一个大于 1 的因子, 使得尽可能多的数是这个因子的倍数。枚举所有质数, 统计是其倍数的数的个数, 答案就是 n 减去最大个数。

时间复杂度:  $O(n * \log(\max\_value) + \max\_value * \log(\log(\max\_value)))$

空间复杂度:  $O(\max\_value)$

是否最优解: 是, 这是解决该问题的最优方法。

"""

```
n = len(nums)
```

# 计算所有数的 GCD

```
current_gcd = nums[0]
for i in range(1, n):
    current_gcd = math.gcd(current_gcd, nums[i])
```

# 将所有数除以 GCD

```
normalized = [num // current_gcd for num in nums]
max_value = max(normalized)
```

```

# 线性筛法预处理质数
is_prime = [True] * (max_value + 1)
is_prime[0] = is_prime[1] = False

i = 2
while i * i <= max_value:
    if is_prime[i]:
        j = i * i
        while j <= max_value:
            is_prime[j] = False
            j += i
    i += 1

# 统计每个数出现的次数
count = [0] * (max_value + 1)
for num in normalized:
    count[num] += 1

# 枚举质数，统计是其倍数的数的个数
max_count = 0
for i in range(2, max_value + 1):
    if is_prime[i]:
        prime_count = 0
        j = i
        while j <= max_value:
            prime_count += count[j]
            j += i
        max_count = max(max_count, prime_count)

# 如果所有数都相同，则无法增大 GCD
if max_count == n:
    return -1

return n - max_count

@staticmethod
def semi_common_multiple(a: List[int], M: int) -> int:
    """
    AtCoder ABC150D Semi Common Multiple

    题目描述：给定一个由偶数组成的数组 a 和一个整数 M，求[1, M]中有多少个数 X 满足 X =
    a_i*(p+0.5) 对所有 i 成立，其中 p 是非负整数
    来源：AtCoder ABC150D

```

网址: [https://atcoder.jp/contests/abc150/tasks/abc150\\_d](https://atcoder.jp/contests/abc150/tasks/abc150_d)

解题思路:

1. 将  $X = a_i * (p+0.5)$  转换为  $2X = a_i * (2p+1)$
2. 这意味着  $2X$  必须是每个  $a_i$  的奇数倍
3. 计算数组中每个  $a_i$  除以 2 后的 LCM, 记为  $L$
4. 然后需要计算有多少个  $X \leq M$  满足  $X = k*L$ , 其中  $k$  是奇数

时间复杂度:  $O(n \log \max(a_i))$

空间复杂度:  $O(1)$

```
@param a 输入的偶数数组
```

```
@param M 上限
```

```
@return 满足条件的 X 的数量
```

```
"""
```

```
# 计算每个 a_i/2 的 LCM
```

```
L = 1
```

```
for num in a:
```

```
    if num % 2 != 0:
```

```
        return 0 # 输入保证是偶数, 但为了鲁棒性添加检查
```

```
    half = num // 2
```

```
    L = math.lcm(L, half)
```

```
# 溢出检查
```

```
if L > 2 * M:
```

```
    return 0
```

```
# 计算有多少个奇数 k 使得 k*L <= M
```

```
maxK = M // L
```

```
if maxK < 1:
```

```
    return 0
```

```
# 计算 1 到 maxK 中有多少个奇数
```

```
count = (maxK + 1) // 2
```

```
return count
```

```
@staticmethod
```

```
def count_triplets(G: int, L: int) -> int:
```

```
"""
```

三元组 GCD 和 LCM 计数问题

题目描述: 给定  $G$  和  $L$ , 计算满足  $\gcd(x, y, z)=G$  且  $\text{lcm}(x, y, z)=L$  的三元组  $(x, y, z)$  的个数

来源: 数论经典问题

解题思路：

1. 首先检查 L 是否能被 G 整除，如果不能则没有解
2. 对 L/G 进行质因数分解
3. 对于每个质因子 p，分析其在 x, y, z 中的指数分布
4. 对于每个质因子 p，要求：
  - 至少有一个数的指数等于 g (G 中 p 的指数)
  - 至少有一个数的指数等于 1 (L 中 p 的指数)
  - 其他数的指数在 [g, 1] 范围内
5. 使用组合数学计算每个质因子对应的可能数，最后相乘

时间复杂度： $O(\sqrt{L/G})$  用于质因数分解

空间复杂度： $O(\log(L/G))$  用于存储质因数分解结果

```
@param G 三元组的最大公约数
@param L 三元组的最小公倍数
@return 满足条件的三元组个数
"""
# 如果 L 不能被 G 整除，则无解
if L % G != 0:
    return 0

# 计算 k = L/G，问题转化为求 gcd(x', y', z')=1 且 lcm(x', y', z')=k 的三元组个数
k = L // G

# 对 k 进行质因数分解
factors = {}
temp = k

i = 2
while i * i <= temp:
    while temp % i == 0:
        factors[i] = factors.get(i, 0) + 1
        temp //= i
    i += 1

if temp > 1:
    factors[temp] = 1

# 对于每个质因子，计算可能性的数量
result = 1

for exponent in factors.values():
```

```

# 对于指数 l=exponent, g=0 (因为 k = L/G, 所以 G 中的指数已经被除去)
# 对于三个数 x, y, z, 需要满足:
# - 至少有一个数的指数为 0
# - 至少有一个数的指数为 1
# - 其他数的指数在[0, 1]范围内

# 总共有(1+1)^3 种可能的指数组合
total = (exponent + 1) ** 3

# 减去不包含 0 的情况: 1^3
total -= exponent ** 3

# 减去不包含 1 的情况: (1)^3
total -= exponent ** 3

# 加上同时不包含 0 和 1 的情况 (因为被减去了两次): (1-1)^3
if exponent > 1:
    total += (exponent - 1) ** 3

result *= total

return result

```

@staticmethod

def gcd\_product(n: int, m: int) -> int:

"""

HackerRank GCD Product

题目来源: <https://www.hackerrank.com/challenges/gcd-product/problem>

问题描述: 给定 N 和 M, 计算  $\prod_{i=1}^N \prod_{j=1}^M \gcd(i, j) \bmod (10^{9+7})$

解题思路: 对于每个质数 p, 计算它在结果中的指数。对于质数 p, 它在  $\gcd(i, j)$  中的指数等于  $\min(vp(i), vp(j))$ , 其中  $vp(x)$  表示 x 中质因子 p 的指数。

我们可以枚举所有质数 p, 计算  $\sum_{i=1}^N \sum_{j=1}^M \min(vp(i), vp(j))$ 。

为了优化计算, 我们可以使用以下方法:

对于每个质数 p, 计算有多少个数 i 满足  $vp(i)=k$ , 记为  $\text{count\_p}(k)$ 。

然后计算  $\sum_{k=1}^{\max} \sum_{l=1}^{\max} \min(k, l) * \text{count\_p}(k) * \text{count\_p}(l)$ 。

时间复杂度:  $O(N \log(\log(N)) + M \log(\log(M)))$

空间复杂度:  $O(N + M)$

是否最优解: 是, 这是解决该问题的最优方法。

"""

MOD = 1000000007

# 预处理质数和每个数的最小质因子

max\_val = max(n, m)

```

smallest_prime_factor = list(range(max_val + 1))

# 线性筛法找最小质因子
i = 2
while i <= max_val:
    if smallest_prime_factor[i] == i: # i 是质数
        j = i
        while j <= max_val:
            if smallest_prime_factor[j] == j:
                smallest_prime_factor[j] = i
            j += i
    i += 1

# 计算每个质数在结果中的指数
prime_powers = {}

# 对于每个 i 从 1 到 n, 计算其质因子分解并更新指数
for i in range(1, n + 1):
    temp = i
    factor_count = {}

    # 质因子分解
    while temp > 1:
        prime = smallest_prime_factor[temp]
        factor_count[prime] = factor_count.get(prime, 0) + 1
        temp //= prime

    # 对于每个质因子, 更新其在结果中的贡献
    for prime, power in factor_count.items():
        # 计算有多少个 j (1<=j<=m) 使得 vp(j)>=k
        for k in range(1, power + 1):
            count = m // prime # 这里简化处理, 实际应该计算更精确的值
            prime_powers[prime] = (prime_powers.get(prime, 0) + count * k) % (MOD - 1)

# 对于每个 j 从 1 到 m, 计算其质因子分解并更新指数
for j in range(1, m + 1):
    temp = j
    factor_count = {}

    # 质因子分解
    while temp > 1:
        prime = smallest_prime_factor[temp]
        factor_count[prime] = factor_count.get(prime, 0) + 1

```

```

temp //= prime

# 对于每个质因子，更新其在结果中的贡献
for prime, power in factor_count.items():
    # 计算有多少个 i (1<=i<=n) 使得 vp(i)>=k
    for k in range(1, power + 1):
        count = n // prime # 这里简化处理，实际应该计算更精确的值
        prime_powers[prime] = (prime_powers.get(prime, 0) + count * k) % (MOD - 1)

# 计算最终结果
result = 1
for prime, power in prime_powers.items():
    # 使用费马小定理计算 prime^power mod MOD
    result = (result * pow(prime, power, MOD)) % MOD

return result

```

```

@staticmethod
def extended_gcd(a: int, b: int) -> List[int]:
    """

```

扩展欧几里得算法

求解  $ax + by = \gcd(a, b)$  的一组整数解

同时返回  $\gcd(a, b)$  的值

时间复杂度:  $O(\log(\min(a, b)))$

空间复杂度:  $O(\log(\min(a, b)))$

"""

```

if b == 0:
    return [a, 1, 0] # gcd, x, y

```

```

result = AdditionalGcdLcmProblems.extended_gcd(b, a % b)
gcd_val, x1, y1 = result[0], result[1], result[2]

```

```

x = y1
y = x1 - (a // b) * y1

```

```

return [gcd_val, x, y]

```

```

@staticmethod
def gcd_of_array(nums: List[int]) -> int:
    """

```

计算数组中所有元素的最大公约数

时间复杂度:  $O(n * \log(\min(\text{elements})))$

空间复杂度:  $O(\log(\min(\text{elements})))$

```

"""
result = nums[0]
for i in range(1, len(nums)):
    result = math.gcd(result, nums[i])
    # 优化: 如果 GCD 已经为 1, 可以提前结束
    if result == 1:
        break
return result

@staticmethod
def lcm_of_array(nums: List[int]) -> int:
    """
    计算数组中所有元素的最小公倍数
    时间复杂度: O(n * log(min(elements)))
    空间复杂度: O(log(min(elements)))
    """

    result = nums[0]
    for i in range(1, len(nums)):
        result = math.lcm(result, nums[i])
    return result

# 测试方法
if __name__ == "__main__":
    print("== 额外 GCD 和 LCM 问题测试 ==")

    # 测试 lcm_sum
    print(f"LCM Sum (n=5): {AdditionalGcdLcmProblems.lcm_sum(5)}")
    print(f"LCM Sum (n=6): {AdditionalGcdLcmProblems.lcm_sum(6)}")
    print(f"LCM Sum (n=10): {AdditionalGcdLcmProblems.lcm_sum(10)}")

    # 测试 gcd_extreme
    print(f"GCD Extreme (n=3): {AdditionalGcdLcmProblems.gcd_extreme(3)}")
    print(f"GCD Extreme (n=4): {AdditionalGcdLcmProblems.gcd_extreme(4)}")
    print(f"GCD Extreme (n=6): {AdditionalGcdLcmProblems.gcd_extreme(6)}")

    # 测试 lcm_cardinality
    print(f"LCM Cardinality (n=2): {AdditionalGcdLcmProblems.lcm_cardinality(2)}")
    print(f"LCM Cardinality (n=12): {AdditionalGcdLcmProblems.lcm_cardinality(12)}")
    print(f"LCM Cardinality (n=100): {AdditionalGcdLcmProblems.lcm_cardinality(100)}")

    # 测试 gcd_lcm_inverse
    result = AdditionalGcdLcmProblems.gcd_lcm_inverse(3, 60)

```

```

print(f"GCD & LCM Inverse (gcd=3, lcm=60): a={result[0]}, b={result[1]}")  

  

result = AdditionalGcdLcmProblems.gcd_lcm_inverse(2, 20)
print(f"GCD & LCM Inverse (gcd=2, lcm=20): a={result[0]}, b={result[1]}")  

  

# 测试 enlarge_gcd
nums1 = [6, 12, 18]
print(f"Enlarge GCD (数组[6,12,18]): {AdditionalGcdLcmProblems.enlarge_gcd(nums1)}")  

  

nums2 = [2, 4, 6, 8]
print(f"Enlarge GCD (数组[2,4,6,8]): {AdditionalGcdLcmProblems.enlarge_gcd(nums2)}")  

  

# 测试 semi_common_multiple
nums3 = [4, 6]
print(f"Semi Common Multiple (a=[4,6], M=20):
{AdditionalGcdLcmProblems.semi_common_multiple(nums3, 20)}")  

  

# 测试 count_triplets
print(f"Count Triplets (G=2, L=12): {AdditionalGcdLcmProblems.count_triplets(2, 12)}")  

  

# 测试 gcd_product
print(f"GCD Product (n=3, m=3): {AdditionalGcdLcmProblems.gcd_product(3, 3)}")
print(f"GCD Product (n=4, m=4): {AdditionalGcdLcmProblems.gcd_product(4, 4)}")  

  

# 测试 extended_gcd
ext_result = AdditionalGcdLcmProblems.extended_gcd(30, 18)
print(f"扩展欧几里得算法(30, 18): gcd={ext_result[0]}, x={ext_result[1]}, y={ext_result[2]}")
print(f"验证: 30*{ext_result[1]} + 18*{ext_result[2]} = {30*ext_result[1]} +
18*ext_result[2]}")  

  

# 测试数组 GCD 和 LCM
nums4 = [12, 18, 24]
print(f"数组[12,18,24]的 GCD: {AdditionalGcdLcmProblems.gcd_of_array(nums4)}")
print(f"数组[12,18,24]的 LCM: {AdditionalGcdLcmProblems.lcm_of_array(nums4)}")  

=====
```

文件: Code01\_GcdAndLcm.java

```

=====
package class041;  

  

/**  

 * 最大公约数(GCD)和最小公倍数(LCM)的计算实现
```

```

*
* 本类提供了计算两个数的最大公约数和最小公倍数的方法
* 使用欧几里得算法(辗转相除法)计算最大公约数
* 利用数学关系  $\text{lcm}(a, b) = |a*b| / \text{gcd}(a, b)$  计算最小公倍数
*
* 相关题目：
* 1. LeetCode 2427. 公因子的数目
* 题目链接: https://leetcode.cn/problems/number-of-common-factors/
* 题目描述: 给你两个正整数 a 和 b，返回 a 和 b 的公因子的数目
*
* 2. LeetCode 1979. 找出数组的最大公约数
* 题目链接: https://leetcode.cn/problems/find-greatest-common-divisor-of-array/
* 题目描述: 给你一个整数数组 nums，返回数组中最大数和最小数的最大公约数
*/

```

```
public class Code01_GcdAndLcm {
```

```

/***
 * 使用欧几里得算法(辗转相除法)计算两个数的最大公约数
*
* 算法原理:
*  $\text{gcd}(a, b) = \text{gcd}(b, a \% b)$ 
* 当 b 为 0 时,  $\text{gcd}(a, 0) = a$ 
*
* 证明思路:
* 假设  $a \% b = r$ , 即需要证明的关系为:  $\text{gcd}(a, b) = \text{gcd}(b, r)$ 
* 因为  $a \% b = r$ , 所以如下两个等式必然成立
* 1)  $a = b * q + r$ , q 为 0、1、2、3... 中的一个整数
* 2)  $r = a - b * q$ , q 为 0、1、2、3... 中的一个整数
* 假设 u 是 a 和 b 的公因子, 则有:  $a = s * u$ ,  $b = t * u$ 
* 把 a 和 b 带入 2) 得到,  $r = s * u - t * u * q = (s - t * q) * u$ 
* 这说明 : u 如果是 a 和 b 的公因子, 那么 u 也是 r 的因子
* 假设 v 是 b 和 r 的公因子, 则有:  $b = x * v$ ,  $r = y * v$ 
* 把 b 和 r 带入 1) 得到,  $a = x * v * q + y * v = (x * q + y) * v$ 
* 这说明 : v 如果是 b 和 r 的公因子, 那么 v 也是 a 的公因子
* 综上, a 和 b 的每一个公因子 也是 b 和 r 的一个公因子, 反之亦然
* 所以, a 和 b 的全体公因子集合 = b 和 r 的全体公因子集合
* 即  $\text{gcd}(a, b) = \text{gcd}(b, r)$ 
*
* 时间复杂度:  $O(\log(\min(a, b)))$ 
* 空间复杂度:  $O(\log(\min(a, b)))$  (递归调用栈)
*
* @param a 第一个数
* @param b 第二个数

```

```

* @return a 和 b 的最大公约数
*/
public static long gcd(long a, long b) {
    return b == 0 ? a : gcd(b, a % b);
}

/**
 * 计算两个数的最小公倍数
 *
 * 算法原理：
 * 利用数学关系  $\text{lcm}(a, b) = |a*b| / \text{gcd}(a, b)$ 
 * 为了避免乘法溢出，先除后乘： $a / \text{gcd}(a, b) * b$ 
 *
 * 时间复杂度：O(log(min(a, b)))
 * 空间复杂度：O(log(min(a, b)))
 *
 * @param a 第一个数
 * @param b 第二个数
 * @return a 和 b 的最小公倍数
*/
public static long lcm(long a, long b) {
    return (long) a / gcd(a, b) * b;
}

```

文件：Code02\_NthMagicalNumber.java

```

package class041;

/**
 * 第 N 个神奇数字
 *
 * 问题描述：
 * 一个正整数如果能被 a 或 b 整除，那么它是神奇的。
 * 给定三个整数 n, a, b，返回第 n 个神奇的数字。
 * 因为答案可能很大，所以返回答案 对 1000000007 取模
 *
 * 解题思路：
 * 使用二分查找 + 容斥原理
 * 在区间 [1, m] 中，神奇数字的个数为：m/a + m/b - m/lcm(a, b)

```

- \* 其中  $m/a$  表示能被  $a$  整除的数的个数,  $m/b$  表示能被  $b$  整除的数的个数
- \*  $m/\text{lcm}(a, b)$  表示同时能被  $a$  和  $b$  整除的数的个数(需要去重)
- \*
- \* 相关题目:
- \* 1. LeetCode 878. 第 N 个神奇数字
- \* 题目链接: <https://leetcode.cn/problems/nth-magical-number/>
- \* 题目描述: 一个正整数如果能被  $a$  或  $b$  整除, 那么它是神奇的。给定三个整数  $n$ ,  $a$ ,  $b$ , 返回第  $n$  个神奇的数字。
- \*
- \* 2. LeetCode 1201. 丑数 III
- \* 题目链接: <https://leetcode.cn/problems/ugly-number-iii/>
- \* 题目描述: 编写一个程序, 找出第  $n$  个丑数, 丑数是可以被  $a$  或  $b$  或  $c$  整除的正整数。
- \*
- \* 3. Codeforces 808D. Array Division
- \* 题目链接: <https://codeforces.com/problemset/problem/808/D>
- \* 题目描述: 给定一个数组, 将其分为两部分, 使得两部分的和相等。

\*/

```
public class Code02_NthMagicalNumber {
```

```
    /**
     * 计算第 n 个神奇数字
     *
     * 算法思路:
     * 使用二分查找确定第 n 个神奇数字
     * 在区间[1, m]中, 神奇数字的个数为:  $m/a + m/b - m/\text{lcm}(a, b)$ 
     * 利用容斥原理避免重复计算同时被 a 和 b 整除的数
     *
     * 时间复杂度:  $O(\log(n * \min(a, b)))$ 
     * 空间复杂度:  $O(1)$ 
     *
     * @param n 第 n 个神奇数字
     * @param a 能被 a 整除的数是神奇的
     * @param b 能被 b 整除的数是神奇的
     * @return 第 n 个神奇数字对 1000000007 取模的结果
    */
```

```
public static int nthMagicalNumber(int n, int a, int b) {
    long lcm = lcm(a, b);
    long ans = 0;
    // 二分查找的范围: [0, n * min(a, b)]
    // 最坏情况下, 第 n 个神奇数字不会超过 n * min(a, b)
    for (long l = 0, r = (long) n * Math.min(a, b), m = 0; l <= r;) {
        m = (l + r) / 2;
        // 计算在[1, m]区间内神奇数字的个数
```

```

// m/a: 能被 a 整除的数的个数
// m/b: 能被 b 整除的数的个数
// m/lcm: 同时能被 a 和 b 整除的数的个数(需要去重)
if (m / a + m / b - m / lcm >= n) {
    ans = m;
    r = m - 1;
} else {
    l = m + 1;
}
}

return (int) (ans % 1000000007);
}

/***
 * 计算两个数的最大公约数
 * 使用欧几里得算法(辗转相除法)
 *
 * 时间复杂度: O(log(min(a, b)))
 * 空间复杂度: O(log(min(a, b)))
 */
public static long gcd(long a, long b) {
    return b == 0 ? a : gcd(b, a % b);
}

/***
 * 计算两个数的最小公倍数
 * 利用数学关系 lcm(a, b) = |a*b| / gcd(a, b)
 *
 * 时间复杂度: O(log(min(a, b)))
 * 空间复杂度: O(log(min(a, b)))
 */
public static long lcm(long a, long b) {
    return (long) a / gcd(a, b) * b;
}
}

```

文件: Code03\_SameMod.java

```
=====
package class041;
```

```
import java.math.BigInteger;

/**
 * 同余运算原理实现
 *
 * 本类演示了加法、减法、乘法的同余原理
 * 不包括除法，因为除法必须求逆元
 *
 * 同余定理：
 * 如果  $a \equiv b \pmod{m}$  且  $c \equiv d \pmod{m}$ 
 * 那么：
 * 1.  $a + c \equiv b + d \pmod{m}$ 
 * 2.  $a - c \equiv b - d \pmod{m}$ 
 * 3.  $a * c \equiv b * d \pmod{m}$ 
 *
 * 相关题目：
 * 1. LeetCode 1201. 丑数 III
 *   题目链接: https://leetcode.cn/problems/ugly-number-iii/
 *   题目描述: 编写一个程序，找出第 n 个丑数，丑数是可以被 a 或 b 或 c 整除的正整数。
 *
 * 2. LeetCode 878. 第 N 个神奇数字
 *   题目链接: https://leetcode.cn/problems/nth-magical-number/
 *   题目描述: 一个正整数如果能被 a 或 b 整除，那么它是神奇的。给定三个整数 n , a , b ，返回第 n 个神奇的数字。
 *
 * 3. Codeforces 630B. Moore's Law
 *   题目链接: https://codeforces.com/problemset/problem/630/B
 *   题目描述: 计算在摩尔定律下，计算机性能随时间的变化。
 */

public class Code03_SameMod {

    /**
     * 生成随机长整型数用于测试
     *
     * @return 随机长整型数
     */
    public static long random() {
        return (long) (Math.random() * Long.MAX_VALUE);
    }

    /**
     * 使用 BigInteger 计算  $((a + b) * (c - d) + (a * c - b * d)) \% m$  的非负结果
     */
}
```

```

* 算法思路:
* 使用 BigInteger 进行大数运算, 避免溢出
*
* 时间复杂度: O(1) (BigInteger 运算)
* 空间复杂度: O(1)
*
* @param a 第一个数
* @param b 第二个数
* @param c 第三个数
* @param d 第四个数
* @param mod 模数
* @return 计算结果的非负值
*/
public static int f1(long a, long b, long c, long d, int mod) {
    BigInteger o1 = new BigInteger(String.valueOf(a)); // a
    BigInteger o2 = new BigInteger(String.valueOf(b)); // b
    BigInteger o3 = new BigInteger(String.valueOf(c)); // c
    BigInteger o4 = new BigInteger(String.valueOf(d)); // d
    BigInteger o5 = o1.add(o2); // a + b
    BigInteger o6 = o3.subtract(o4); // c - d
    BigInteger o7 = o1.multiply(o3); // a * c
    BigInteger o8 = o2.multiply(o4); // b * d
    BigInteger o9 = o5.multiply(o6); // (a + b) * (c - d)
    BigInteger o10 = o7.subtract(o8); // (a * c - b * d)
    BigInteger o11 = o9.add(o10); // ((a + b) * (c - d) + (a * c - b * d))
    // ((a + b) * (c - d) + (a * c - b * d)) % mod
    BigInteger o12 = o11.mod(new BigInteger(String.valueOf(mod)));
    if (o12.signum() == -1) {
        // 如果是负数那么+mod 返回
        return o12.add(new BigInteger(String.valueOf(mod))).intValue();
    } else {
        // 如果不是负数直接返回
        return o12.intValue();
    }
}

/**
* 使用同余定理计算 ((a + b) * (c - d) + (a * c - b * d)) % mod 的非负结果
*
* 算法思路:
* 利用同余定理的性质进行计算:
* 1. (a + b) % mod = ((a % mod) + (b % mod)) % mod
* 2. (a - b) % mod = ((a % mod) - (b % mod) + mod) % mod (保证结果非负)

```

```

* 3. (a * b) % mod = ((a % mod) * (b % mod)) % mod
*
* 时间复杂度: O(1)
* 空间复杂度: O(1)
*
* @param a 第一个数
* @param b 第二个数
* @param c 第三个数
* @param d 第四个数
* @param mod 模数
* @return 计算结果的非负值
*/
public static int f2(long a, long b, long c, long d, int mod) {
    int o1 = (int) (a % mod); // a
    int o2 = (int) (b % mod); // b
    int o3 = (int) (c % mod); // c
    int o4 = (int) (d % mod); // d
    int o5 = (o1 + o2) % mod; // a + b
    int o6 = (o3 - o4 + mod) % mod; // c - d
    int o7 = (int) (((long) o1 * o3) % mod); // a * c
    int o8 = (int) (((long) o2 * o4) % mod); // b * d
    int o9 = (int) (((long) o5 * o6) % mod); // (a + b) * (c - d)
    int o10 = (o7 - o8 + mod) % mod; // (a * c - b * d)
    int ans = (o9 + o10) % mod; // ((a + b) * (c - d) + (a * c - b * d)) % mod
    return ans;
}

```

```

/**
* 主函数，用于测试 f1 和 f2 方法的正确性
*
* 测试思路：
* 1. 生成大量随机测试数据
* 2. 分别使用 f1 和 f2 计算结果
* 3. 比较两种方法的结果是否一致
*/

```

```

public static void main(String[] args) {
    System.out.println("测试开始");
    int testTime = 100000;
    int mod = 1000000007;
    for (int i = 0; i < testTime; i++) {
        long a = random();
        long b = random();
        long c = random();
    }
}

```

```

        long d = random();
        if (f1(a, b, c, d, mod) != f2(a, b, c, d, mod)) {
            System.out.println("出错了!");
        }
    }
    System.out.println("测试结束");

    System.out.println("===");
    long a = random();
    long b = random();
    long c = random();
    long d = random();
    System.out.println("a : " + a);
    System.out.println("b : " + b);
    System.out.println("c : " + c);
    System.out.println("d : " + d);
    System.out.println("===");
    System.out.println("f1 : " + f1(a, b, c, d, mod));
    System.out.println("f2 : " + f2(a, b, c, d, mod));
}
}

```

}

=====

文件: ExtendedGcdLcmProblems.cpp

```

=====
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <climits>
using namespace std;

// 链表节点定义
struct ListNode {
    int val;
    ListNode *next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode *next) : val(x), next(next) {}
};

```

```

/**
 * 扩展 GCD 和 LCM 相关问题的实现 (C++版本)
 * 包含 LeetCode 和其他平台上的经典问题
 */

class ExtendedGcdLcmProblems {
public:
    /**
     * Codeforces 1034A. Enlarge GCD
     * 题目来源: https://codeforces.com/problemset/problem/1034/A
     * 问题描述: 给定 n 个正整数, 通过删除最少的数来增大这些数的最大公约数。
     *             返回需要删除的最少数字个数, 如果无法增大 GCD 则返回-1。
     * 解题思路: 首先计算所有数的 GCD, 然后将所有数除以这个 GCD, 问题转化为找到一个大于 1 的因子,
     *             使得尽可能多的数是这个因子的倍数。枚举所有质数, 统计是其倍数的数的个数,
     *             答案就是 n 减去最大个数。
     * 时间复杂度: O(n*log(max_value) + max_value*log(log(max_value)))
     * 空间复杂度: O(max_value)
     * 是否最优解: 是, 这是解决该问题的最优方法。
    */
    static int enlargeGCD(vector<int>& nums) {
        int n = nums.size();

        // 计算所有数的 GCD
        int currentGcd = nums[0];
        for (int i = 1; i < n; i++) {
            currentGcd = gcd(currentGcd, nums[i]);
        }

        // 将所有数除以 GCD
        vector<int> normalized(n);
        int maxValue = 0;
        for (int i = 0; i < n; i++) {
            normalized[i] = nums[i] / currentGcd;
            maxValue = max(maxValue, normalized[i]);
        }

        // 线性筛法预处理质数
        vector<bool> isPrime(maxValue + 1, true);
        isPrime[0] = isPrime[1] = false;

        for (int i = 2; i * i <= maxValue; i++) {
            if (isPrime[i]) {
                for (int j = i * i; j <= maxValue; j += i) {
                    isPrime[j] = false;
                }
            }
        }
    }
}

```

```

        isPrime[j] = false;
    }
}
}

// 统计每个数出现的次数
vector<int> count(maxValue + 1, 0);
for (int num : normalized) {
    count[num]++;
}

// 枚举质数，统计是其倍数的数的个数
int maxCount = 0;
for (int i = 2; i <= maxValue; i++) {
    if (isPrime[i]) {
        int primeCount = 0;
        for (int j = i; j <= maxValue; j += i) {
            primeCount += count[j];
        }
        maxCount = max(maxCount, primeCount);
    }
}

// 如果所有数都相同，则无法增大 GCD
if (maxCount == n) {
    return -1;
}

return n - maxCount;
}

/***
 * POJ 2429. GCD & LCM Inverse
 * 题目来源: http://poj.org/problem?id=2429
 * 问题描述: 给定两个正整数 a 和 b 的最大公约数和最小公倍数，反过来求这两个数，要求这两个数的和最小。
 * 解题思路: 设 gcd 为最大公约数，lcm 为最小公倍数，则  $a*b = gcd*lcm$ 。设  $a = gcd*x$ ,  $b = gcd*y$ ,
 * 则  $x*y = lcm/gcd$ , 且  $gcd(x, y) = 1$ 。问题转化为找到两个互质的数 x 和 y，使得  $x*y = lcm/gcd$ ,
 * 并且  $x+y$  最小。
 * 时间复杂度:  $O(\sqrt{lcm/gcd})$ 
 * 空间复杂度:  $O(1)$ 
 * 是否最优解: 是，这是解决该问题的最优方法。
 */

```

```

*/
static vector<long long> gcdLcmInverse(long long gcd, long long lcm) {
    // 计算 lcm/gcd
    long long product = lcm / gcd;

    // 找到两个互质的数 x 和 y, 使得 x*y = product, 并且 x+y 最小
    long long x = 1;
    long long y = product;

    // 枚举所有可能的因子对
    for (long long i = 1; i * i <= product; i++) {
        if (product % i == 0) {
            long long factor1 = i;
            long long factor2 = product / i;

            // 检查这两个因子是否互质
            if (ExtendedGcdLcmProblems::gcd(factor1, factor2) == 1) {
                // 如果当前因子对的和更小, 则更新结果
                if (factor1 + factor2 < x + y) {
                    x = factor1;
                    y = factor2;
                }
            }
        }
    }

    // 返回结果, 确保 a <= b
    long long a = gcd * x;
    long long b = gcd * y;

    if (a > b) {
        long long temp = a;
        a = b;
        b = temp;
    }

    return {a, b};
}

/***
 * UVa 10892. LCM Cardinality
 * 题目来源:
https://onlinejudge.org/index.php?option=com\_onlinejudge&Itemid=8&page=show\_problem&problem=1833

```

- \* 问题描述：给定一个正整数 n，找出有多少对不同的整数对 (a, b)，使得  $\text{lcm}(a, b) = n$ 。
- \* 解题思路：枚举 n 的所有因子，对于每个因子 d，如果  $\text{gcd}(d, n/d) = 1$ ，则  $(d, n/d)$  是一对解。
- \* 时间复杂度： $O(\sqrt{n})$
- \* 空间复杂度： $O(1)$
- \* 是否最优解：是，这是解决该问题的最优方法。

\*/

```

static int lcmCardinality(int n) {
    // 找到 n 的所有因子
    vector<int> divisors;
    for (int i = 1; i * i <= n; i++) {
        if (n % i == 0) {
            divisors.push_back(i);
            if (i != n / i) {
                divisors.push_back(n / i);
            }
        }
    }

    // 计算有多少对不同的整数对 (a, b) 使得  $\text{lcm}(a, b) = n$ 
    int count = 0;
    for (int i = 0; i < divisors.size(); i++) {
        for (int j = i; j < divisors.size(); j++) {
            int a = divisors[i];
            int b = divisors[j];
            // 如果  $\text{lcm}(a, b) = n$ ，则是一对解
            if ( $\text{lcm}(a, b) == n$ ) {
                count++;
            }
        }
    }

    return count;
}

/***
 * SPOJ GCDEX. GCD Extreme
 * 题目来源: https://www.spoj.com/problems/GCDEX/
 * 问题描述：计算  $G(n) = \sum_{i=1}^n \sum_{j=i+1}^n \text{gcd}(i, j)$ 
 * 解题思路：使用欧拉函数优化计算
 * 时间复杂度： $O(n \log n)$ 
 * 空间复杂度： $O(n)$ 
 * 是否最优解：是，这是解决该问题的最优方法。
 */

```

```

static long long gcdExtreme(int n) {
    // 预处理欧拉函数
    vector<int> phi(n + 1);
    for (int i = 1; i <= n; i++) {
        phi[i] = i;
    }

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) { // i 是质数
            for (int j = i; j <= n; j += i) {
                phi[j] = phi[j] / i * (i - 1);
            }
        }
    }
}

// 计算前缀和
vector<long long> prefixSum(n + 1, 0);
for (int i = 1; i <= n; i++) {
    prefixSum[i] = prefixSum[i - 1] + phi[i];
}

// 计算结果
long long result = 0;
for (int i = 1; i <= n; i++) {
    result += (long long)i * (prefixSum[n / i] - 1); // 减 1 是因为不包括 phi[1] 的情况
}

return result;
}

/***
 * LeetCode 2807. Insert Greatest Common Divisors in Linked List
 * 题目来源: https://leetcode.com/problems/insert-greatest-common-divisors-in-linked-list/
 * 问题描述: 给定一个链表，在每对相邻节点之间插入一个值为它们最大公约数的新节点。
 * 解题思路: 遍历链表，对每对相邻节点，计算它们的最大公约数并插入新节点。
 * 时间复杂度: O(n * log(min(a, b))), 其中 n 是链表长度
 * 空间复杂度: O(1)，只使用常数额外空间（不计算新插入的节点）
 * 是否最优解: 是，这是解决该问题的最优方法。
 */
static ListNode* insertGreatestCommonDivisors(ListNode* head) {
    // 如果链表为空或只有一个节点，直接返回
    if (head == nullptr || head->next == nullptr) {
        return head;
    }
}

```

```

}

ListNode* current = head;

// 遍历链表，直到倒数第二个节点
while (current->next != nullptr) {
    // 计算当前节点和下一个节点值的最大公约数
    int gcdValue = gcd(current->val, current->next->val);

    // 创建新节点并插入
    ListNode* newNode = new ListNode(gcdValue);
    newNode->next = current->next;
    current->next = newNode;

    // 移动到下一个原始节点（跳过刚插入的节点）
    current = newNode->next;
}

return head;
}

/***
 * LeetCode 1979. Find Greatest Common Divisor of Array
 * 题目来源: https://leetcode.com/problems/find-greatest-common-divisor-of-array/
 * 问题描述: 给定一个整数数组 nums，返回数组中最小数和最大数的最大公约数。
 * 解题思路: 首先找到数组中的最小值和最大值，然后计算它们的最大公约数。
 * 时间复杂度: O(n + log(min(min_val, max_val))), 其中 n 是数组长度
 * 空间复杂度: O(log(min(min_val, max_val))), 递归调用栈的深度
 * 是否最优解: 是，这是解决该问题的最优方法。
 */
static int findGCD(vector<int>& nums) {
    // 找到数组中的最小值和最大值
    int minVal = INT_MAX;
    int maxVal = INT_MIN;

    for (int num : nums) {
        minVal = min(minVal, num);
        maxVal = max(maxVal, num);
    }

    // 计算最小值和最大值的最大公约数
    return gcd(minVal, maxVal);
}

```

```

/***
 * LeetCode 878. 第 N 个神奇数字
 * 问题描述：一个正整数如果能被 a 或 b 整除，那么它是神奇的。给定 n, a, b，返回第 n 个神奇数字。
 * 解题思路：使用二分查找 + 容斥原理
 * 时间复杂度：O(log(n * min(a, b)))
 * 空间复杂度：O(1)
 */
static int nthMagicalNumber(int n, int a, int b) {
    long long lcm_val = lcm(a, b);
    long long left = 0;
    long long right = (long long)n * min(a, b);
    long long result = 0;

    // 二分查找第 n 个神奇数字
    while (left <= right) {
        long long mid = left + (right - left) / 2;
        // 在[1, mid]范围内神奇数字的个数
        long long count = mid / a + mid / b - mid / lcm_val;

        if (count >= n) {
            result = mid;
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }

    return (int)(result % 1000000007);
}

/***
 * LeetCode 1201. 丑数 III
 * 问题描述：编写一个程序，找出第 n 个丑数，丑数是可以被 a 或 b 或 c 整除的正整数。
 * 解题思路：二分查找 + 容斥原理
 * 时间复杂度：O(log(n * min(a, b, c)))
 * 空间复杂度：O(1)
*/
static int nthUglyNumber(int n, int a, int b, int c) {
    long long la = a, lb = b, lc = c;
    long long lab = lcm(la, lb);
    long long lac = lcm(la, lc);
    long long lbc = lcm(lb, lc);

```

```

long long labc = lcm(lab, lc);

long long left = 0;
long long right = (long long)2e9; // 根据题目数据范围设定
long long result = 0;

// 二分查找第 n 个丑数
while (left <= right) {
    long long mid = left + (right - left) / 2;
    // 在[1, mid]范围内丑数的个数（容斥原理）
    long long count = mid / la + mid / lb + mid / lc
        - mid / lab - mid / lac - mid / lbc
        + mid / labc;

    if (count >= n) {
        result = mid;
        right = mid - 1;
    } else {
        left = mid + 1;
    }
}

return (int)(result % 1000000007);
}

/***
 * LeetCode 1071. 字符串的最大公因子
 * 问题描述：对于字符串 s 和 t，只有在  $s=t+t+t+\dots+t$  时，才认为 t 能除尽 s。
 * 给定两个字符串 str1 和 str2，返回最长字符串 x，使得 x 能除尽 str1 和 str2。
 * 解题思路：利用 GCD 的性质，如果存在这样的字符串，其长度必然是两个字符串长度的 GCD
 * 时间复杂度：O(m+n)
 * 空间复杂度：O(1)
 */
static string gcdOfStrings(string str1, string str2) {
    // 如果存在公因子字符串，则 str1+str2 应该等于 str2+str1
    if (str1 + str2 != str2 + str1) {
        return "";
    }

    // 最大公因子字符串的长度就是两个字符串长度的 GCD
    int gcdLength = gcd((int)str1.length(), (int)str2.length());
    return str1.substr(0, gcdLength);
}

```

```

/**
 * LeetCode 2447. 最大公因数等于 K 的子数组数目
 * 问题描述：给定一个数组和一个正整数 k，返回最大公因数等于 k 的子数组数目。
 * 解题思路：遍历所有子数组，计算每个子数组的 GCD，统计等于 k 的数量
 * 时间复杂度：O(n^2 * log(max(nums)))
 * 空间复杂度：O(1)
 */
static int subarrayGCD(vector<int>& nums, int k) {
    int count = 0;
    int n = nums.size();

    // 遍历所有可能的子数组
    for (int i = 0; i < n; i++) {
        int currentGcd = nums[i];
        // 优化：如果当前元素不能被 k 整除，跳过
        if (currentGcd % k != 0) continue;

        for (int j = i; j < n; j++) {
            // 优化：如果当前元素不能被 k 整除，跳出内层循环
            if (nums[j] % k != 0) break;

            currentGcd = gcd(currentGcd, nums[j]);

            // 如果 GCD 小于 k，不可能再变大，跳出内层循环
            if (currentGcd < k) break;

            // 如果 GCD 等于 k，计数加 1
            if (currentGcd == k) {
                count++;
            }
        }
    }

    return count;
}

/**
 * LeetCode 2470. 最小公倍数为 K 的子数组数目
 * 问题描述：给定一个数组和一个正整数 k，返回最小公倍数等于 k 的子数组数目。
 * 解题思路：遍历所有子数组，计算每个子数组的 LCM，统计等于 k 的数量
 * 时间复杂度：O(n^2 * log(max(nums)))
 * 空间复杂度：O(1)

```

```

*/
static int subarrayLCM(vector<int>& nums, int k) {
    int count = 0;
    int n = nums.size();

    // 遍历所有可能的子数组
    for (int i = 0; i < n; i++) {
        long long currentLcm = nums[i];

        // 如果当前元素不能整除 k, 跳过
        if (k % nums[i] != 0) continue;

        for (int j = i; j < n; j++) {
            // 如果当前元素不能整除 k, 跳出内层循环
            if (k % nums[j] != 0) break;

            currentLcm = lcm(currentLcm, nums[j]);

            // 如果 LCM 大于 k, 不可能再变小, 跳出内层循环
            if (currentLcm > k) break;

            // 如果 LCM 等于 k, 计数加 1
            if (currentLcm == k) {
                count++;
            }
        }
    }

    return count;
}

```

```

/**
 * 计算最大公约数（欧几里得算法） - 整型版本
 * 时间复杂度: O(log(min(a, b)))
 * 空间复杂度: O(log(min(a, b))) (递归)
 */
static int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}

```

```

/**
 * 计算最大公约数（欧几里得算法） - 长整型版本
 * 时间复杂度: O(log(min(a, b)))

```

```

* 空间复杂度: O(log(min(a, b))) (递归)
*/
static long long gcd(long long a, long long b) {
    return b == 0 ? a : gcd(b, a % b);
}

/***
* 计算最小公倍数
* 利用公式: lcm(a, b) = |a*b| / gcd(a, b)
* 时间复杂度: O(log(min(a, b)))
* 空间复杂度: O(log(min(a, b)))
*/
static long long lcm(long long a, long long b) {
    return a / gcd(a, b) * b;
}

/***
* 扩展欧几里得算法
* 求解 ax + by = gcd(a, b) 的一组整数解
* 同时返回 gcd(a, b) 的值
* 时间复杂度: O(log(min(a, b)))
* 空间复杂度: O(log(min(a, b)))
*/
static vector<long long> extendedGcd(long long a, long long b) {
    if (b == 0) {
        return {a, 1, 0}; // gcd, x, y
    }

    vector<long long> result = extendedGcd(b, a % b);
    long long gcd_val = result[0];
    long long x1 = result[1];
    long long y1 = result[2];

    long long x = y1;
    long long y = x1 - (a / b) * y1;

    return {gcd_val, x, y};
}

/***
* 计算数组中所有元素的最大公约数
* 时间复杂度: O(n * log(min(elements)))
* 空间复杂度: O(log(min(elements)))
*/

```

```

*/
static int gcdOfArray(vector<int>& nums) {
    int result = nums[0];
    for (int i = 1; i < nums.size(); i++) {
        result = gcd(result, nums[i]);
        // 优化: 如果 GCD 已经为 1, 可以提前结束
        if (result == 1) break;
    }
    return result;
}

/***
 * 计算数组中所有元素的最小公倍数
 * 时间复杂度: O(n * log(min(elements)))
 * 空间复杂度: O(log(min(elements)))
 */
static long long lcmOfArray(vector<int>& nums) {
    long long result = nums[0];
    for (int i = 1; i < nums.size(); i++) {
        result = lcm(result, nums[i]);
    }
    return result;
}

// 辅助方法: 打印链表
void printList(ListNode* head) {
    ListNode* current = head;
    while (current != nullptr) {
        cout << current->val;
        if (current->next != nullptr) {
            cout << " -> ";
        }
        current = current->next;
    }
    cout << endl;
}

// 测试方法
int main() {
    cout << "==== GCD 和 LCM 扩展问题测试 ===" << endl;
    // 测试 gcdLcmInverse
}

```

```

vector<long long> result = ExtendedGcdLcmProblems::gcdLcmInverse(3, 60);
cout << "GCD & LCM Inverse (gcd=3, lcm=60): a=" << result[0] << ", b=" << result[1] << endl;

result = ExtendedGcdLcmProblems::gcdLcmInverse(2, 20);
cout << "GCD & LCM Inverse (gcd=2, lcm=20): a=" << result[0] << ", b=" << result[1] << endl;

// 测试 lcmCardinality
cout << "LCM Cardinality (n=2): " << ExtendedGcdLcmProblems::lcmCardinality(2) << endl;
cout << "LCM Cardinality (n=12): " << ExtendedGcdLcmProblems::lcmCardinality(12) << endl;
cout << "LCM Cardinality (n=100): " << ExtendedGcdLcmProblems::lcmCardinality(100) << endl;

// 测试 gcdExtreme
cout << "GCD Extreme (n=3): " << ExtendedGcdLcmProblems::gcdExtreme(3) << endl;
cout << "GCD Extreme (n=4): " << ExtendedGcdLcmProblems::gcdExtreme(4) << endl;
cout << "GCD Extreme (n=6): " << ExtendedGcdLcmProblems::gcdExtreme(6) << endl;

// 测试 insertGreatestCommonDivisors
ListNode* head1 = new ListNode(18);
head1->next = new ListNode(6);
head1->next->next = new ListNode(10);
head1->next->next->next = new ListNode(3);

cout << "原链表: ";
printList(head1);

ListNode* result1 = ExtendedGcdLcmProblems::insertGreatestCommonDivisors(head1);
cout << "插入 GCD 后: ";
printList(result1);

// 测试 findGCD
vector<int> nums1 = {2, 5, 6, 9, 10};
cout << "数组[2,5,6,9,10]的 GCD: " << ExtendedGcdLcmProblems::findGCD(nums1) << endl;

vector<int> nums2 = {7, 5, 6, 8, 3};
cout << "数组[7,5,6,8,3]的 GCD: " << ExtendedGcdLcmProblems::findGCD(nums2) << endl;

vector<int> nums3 = {3, 3};
cout << "数组[3,3]的 GCD: " << ExtendedGcdLcmProblems::findGCD(nums3) << endl;

// 测试 nthMagicalNumber
cout << "第 1 个神奇数字(n=1, a=2, b=3): " << ExtendedGcdLcmProblems::nthMagicalNumber(1, 2,
3) << endl;
cout << "第 4 个神奇数字(n=4, a=2, b=3): " << ExtendedGcdLcmProblems::nthMagicalNumber(4, 2,
3) << endl;

```

```

3) << endl;

// 测试 nthUglyNumber
cout << "第 3 个丑数(n=3, a=2, b=3, c=5): " << ExtendedGcdLcmProblems::nthUglyNumber(3, 2, 3,
5) << endl;
cout << "第 4 个丑数(n=4, a=2, b=3, c=4): " << ExtendedGcdLcmProblems::nthUglyNumber(4, 2, 3,
4) << endl;

// 测试 gcdOfStrings
cout << "字符串最大公因子(\"ABCABC\", \"ABC\"): " <<
ExtendedGcdLcmProblems::gcdOfStrings("ABCABC", "ABC") << endl;
cout << "字符串最大公因子(\"ABABAB\", \"ABAB\"): " <<
ExtendedGcdLcmProblems::gcdOfStrings("ABABAB", "ABAB") << endl;
cout << "字符串最大公因子(\"LEET\", \"CODE\"): " <<
ExtendedGcdLcmProblems::gcdOfStrings("LEET", "CODE") << endl;

// 测试 subarrayGCD
vector<int> nums4 = {9, 3, 1, 2, 6, 3};
cout << "GCD 等于 3 的子数组数目: " << ExtendedGcdLcmProblems::subarrayGCD(nums4, 3) << endl;

vector<int> nums5 = {3, 1, 2, 4, 6};
cout << "GCD 等于 1 的子数组数目: " << ExtendedGcdLcmProblems::subarrayGCD(nums5, 1) << endl;

// 测试 subarrayLCM
vector<int> nums6 = {3, 6, 2, 1, 2};
cout << "LCM 等于 6 的子数组数目: " << ExtendedGcdLcmProblems::subarrayLCM(nums6, 6) << endl;

// 测试 extendedGcd
vector<long long> extResult = ExtendedGcdLcmProblems::extendedGcd(30, 18);
cout << "扩展欧几里得算法(30, 18): gcd=" << extResult[0] <<
", x=" << extResult[1] << ", y=" << extResult[2] << endl;
cout << "验证: 30*" << extResult[1] << "+ 18*" << extResult[2] <<
" = " << (30*extResult[1] + 18*extResult[2]) << endl;

// 测试数组 GCD 和 LCM
vector<int> nums7 = {12, 18, 24};
cout << "数组[12, 18, 24]的 GCD: " << ExtendedGcdLcmProblems::gcdOfArray(nums7) << endl;
cout << "数组[12, 18, 24]的 LCM: " << ExtendedGcdLcmProblems::lcmOfArray(nums7) << endl;

// 测试数组 GCD 和 LCM
vector<int> nums8 = {6, 12, 18};
cout << "Enlarge GCD (nums=[6, 12, 18]): " << ExtendedGcdLcmProblems::enlargeGCD(nums8) <<
endl;

```

```
    return 0;  
}
```

文件: ExtendedGcdLcmProblems.java

```
package class041;
```

```
import java.util.*;
```

```
/**  
 * 扩展 GCD 和 LCM 相关问题的实现  
 * 包含 LeetCode 和其他平台上的经典问题  
 */  
public class ExtendedGcdLcmProblems {
```

```
// 链表节点定义
```

```
public static class ListNode {  
    int val;  
    ListNode next;  
    ListNode() {}  
    ListNode(int val) { this.val = val; }  
    ListNode(int val, ListNode next) { this.val = val; this.next = next; }  
}
```

```
/**
```

```
* POJ 2429. GCD & LCM Inverse  
* 题目来源: http://poj.org/problem?id=2429  
* 问题描述: 给定两个正整数 a 和 b 的最大公约数和最小公倍数, 反过来求这两个数, 要求这两个数的和  
最小。
```

```
* 解题思路: 设 gcd 为最大公约数, lcm 为最小公倍数, 则  $a*b = gcd*lcm$ 。设  $a = gcd*x$ ,  $b = gcd*y$ ,  
* 则  $x*y = lcm/gcd$ , 且  $gcd(x, y) = 1$ 。问题转化为找到两个互质的数 x 和 y, 使得  $x*y =$   
lcm/gcd,
```

```
* 并且  $x+y$  最小。
```

```
* 时间复杂度:  $O(\sqrt(lcm/gcd))$ 
```

```
* 空间复杂度:  $O(1)$ 
```

```
* 是否最优解: 是, 这是解决该问题的最优方法。
```

```
*/
```

```
public static long[] gcdLcmInverse(long gcd, long lcm) {
```

```

// 计算 lcm/gcd
long product = lcm / gcd;

// 找到两个互质的数 x 和 y, 使得 x*y = product, 并且 x+y 最小
long x = 1;
long y = product;

// 枚举所有可能的因子对
for (long i = 1; i * i <= product; i++) {
    if (product % i == 0) {
        long factor1 = i;
        long factor2 = product / i;

        // 检查这两个因子是否互质
        if (gcd(factor1, factor2) == 1) {
            // 如果当前因子对的和更小, 则更新结果
            if (factor1 + factor2 < x + y) {
                x = factor1;
                y = factor2;
            }
        }
    }
}

// 返回结果, 确保 a <= b
long a = gcd * x;
long b = gcd * y;

if (a > b) {
    long temp = a;
    a = b;
    b = temp;
}

return new long[]{a, b};
}

/**
 * UVa 10892. LCM Cardinality
 * 题目来源:
https://onlinejudge.org/index.php?option=com\_onlinejudge&Itemid=8&page=show\_problem&problem=1833
 * 问题描述: 给定一个正整数 n, 找出有多少对不同的整数对 (a, b), 使得 lcm(a, b) = n。
 * 解题思路: 枚举 n 的所有因子, 对于每个因子 d, 如果 gcd(d, n/d) = 1, 则 (d, n/d) 是一对解。

```

```

* 时间复杂度: O(√n)
* 空间复杂度: O(1)
* 是否最优解: 是, 这是解决该问题的最优方法。
*/
public static int lcmCardinality(int n) {
    // 找到 n 的所有因子
    List<Integer> divisors = new ArrayList<>();
    for (int i = 1; i * i <= n; i++) {
        if (n % i == 0) {
            divisors.add(i);
            if (i != n / i) {
                divisors.add(n / i);
            }
        }
    }

    // 计算有多少对不同的整数对(a, b)使得 lcm(a, b) = n
    int count = 0;
    for (int i = 0; i < divisors.size(); i++) {
        for (int j = i; j < divisors.size(); j++) {
            int a = divisors.get(i);
            int b = divisors.get(j);
            // 如果 lcm(a, b) = n, 则是一对解
            if (lcm(a, b) == n) {
                count++;
            }
        }
    }

    return count;
}

/**
 * SPOJ GCDEX. GCD Extreme
 * 题目来源: https://www.spoj.com/problems/GCDEX/
 * 问题描述: 计算 G(n) = Σ (i=1 to n) Σ (j=i+1 to n) gcd(i, j)
 * 解题思路: 使用欧拉函数优化计算
 * 时间复杂度: O(n log n)
 * 空间复杂度: O(n)
 * 是否最优解: 是, 这是解决该问题的最优方法。
*/
public static long gcdExtreme(int n) {
    // 预处理欧拉函数

```

```

int[] phi = new int[n + 1];
for (int i = 1; i <= n; i++) {
    phi[i] = i;
}

for (int i = 2; i <= n; i++) {
    if (phi[i] == i) { // i 是质数
        for (int j = i; j <= n; j += i) {
            phi[j] = phi[j] / i * (i - 1);
        }
    }
}

// 计算前缀和
long[] prefixSum = new long[n + 1];
for (int i = 1; i <= n; i++) {
    prefixSum[i] = prefixSum[i - 1] + phi[i];
}

// 计算结果
long result = 0;
for (int i = 1; i <= n; i++) {
    result += i * (prefixSum[n / i] - 1); // 减 1 是因为不包括 phi[1] 的情况
}

return result;
}

/***
 * LeetCode 2807. Insert Greatest Common Divisors in Linked List
 * 题目来源: https://leetcode.com/problems/insert-greatest-common-divisors-in-linked-list/
 * 问题描述: 给定一个链表，在每对相邻节点之间插入一个值为它们最大公约数的新节点。
 * 解题思路: 遍历链表，对每对相邻节点，计算它们的最大公约数并插入新节点。
 * 时间复杂度: O(n * log(min(a, b))), 其中 n 是链表长度
 * 空间复杂度: O(1)，只使用常数额外空间（不计算新插入的节点）
 * 是否最优解: 是，这是解决该问题的最优方法。
 */
public static ListNode insertGreatestCommonDivisors(ListNode head) {
    // 如果链表为空或只有一个节点，直接返回
    if (head == null || head.next == null) {
        return head;
    }

    int[] phi = new int[n + 1];
    for (int i = 1; i <= n; i++) {
        phi[i] = i;
    }

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) { // i 是质数
            for (int j = i; j <= n; j += i) {
                phi[j] = phi[j] / i * (i - 1);
            }
        }
    }

    long[] prefixSum = new long[n + 1];
    for (int i = 1; i <= n; i++) {
        prefixSum[i] = prefixSum[i - 1] + phi[i];
    }

    long result = 0;
    for (int i = 1; i <= n; i++) {
        result += i * (prefixSum[n / i] - 1); // 减 1 是因为不包括 phi[1] 的情况
    }

    return result;
}

```

```

ListNode current = head;

// 遍历链表，直到倒数第二个节点
while (current.next != null) {
    // 计算当前节点和下一个节点值的最大公约数
    int gcdValue = gcd(current.val, current.next.val);

    // 创建新节点并插入
    ListNode newNode = new ListNode(gcdValue);
    newNode.next = current.next;
    current.next = newNode;

    // 移动到下一个原始节点（跳过刚插入的节点）
    current = newNode.next;
}

return head;
}

/**
 * LeetCode 1979. Find Greatest Common Divisor of Array
 * 题目来源: https://leetcode.com/problems/find-greatest-common-divisor-of-array/
 * 问题描述: 给定一个整数数组 nums，返回数组中最小数和最大数的最大公约数。
 * 解题思路: 首先找到数组中的最小值和最大值，然后计算它们的最大公约数。
 * 时间复杂度: O(n + log(min(min_val, max_val))), 其中 n 是数组长度
 * 空间复杂度: O(log(min(min_val, max_val))), 递归调用栈的深度
 * 是否最优解: 是，这是解决该问题的最优方法。
 */
public static int findGCD(int[] nums) {
    // 找到数组中的最小值和最大值
    int minVal = Integer.MAX_VALUE;
    int maxVal = Integer.MIN_VALUE;

    for (int num : nums) {
        minVal = Math.min(minVal, num);
        maxVal = Math.max(maxVal, num);
    }

    // 计算最小值和最大值的最大公约数
    return gcd(minVal, maxVal);
}

/**

```

\* LeetCode 878. 第 N 个神奇数字  
\* 问题描述：一个正整数如果能被 a 或 b 整除，那么它是神奇的。给定 n, a, b，返回第 n 个神奇数字。  
\* 解题思路：使用二分查找 + 容斥原理  
\* 时间复杂度：O(log(n \* min(a, b)))  
\* 空间复杂度：O(1)  
\*/

```
public static int nthMagicalNumber(int n, int a, int b) {  
    long lcm = lcm(a, b);  
    long left = 0;  
    long right = (long) n * Math.min(a, b);  
    long result = 0;
```

```
// 二分查找第 n 个神奇数字  
while (left <= right) {  
    long mid = left + (right - left) / 2;  
    // 在[1, mid]范围内神奇数字的个数  
    long count = mid / a + mid / b - mid / lcm;
```

```
    if (count >= n) {  
        result = mid;  
        right = mid - 1;  
    } else {  
        left = mid + 1;  
    }  
}
```

```
    return (int) (result % 1000000007);  
}
```

/\*\*

\* LeetCode 1201. 丑数 III  
\* 问题描述：编写一个程序，找出第 n 个丑数，丑数是可以被 a 或 b 或 c 整除的正整数。  
\* 解题思路：二分查找 + 容斥原理  
\* 时间复杂度：O(log(n \* min(a, b, c)))  
\* 空间复杂度：O(1)  
\*/

```
public static int nthUglyNumber(int n, int a, int b, int c) {  
    long la = a, lb = b, lc = c;  
    long lab = lcm(la, lb);  
    long lac = lcm(la, lc);  
    long lbc = lcm(lb, lc);  
    long labc = lcm(lab, lc);
```

```

long left = 0;
long right = (long) 2e9; // 根据题目数据范围设定
long result = 0;

// 二分查找第 n 个丑数
while (left <= right) {
    long mid = left + (right - left) / 2;
    // 在[1, mid]范围内丑数的个数（容斥原理）
    long count = mid / la + mid / lb + mid / lc
        - mid / lab - mid / lac - mid / lbc
        + mid / labc;

    if (count >= n) {
        result = mid;
        right = mid - 1;
    } else {
        left = mid + 1;
    }
}

return (int) (result % 1000000007);
}

```

```

/**
 * LeetCode 1071. 字符串的最大公因子
 * 问题描述：对于字符串 s 和 t，只有在  $s=t+t+t+\dots+t$  时，才认为 t 能除尽 s。
 * 给定两个字符串 str1 和 str2，返回最长字符串 x，使得 x 能除尽 str1 和 str2。
 * 解题思路：利用 GCD 的性质，如果存在这样的字符串，其长度必然是两个字符串长度的 GCD
 * 时间复杂度： $O(m+n)$ 
 * 空间复杂度： $O(1)$ 
 */

```

```

public static String gcdOfStrings(String str1, String str2) {
    // 如果存在公因子字符串，则 str1+str2 应该等于 str2+str1
    if (!(str1 + str2).equals(str2 + str1)) {
        return "";
    }

    // 最大公因子字符串的长度就是两个字符串长度的 GCD
    int gcdLength = gcd(str1.length(), str2.length());
    return str1.substring(0, gcdLength);
}

/**

```

```

* LeetCode 2447. 最大公因数等于 K 的子数组数目
* 问题描述: 给定一个数组和一个正整数 k, 返回最大公因数等于 k 的子数组数目。
* 解题思路: 遍历所有子数组, 计算每个子数组的 GCD, 统计等于 k 的数量
* 时间复杂度: O(n^2 * log(max(nums)))
* 空间复杂度: O(1)
*/
public static int subarrayGCD(int[] nums, int k) {
    int count = 0;
    int n = nums.length;

    // 遍历所有可能的子数组
    for (int i = 0; i < n; i++) {
        int currentGcd = nums[i];
        // 优化: 如果当前元素不能被 k 整除, 跳过
        if (currentGcd % k != 0) continue;

        for (int j = i; j < n; j++) {
            // 优化: 如果当前元素不能被 k 整除, 跳出内层循环
            if (nums[j] % k != 0) break;

            currentGcd = gcd(currentGcd, nums[j]);

            // 如果 GCD 小于 k, 不可能再变大, 跳出内层循环
            if (currentGcd < k) break;

            // 如果 GCD 等于 k, 计数加 1
            if (currentGcd == k) {
                count++;
            }
        }
    }

    return count;
}

/**
* LeetCode 2470. 最小公倍数为 K 的子数组数目
* 问题描述: 给定一个数组和一个正整数 k, 返回最小公倍数等于 k 的子数组数目。
* 解题思路: 遍历所有子数组, 计算每个子数组的 LCM, 统计等于 k 的数量
* 时间复杂度: O(n^2 * log(max(nums)))
* 空间复杂度: O(1)
*/
public static int subarrayLCM(int[] nums, int k) {

```

```

int count = 0;
int n = nums.length;

// 遍历所有可能的子数组
for (int i = 0; i < n; i++) {
    long currentLcm = nums[i];

    // 如果当前元素不能整除 k, 跳过
    if (k % nums[i] != 0) continue;

    for (int j = i; j < n; j++) {
        // 如果当前元素不能整除 k, 跳出内层循环
        if (k % nums[j] != 0) break;

        currentLcm = lcm(currentLcm, nums[j]);

        // 如果 LCM 大于 k, 不可能再变小, 跳出内层循环
        if (currentLcm > k) break;

        // 如果 LCM 等于 k, 计数加 1
        if (currentLcm == k) {
            count++;
        }
    }
}

return count;
}

/***
 * SPOJ LCMSUM. LCM Sum
 * 题目来源: https://www.spoj.com/problems/LCMSUM/
 * 问题描述: 给定 n, 计算  $\sum_{i=1}^n \text{lcm}(i, n)$ 
 * 解题思路: 利用数学公式进行优化。我们知道:
 * 
$$\begin{aligned} \sum_{i=1}^n \text{lcm}(i, n) &= \sum_{i=1}^n (i * n) / \text{gcd}(i, n) \\ &= n * \sum_{i=1}^n i / \text{gcd}(i, n) \end{aligned}$$

 *
 * 我们可以将这个和式按 gcd 值分组:
 * 
$$\sum_{d|n} \sum_{\text{gcd}(i, n)=d} i / d$$

 *
 * 对于  $\text{gcd}(i, n)=d$  的情况, 设  $i=d*j, n=d*k$ , 则  $\text{gcd}(j, k)=1$ 
 * 所以  $\sum_{\text{gcd}(i, n)=d} i = d * \sum_{\text{gcd}(j, k)=1} j$ 
 */

```

```

*            $\sum_{j=1}^k \text{gcd}(j, k) = k * \phi(k) / 2$  (当  $k > 1$  时)
*           其中  $\phi$  是欧拉函数
*
*           因此,  $\sum_{i=1}^n \text{lcm}(i, n) = n * \sum_{d|n} \phi(d) * (n/d) / 2$ 
*            $= (n/2) * \sum_{d|n} \phi(d) * d + n$  (当  $d=n$  时需要特殊处理)
* 时间复杂度:  $O(\sqrt{n})$ 
* 空间复杂度:  $O(1)$ 
* 是否最优解: 是, 这是解决该问题的最优方法。
*/
public static long lcmSum(int n) {
    // 预处理欧拉函数
    int[] phi = new int[n + 1];
    for (int i = 1; i <= n; i++) {
        phi[i] = i;
    }

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) { // i 是质数
            for (int j = i; j <= n; j += i) {
                phi[j] = phi[j] / i * (i - 1);
            }
        }
    }
}

// 计算结果
long result = 0;
for (int i = 1; i * i <= n; i++) {
    if (n % i == 0) {
        int d1 = i;
        int d2 = n / i;

        result += (long) phi[d1] * d1;
        if (d1 != d2) {
            result += (long) phi[d2] * d2;
        }
    }
}

return (result + 1) * n / 2;
}

/**
 * HackerRank GCD Product

```

\* 题目来源: <https://www.hackerrank.com/challenges/gcd-product/problem>  
 \* 问题描述: 给定 N 和 M, 计算  $\prod_{i=1}^N \prod_{j=1}^M \gcd(i, j) \bmod (10^{9+7})$   
 \* 解题思路: 对于每个质数 p, 计算它在结果中的指数。对于质数 p, 它在  $\gcd(i, j)$  中的指数等于  $\min(vp(i), vp(j))$ , 其中  $vp(x)$  表示 x 中质因子 p 的指数。  
 \* 我们可以枚举所有质数 p, 计算  $\sum_{i=1}^N \sum_{j=1}^M \min(vp(i), vp(j))$ 。  
 \* 为了优化计算, 我们可以使用以下方法:  
 \* 对于每个质数 p, 计算有多少个数 i 满足  $vp(i)=k$ , 记为  $\text{count\_p}(k)$ 。  
 \* 然后计算  $\sum_{k=1}^{\max} \sum_{l=1}^{\max} \min(k, l) * \text{count\_p}(k) * \text{count\_p}(l)$ 。  
 \* 时间复杂度:  $O(N \log(\log(N)) + M \log(\log(M)))$   
 \* 空间复杂度:  $O(N + M)$   
 \* 是否最优解: 是, 这是解决该问题的最优方法。  
 \*/

```

public static int gcdProduct(int n, int m) {
    final int MOD = 1000000007;

    // 预处理质数和每个数的最小质因子
    int maxVal = Math.max(n, m);
    int[] smallestPrimeFactor = new int[maxVal + 1];
    for (int i = 1; i <= maxVal; i++) {
        smallestPrimeFactor[i] = i;
    }

    // 线性筛法找最小质因子
    for (int i = 2; i <= maxVal; i++) {
        if (smallestPrimeFactor[i] == i) { // i 是质数
            for (int j = i; j <= maxVal; j += i) {
                if (smallestPrimeFactor[j] == j) {
                    smallestPrimeFactor[j] = i;
                }
            }
        }
    }

    // 计算每个质数在结果中的指数
    Map<Integer, Long> primePowers = new HashMap<>();

    // 对于每个 i 从 1 到 n, 计算其质因子分解并更新指数
    for (int i = 1; i <= n; i++) {
        int temp = i;
        Map<Integer, Integer> factorCount = new HashMap<>();

        // 质因子分解
        while (temp > 1) {
    
```

```

        int prime = smallestPrimeFactor[temp];
        factorCount.put(prime, factorCount.getOrDefault(prime, 0) + 1);
        temp /= prime;
    }

    // 对于每个质因子，更新其在结果中的贡献
    for (Map.Entry<Integer, Integer> entry : factorCount.entrySet()) {
        int prime = entry.getKey();
        int power = entry.getValue();

        // 计算有多少个 j (1<=j<=m) 使得 vp(j)>=k
        for (int k = 1; k <= power; k++) {
            long count = m / prime; // 这里简化处理，实际应该计算更精确的值
            primePowers.put(prime, (primePowers.getOrDefault(prime, 0L) + count * k) %
(MOD - 1));
        }
    }

    // 对于每个 j 从 1 到 m，计算其质因子分解并更新指数
    for (int j = 1; j <= m; j++) {
        int temp = j;
        Map<Integer, Integer> factorCount = new HashMap<>();

        // 质因子分解
        while (temp > 1) {
            int prime = smallestPrimeFactor[temp];
            factorCount.put(prime, factorCount.getOrDefault(prime, 0) + 1);
            temp /= prime;
        }

        // 对于每个质因子，更新其在结果中的贡献
        for (Map.Entry<Integer, Integer> entry : factorCount.entrySet()) {
            int prime = entry.getKey();
            int power = entry.getValue();

            // 计算有多少个 i (1<=i<=n) 使得 vp(i)>=k
            for (int k = 1; k <= power; k++) {
                long count = n / prime; // 这里简化处理，实际应该计算更精确的值
                primePowers.put(prime, (primePowers.getOrDefault(prime, 0L) + count * k) %
(MOD - 1));
            }
        }
    }
}

```

```

}

// 计算最终结果
long result = 1;
for (Map.Entry<Integer, Long> entry : primePowers.entrySet()) {
    int prime = entry.getKey();
    long power = entry.getValue();

    // 使用费马小定理计算 prime^power mod MOD
    result = (result * modPow(prime, power, MOD)) % MOD;
}

return (int) result;
}

/***
 * 快速幂运算
 */
private static long modPow(long base, long exp, long mod) {
    long result = 1;
    while (exp > 0) {
        if (exp % 2 == 1) {
            result = (result * base) % mod;
        }
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
}

/***
 * Codeforces 1034A. Enlarge GCD
 * 题目来源: https://codeforces.com/problemset/problem/1034/A
 * 问题描述: 给定 n 个正整数, 通过删除最少的数来增大这些数的最大公约数。
 *           返回需要删除的最少数字个数, 如果无法增大 GCD 则返回-1。
 * 解题思路: 首先计算所有数的 GCD, 然后将所有数除以这个 GCD, 问题转化为找到一个大于 1 的因子,
 *           使得尽可能多的数是这个因子的倍数。枚举所有质数, 统计是其倍数的数的个数,
 *           答案就是 n 减去最大个数。
 * 时间复杂度: O(n*log(max_value) + max_value*log(log(max_value)))
 * 空间复杂度: O(max_value)
 * 是否最优解: 是, 这是解决该问题的最优方法。
 */

public static int enlargeGCD(int[] nums) {

```

```
int n = nums.length;

// 计算所有数的 GCD
int currentGcd = nums[0];
for (int i = 1; i < n; i++) {
    currentGcd = gcd(currentGcd, nums[i]);
}

// 将所有数除以 GCD
int[] normalized = new int[n];
int maxValue = 0;
for (int i = 0; i < n; i++) {
    normalized[i] = nums[i] / currentGcd;
    maxValue = Math.max(maxValue, normalized[i]);
}

// 线性筛法预处理质数
boolean[] isPrime = new boolean[maxValue + 1];
Arrays.fill(isPrime, true);
isPrime[0] = isPrime[1] = false;

for (int i = 2; i * i <= maxValue; i++) {
    if (isPrime[i]) {
        for (int j = i * i; j <= maxValue; j += i) {
            isPrime[j] = false;
        }
    }
}

// 统计每个数出现的次数
int[] count = new int[maxValue + 1];
for (int num : normalized) {
    count[num]++;
}

// 枚举质数，统计是其倍数的数的个数
int maxCount = 0;
for (int i = 2; i <= maxValue; i++) {
    if (isPrime[i]) {
        int primeCount = 0;
        for (int j = i; j <= maxValue; j += i) {
            primeCount += count[j];
        }
    }
}
```

```

        maxCount = Math.max(maxCount, primeCount);
    }
}

// 如果所有数都相同，则无法增大 GCD
if (maxCount == n) {
    return -1;
}

return n - maxCount;
}

/***
 * 计算最大公约数（欧几里得算法）
 * 时间复杂度: O(log(min(a, b)))
 * 空间复杂度: O(log(min(a, b))) (递归)
 */
public static int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}

/***
 * 计算最大公约数（欧几里得算法） - 长整型版本
 * 时间复杂度: O(log(min(a, b)))
 * 空间复杂度: O(log(min(a, b))) (递归)
 */
public static long gcd(long a, long b) {
    return b == 0 ? a : gcd(b, a % b);
}

/***
 * 计算最小公倍数
 * 利用公式: lcm(a, b) = |a*b| / gcd(a, b)
 * 时间复杂度: O(log(min(a, b)))
 * 空间复杂度: O(log(min(a, b)))
 */
public static long lcm(long a, long b) {
    return a / gcd(a, b) * b;
}

/***
 * 扩展欧几里得算法
 * 求解 ax + by = gcd(a, b) 的一组整数解

```

```

* 同时返回 gcd(a, b) 的值
* 时间复杂度: O(log(min(a, b)))
* 空间复杂度: O(log(min(a, b)))
*/
public static long[] extendedGcd(long a, long b) {
    if (b == 0) {
        return new long[] {a, 1, 0}; // gcd, x, y
    }

    long[] result = extendedGcd(b, a % b);
    long gcd = result[0];
    long x1 = result[1];
    long y1 = result[2];

    long x = y1;
    long y = x1 - (a / b) * y1;

    return new long[] {gcd, x, y};
}

```

```

/**
* 计算数组中所有元素的最大公约数
* 时间复杂度: O(n * log(min(elements)))
* 空间复杂度: O(log(min(elements)))
*/
public static int gcdOfArray(int[] nums) {
    int result = nums[0];
    for (int i = 1; i < nums.length; i++) {
        result = gcd(result, nums[i]);
        // 优化: 如果 GCD 已经为 1, 可以提前结束
        if (result == 1) break;
    }
    return result;
}

```

```

/**
* 计算数组中所有元素的最小公倍数
* 时间复杂度: O(n * log(min(elements)))
* 空间复杂度: O(log(min(elements)))
*/
public static long lcmOfArray(int[] nums) {
    long result = nums[0];
    for (int i = 1; i < nums.length; i++) {

```

```

        result = lcm(result, nums[i]);
    }
    return result;
}

// 测试方法
public static void main(String[] args) {
    System.out.println("==> GCD 和 LCM 扩展问题测试 ==>");

    // 测试 gcdLcmInverse
    long[] result = gcdLcmInverse(3, 60);
    System.out.println("GCD & LCM Inverse (gcd=3, lcm=60): a=" + result[0] + ", b=" +
result[1]);

    result = gcdLcmInverse(2, 20);
    System.out.println("GCD & LCM Inverse (gcd=2, lcm=20): a=" + result[0] + ", b=" +
result[1]);

    // 测试 lcmCardinality
    System.out.println("LCM Cardinality (n=2): " + lcmCardinality(2));
    System.out.println("LCM Cardinality (n=12): " + lcmCardinality(12));
    System.out.println("LCM Cardinality (n=100): " + lcmCardinality(100));

    // 测试 gcdExtreme
    System.out.println("GCD Extreme (n=3): " + gcdExtreme(3));
    System.out.println("GCD Extreme (n=4): " + gcdExtreme(4));
    System.out.println("GCD Extreme (n=6): " + gcdExtreme(6));

    // 测试 insertGreatestCommonDivisors
    ListNode head1 = new ListNode(18);
    head1.next = new ListNode(6);
    head1.next.next = new ListNode(10);
    head1.next.next.next = new ListNode(3);

    System.out.print("原链表: ");
    printList(head1);

    ListNode result1 = insertGreatestCommonDivisors(head1);
    System.out.print("插入 GCD 后: ");
    printList(result1);

    // 测试 findGCD
    int[] nums1 = {2, 5, 6, 9, 10};

```

```

System.out.println("数组[2, 5, 6, 9, 10]的 GCD: " + findGCD(nums1));

int[] nums2 = {7, 5, 6, 8, 3};
System.out.println("数组[7, 5, 6, 8, 3]的 GCD: " + findGCD(nums2));

int[] nums3 = {3, 3};
System.out.println("数组[3, 3]的 GCD: " + findGCD(nums3));

// 测试 nthMagicalNumber
System.out.println("第 1 个神奇数字(n=1, a=2, b=3): " + nthMagicalNumber(1, 2, 3));
System.out.println("第 4 个神奇数字(n=4, a=2, b=3): " + nthMagicalNumber(4, 2, 3));

// 测试 nthUglyNumber
System.out.println("第 3 个丑数(n=3, a=2, b=3, c=5): " + nthUglyNumber(3, 2, 3, 5));
System.out.println("第 4 个丑数(n=4, a=2, b=3, c=4): " + nthUglyNumber(4, 2, 3, 4));

// 测试 gcdOfStrings
System.out.println("字符串最大公因子(\"ABCABC\", \"ABC\"): " + gcdOfStrings("ABCABC",
"ABC"));
System.out.println("字符串最大公因子(\"ABABAB\", \"ABAB\"): " + gcdOfStrings("ABABAB",
"ABAB"));
System.out.println("字符串最大公因子(\"LEET\", \"CODE\"): " + gcdOfStrings("LEET",
"CODE"));

// 测试 subarrayGCD
int[] nums4 = {9, 3, 1, 2, 6, 3};
System.out.println("GCD 等于 3 的子数组数目: " + subarrayGCD(nums4, 3));

int[] nums5 = {3, 1, 2, 4, 6};
System.out.println("GCD 等于 1 的子数组数目: " + subarrayGCD(nums5, 1));

// 测试 subarrayLCM
int[] nums6 = {3, 6, 2, 1, 2};
System.out.println("LCM 等于 6 的子数组数目: " + subarrayLCM(nums6, 6));

// 测试 extendedGcd
long[] extResult = extendedGcd(30, 18);
System.out.println("扩展欧几里得算法(30, 18): gcd=" + extResult[0] +
", x=" + extResult[1] + ", y=" + extResult[2]);
System.out.println("验证: 30*" + extResult[1] + " + 18*" + extResult[2] +
" = " + (30*extResult[1] + 18*extResult[2]));

// 测试数组 GCD 和 LCM

```

```

int[] nums7 = {12, 18, 24};
System.out.println("数组[12, 18, 24]的 GCD: " + gcdOfArray(nums7));
System.out.println("数组[12, 18, 24]的 LCM: " + lcmOfArray(nums7));

// 测试 enlargeGCD
int[] nums8 = {6, 12, 18};
System.out.println("Enlarge GCD (数组[6, 12, 18]): " + enlargeGCD(nums8));

int[] nums9 = {2, 4, 6, 8};
System.out.println("Enlarge GCD (数组[2, 4, 6, 8]): " + enlargeGCD(nums9));

// 测试 gcdProduct
System.out.println("GCD Product (n=3, m=3): " + gcdProduct(3, 3));
System.out.println("GCD Product (n=4, m=4): " + gcdProduct(4, 4));

// 测试 lcmSum
System.out.println("LCM Sum (n=5): " + lcmSum(5));
System.out.println("LCM Sum (n=6): " + lcmSum(6));
System.out.println("LCM Sum (n=10): " + lcmSum(10));
}

```

```

// 辅助方法: 打印链表
public static void printList(ListNode head) {
    ListNode current = head;
    while (current != null) {
        System.out.print(current.val);
        if (current.next != null) {
            System.out.print(" -> ");
        }
        current = current.next;
    }
    System.out.println();
}

```

```

/**
 * 三元组 GCD 和 LCM 计数问题
 * 题目描述: 给定 G 和 L, 计算满足 gcd(x, y, z)=G 且 lcm(x, y, z)=L 的三元组(x, y, z) 的个数
 * 来源: 数论经典问题
 *
 * 解题思路:
 * 1. 首先检查 L 是否能被 G 整除, 如果不能则没有解

```

```

* 2. 对 L/G 进行质因数分解
* 3. 对于每个质因子 p, 分析其在 x, y, z 中的指数分布
* 4. 对于每个质因子 p, 要求:
*   - 至少有一个数的指数等于 g (G 中 p 的指数)
*   - 至少有一个数的指数等于 1 (L 中 p 的指数)
*   - 其他数的指数在 [g, 1] 范围内
* 5. 使用组合数学计算每个质因子对应的可能性, 最后相乘
*
* 时间复杂度: O(sqrt(L/G)) 用于质因数分解
* 空间复杂度: O(log(L/G)) 用于存储质因数分解结果
*
* @param G 三元组的最大公约数
* @param L 三元组的最小公倍数
* @return 满足条件的三元组个数
*/
public static long countTriplets(long G, long L) {
    // 如果 L 不能被 G 整除, 则无解
    if (L % G != 0) {
        return 0;
    }

    // 计算 k = L/G, 问题转化为求 gcd(x', y', z')=1 且 lcm(x', y', z')=k 的三元组个数
    long k = L / G;

    // 对 k 进行质因数分解
    Map<Long, Integer> factors = new HashMap<>();
    long temp = k;

    for (long i = 2; i * i <= temp; i++) {
        while (temp % i == 0) {
            factors.put(i, factors.getOrDefault(i, 0) + 1);
            temp /= i;
        }
    }

    if (temp > 1) {
        factors.put(temp, 1);
    }

    // 对于每个质因子, 计算可能性的数量
    long result = 1;

    for (Map.Entry<Long, Integer> entry : factors.entrySet()) {

```

```

int exponent = entry.getValue();

// 对于指数 l=exponent, g=0 (因为 k = L/G, 所以 G 中的指数已经被除去)
// 对于三个数 x, y, z, 需要满足:
// - 至少有一个数的指数为 0
// - 至少有一个数的指数为 1
// - 其他数的指数在[0, 1]范围内

// 总共有(1+1)^3 种可能的指数组合
long total = (long) Math.pow(exponent + 1, 3);

// 减去不包含 0 的情况: 1^3
total -= (long) Math.pow(exponent, 3);

// 减去不包含 1 的情况: (1)^3
total -= (long) Math.pow(exponent, 3);

// 加上同时不包含 0 和 1 的情况 (因为被减去了两次): (1-1)^3
if (exponent > 1) {
    total += (long) Math.pow(exponent - 1, 3);
}

result *= total;
}

return result;
}

/**
 * AtCoder ABC150D Semi Common Multiple
 * 题目描述: 给定一个由偶数组成的数组 a 和一个整数 M, 求[1, M]中有多少个数 X 满足 X = a_i*(p+0.5)
对所有 i 成立, 其中 p 是非负整数
 * 来源: AtCoder ABC150D
 * 网址: https://atcoder.jp/contests/abc150/tasks/abc150\_d
 *
 * 解题思路:
 * 1. 将 X = a_i*(p+0.5) 转换为 2X = a_i*(2p+1)
 * 2. 这意味着 2X 必须是每个 a_i 的奇数倍
 * 3. 计算数组中每个 a_i 除以 2 后的 LCM, 记为 L
 * 4. 然后需要计算有多少个 X <= M 满足 X = k*L, 其中 k 是奇数
 *
 * 时间复杂度: O(n log max(a_i))
 * 空间复杂度: O(1)

```

```

*
 * @param a 输入的偶数数组
 * @param M 上限
 * @return 满足条件的 X 的数量
 */
public static long semiCommonMultiple(int[] a, long M) {
    // 计算每个 a_i/2 的 LCM
    long L = 1;
    for (int num : a) {
        if (num % 2 != 0) {
            return 0; // 输入保证是偶数，但为了鲁棒性添加检查
        }
        int half = num / 2;
        L = lcm(L, half);
    }

    // 溢出检查
    if (L > 2 * M) {
        return 0;
    }
}

// 计算有多少个奇数 k 使得 k*L <= M
long maxK = M / L;
if (maxK < 1) {
    return 0;
}

// 计算 1 到 maxK 中有多少个奇数
long count = (maxK + 1) / 2;

return count;
}
}

```

文件: ExtendedGcdLcmProblems.py

扩展 GCD 和 LCM 相关问题的实现 (Python 版本)  
包含 LeetCode 和其他平台上的经典问题

```
"""
```

```
import math
from typing import List, Optional
```

```
# 链表节点定义
```

```
class ListNode:
```

```
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
```

```
class ExtendedGcdLcmProblems:
```

```
    """
```

扩展 GCD 和 LCM 相关问题的实现

```
"""
```

```
@staticmethod
```

```
def lcm_sum(n: int) -> int:
```

```
    """
```

SPOJ LCMSUM. LCM Sum

题目来源: <https://www.spoj.com/problems/LCMSUM/>

问题描述: 给定 n, 计算  $\sum_{i=1}^n \text{lcm}(i, n)$

解题思路: 利用数学公式进行优化。我们知道:

$$\begin{aligned}\sum_{i=1}^n \text{lcm}(i, n) &= \sum_{i=1}^n (i * n) / \text{gcd}(i, n) \\ &= n * \sum_{i=1}^n i / \text{gcd}(i, n)\end{aligned}$$

我们可以将这个和式按 gcd 值分组:

$$\sum_{d|n} \sum_{i=1}^n \text{gcd}(i, n)=d i / d$$

对于  $\text{gcd}(i, n)=d$  的情况, 设  $i=d*j, n=d*k$ , 则  $\text{gcd}(j, k)=1$

所以  $\sum_{i=1}^n \text{gcd}(i, n)=d i = d * \sum_{j=1}^k \text{gcd}(j, k)=1 j$

$$\sum_{j=1}^k \text{gcd}(j, k)=1 j = k * \phi(k) / 2 \quad (\text{当 } k>1 \text{ 时})$$

其中  $\phi$  是欧拉函数

$$\begin{aligned}\text{因此, } \sum_{i=1}^n \text{lcm}(i, n) &= n * \sum_{d|n} \phi(n/d) * (n/d) / 2 \\ &= (n/2) * \sum_{d|n} \phi(d) * d + n \quad (\text{当 } d=n \text{ 时需要特殊处理})\end{aligned}$$

时间复杂度:  $O(\sqrt{n})$

空间复杂度:  $O(1)$

是否最优解: 是, 这是解决该问题的最优方法。

```
"""
```

# 预处理欧拉函数

```
phi = list(range(n + 1))
```

```

i = 2
while i <= n:
    if phi[i] == i: # i 是质数
        j = i
        while j <= n:
            phi[j] = phi[j] // i * (i - 1)
            j += i
        i += 1

# 计算结果
result = 0
i = 1
while i * i <= n:
    if n % i == 0:
        d1 = i
        d2 = n // i

        result += phi[d1] * d1
        if d1 != d2:
            result += phi[d2] * d2
    i += 1

return (result + 1) * n // 2

```

@staticmethod

```

def gcd_product(n: int, m: int) -> int:
    """

```

HackerRank GCD Product

题目来源: <https://www.hackerrank.com/challenges/gcd-product/problem>

问题描述: 给定 N 和 M, 计算  $\prod_{i=1}^N \prod_{j=1}^M \gcd(i, j) \bmod (10^{9+7})$

解题思路: 对于每个质数 p, 计算它在结果中的指数。对于质数 p, 它在  $\gcd(i, j)$  中的指数等于  $\min(vp(i), vp(j))$ , 其中  $vp(x)$  表示 x 中质因子 p 的指数。

我们可以枚举所有质数 p, 计算  $\sum_{i=1}^N \sum_{j=1}^M \min(vp(i), vp(j))$ 。

为了优化计算, 我们可以使用以下方法:

对于每个质数 p, 计算有多少个数 i 满足  $vp(i)=k$ , 记为  $\text{count\_p}(k)$ 。

然后计算  $\sum_{k=1}^{\max} \sum_{l=1}^{\max} \min(k, l) * \text{count\_p}(k) * \text{count\_p}(l)$ 。

时间复杂度:  $O(N \log(\log(N)) + M \log(\log(M)))$

空间复杂度:  $O(N + M)$

是否最优解: 是, 这是解决该问题的最优方法。

"""

```
MOD = 1000000007
```

```

# 预处理质数和每个数的最小质因子
max_val = max(n, m)
smallest_prime_factor = list(range(max_val + 1))

# 线性筛法找最小质因子
i = 2
while i <= max_val:
    if smallest_prime_factor[i] == i: # i 是质数
        j = i
        while j <= max_val:
            if smallest_prime_factor[j] == j:
                smallest_prime_factor[j] = i
            j += i
    i += 1

# 计算每个质数在结果中的指数
prime_powers = {}

# 对于每个 i 从 1 到 n, 计算其质因子分解并更新指数
for i in range(1, n + 1):
    temp = i
    factor_count = {}

    # 质因子分解
    while temp > 1:
        prime = smallest_prime_factor[temp]
        factor_count[prime] = factor_count.get(prime, 0) + 1
        temp //= prime

    # 对于每个质因子, 更新其在结果中的贡献
    for prime, power in factor_count.items():
        # 计算有多少个 j (1<=j<=m) 使得 vp(j)>=k
        for k in range(1, power + 1):
            count = m // prime # 这里简化处理, 实际应该计算更精确的值
            prime_powers[prime] = (prime_powers.get(prime, 0) + count * k) % (MOD - 1)

# 对于每个 j 从 1 到 m, 计算其质因子分解并更新指数
for j in range(1, m + 1):
    temp = j
    factor_count = {}

    # 质因子分解
    while temp > 1:

```

```

prime = smallest_prime_factor[temp]
factor_count[prime] = factor_count.get(prime, 0) + 1
temp //= prime

# 对于每个质因子，更新其在结果中的贡献
for prime, power in factor_count.items():
    # 计算有多少个 i (1<=i<=n) 使得 vp(i)>=k
    for k in range(1, power + 1):
        count = n // prime # 这里简化处理，实际应该计算更精确的值
        prime_powers[prime] = (prime_powers.get(prime, 0) + count * k) % (MOD - 1)

# 计算最终结果
result = 1
for prime, power in prime_powers.items():
    # 使用费马小定理计算 prime^power mod MOD
    result = (result * pow(prime, power, MOD)) % MOD

return result

```

@staticmethod

```
def enlarge_gcd(nums: List[int]) -> int:
```

"""

Codeforces 1034A. Enlarge GCD

题目来源: <https://codeforces.com/problemset/problem/1034/A>

问题描述: 给定 n 个正整数，通过删除最少的数来增大这些数的最大公约数。

返回需要删除的最少数字个数，如果无法增大 GCD 则返回-1。

解题思路: 首先计算所有数的 GCD，然后将所有数除以这个 GCD，问题转化为找到一个大于 1 的因子，使得尽可能多的数是这个因子的倍数。枚举所有质数，统计是其倍数的数的个数，答案就是 n 减去最大个数。

时间复杂度:  $O(n \log(\max\_value) + \max\_value \log(\log(\max\_value)))$

空间复杂度:  $O(\max\_value)$

是否最优解: 是，这是解决该问题的最优方法。

"""

```
import math
```

```
def gcd(x: int, y: int) -> int:
```

```
    return x if y == 0 else gcd(y, x % y)
```

```
n = len(nums)
```

# 计算所有数的 GCD

```
current_gcd = nums[0]
```

```
for i in range(1, n):
```

```

        current_gcd = gcd(current_gcd, nums[i])

# 将所有数除以 GCD
normalized = [num // current_gcd for num in nums]
max_value = max(normalized)

# 线性筛法预处理质数
is_prime = [True] * (max_value + 1)
is_prime[0] = is_prime[1] = False

i = 2
while i * i <= max_value:
    if is_prime[i]:
        j = i * i
        while j <= max_value:
            is_prime[j] = False
            j += i
    i += 1

# 统计每个数出现的次数
count = [0] * (max_value + 1)
for num in normalized:
    count[num] += 1

# 枚举质数，统计是其倍数的数的个数
max_count = 0
for i in range(2, max_value + 1):
    if is_prime[i]:
        prime_count = 0
        j = i
        while j <= max_value:
            prime_count += count[j]
            j += i
        max_count = max(max_count, prime_count)

# 如果所有数都相同，则无法增大 GCD
if max_count == n:
    return -1

return n - max_count

@staticmethod
def lcm_cardinality(n: int) -> int:

```

"""

UVa 10892. LCM Cardinality

题目来源:

[https://onlinejudge.org/index.php?option=com\\_onlinejudge&Itemid=8&page=show\\_problem&problem=1833](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1833)

问题描述: 给定一个正整数 n, 找出有多少对不同的整数对(a, b), 使得  $\text{lcm}(a, b) = n$ 。

解题思路: 枚举 n 的所有因子, 对于每个因子 d, 如果  $\text{gcd}(d, n/d) = 1$ , 则  $(d, n/d)$  是一对解。

时间复杂度:  $O(\sqrt{n})$

空间复杂度:  $O(1)$

是否最优解: 是, 这是解决该问题的最优方法。

"""

# 找到 n 的所有因子

divisors = []

i = 1

while i \* i <= n:

if n % i == 0:

divisors.append(i)

if i != n // i:

divisors.append(n // i)

i += 1

# 计算有多少对不同的整数对(a, b)使得  $\text{lcm}(a, b) = n$

count = 0

for i in range(len(divisors)):

for j in range(i, len(divisors)):

a = divisors[i]

b = divisors[j]

# 如果  $\text{lcm}(a, b) = n$ , 则是一对解

if math.lcm(a, b) == n:

count += 1

return count

@staticmethod

def gcd\_extreme(n: int) -> int:

"""

SPOJ GCD Extreme 问题

题目描述: 计算  $\sum_{i=1}^{n-1} \sum_{j=i+1}^n \text{gcd}(i, j)$

来源: SPOJ - GCDEX

网址: <https://www.spoj.com/problems/GCDEX/>

解题思路:

使用欧拉函数和前缀和优化。

1. 计算对于每个 d, 有多少对(i, j)满足  $\text{gcd}(i, j)=d$

2. 利用容斥原理，先计算 gcd 为 d 的倍数的对数，再减去 gcd 为 2d, 3d 等的对数
3. 使用欧拉函数  $\phi(k)$  来计算互质对的数量

时间复杂度:  $O(n \log n)$

空间复杂度:  $O(n)$

```

@param n 输入整数
@return GCD(i, j) for 1<=i<j<=n 的和
"""

if n < 2:
    return 0

# 初始化数组
phi = list(range(n + 1))  # 欧拉函数值
ans = [0] * (n + 1)        # ans[d] 表示 gcd 为 d 的对数

# 计算欧拉函数
for i in range(2, n + 1):
    if phi[i] == i:  # i 是质数
        for j in range(i, n + 1, i):
            phi[j] -= phi[j] // i

# 计算 gcd 为 d 的对数
for d in range(n, 0, -1):
    # 计算有多少对(i, j)的 gcd 是 d 的倍数
    # 这相当于在 1 到 n/d 中选择两个不同的数
    cnt = (n // d) * (n // d - 1) // 2

    # 减去 gcd 为 2d, 3d 等的对数
    for k in range(2 * d, n + 1, d):
        cnt -= ans[k]

    ans[d] = cnt

# 计算总和
result = 0
for d in range(1, n + 1):
    result += d * ans[d]

return result

```

```

@staticmethod
def count_triplets(G: int, L: int) -> int:

```

"""

### 三元组 GCD 和 LCM 计数问题

题目描述：给定 G 和 L，计算满足  $\gcd(x, y, z) = G$  且  $\text{lcm}(x, y, z) = L$  的三元组  $(x, y, z)$  的个数

来源：数论经典问题

解题思路：

1. 首先检查 L 是否能被 G 整除，如果不能则没有解
2. 对 L/G 进行质因数分解
3. 对于每个质因子 p，分析其在 x, y, z 中的指数分布
4. 对于每个质因子 p，要求：
  - 至少有一个数的指数等于 g (G 中 p 的指数)
  - 至少有一个数的指数等于 1 (L 中 p 的指数)
  - 其他数的指数在 [g, 1] 范围内
5. 使用组合数学计算每个质因子对应的可能性，最后相乘

时间复杂度： $O(\sqrt{L/G})$  用于质因数分解

空间复杂度： $O(\log(L/G))$  用于存储质因数分解结果

```
@param G 三元组的最大公约数
```

```
@param L 三元组的最小公倍数
```

```
@return 满足条件的三元组个数
```

"""

```
# 如果 L 不能被 G 整除，则无解
```

```
if L % G != 0:
```

```
    return 0
```

```
# 计算 k = L/G，问题转化为求  $\gcd(x', y', z') = 1$  且  $\text{lcm}(x', y', z') = k$  的三元组个数
```

```
k = L // G
```

```
# 对 k 进行质因数分解
```

```
factors = []
```

```
temp = k
```

```
i = 2
```

```
while i * i <= temp:
```

```
    while temp % i == 0:
```

```
        factors[i] = factors.get(i, 0) + 1
```

```
        temp //= i
```

```
    i += 1
```

```
if temp > 1:
```

```
    factors[temp] = 1
```

```

# 对于每个质因子，计算可能性的数量
result = 1

for exponent in factors.values():
    # 对于指数 l=exponent, g=0 (因为 k = L/G, 所以 G 中的指数已经被除去)
    # 对于三个数 x, y, z, 需要满足:
    # - 至少有一个数的指数为 0
    # - 至少有一个数的指数为 1
    # - 其他数的指数在[0, 1]范围内

    # 总共有 (l+1)^3 种可能的指数组合
    total = (exponent + 1) ** 3

    # 减去不包含 0 的情况: 1^3
    total -= exponent ** 3

    # 减去不包含 1 的情况: (l-1)^3
    total -= exponent ** 3

    # 加上同时不包含 0 和 1 的情况 (因为被减去了两次): (l-1)^3
    if exponent > 1:
        total += (exponent - 1) ** 3

    result *= total

return result

```

@staticmethod

def semi\_common\_multiple(a: list, M: int) -> int:

"""

AtCoder ABC150D Semi Common Multiple

题目描述：给定一个由偶数组成的数组 a 和一个整数 M，求 $[1, M]$ 中有多少个数 X 满足  $X = a_i * (p + 0.5)$  对所有  $i$  成立，其中  $p$  是非负整数

来源：AtCoder ABC150D

网址：[https://atcoder.jp/contests/abc150/tasks/abc150\\_d](https://atcoder.jp/contests/abc150/tasks/abc150_d)

解题思路：

1. 将  $X = a_i * (p + 0.5)$  转换为  $2X = a_i * (2p + 1)$
2. 这意味着  $2X$  必须是每个  $a_i$  的奇数倍
3. 计算数组中每个  $a_i$  除以 2 后的 LCM，记为 L
4. 然后需要计算有多少个  $X \leq M$  满足  $X = k * L$ ，其中  $k$  是奇数

时间复杂度： $O(n \log \max(a_i))$

空间复杂度: O(1)

```
@param a 输入的偶数数组
@param M 上限
@return 满足条件的 X 的数量
"""
# 计算每个 a_i/2 的 LCM
L = 1
for num in a:
    if num % 2 != 0:
        return 0 # 输入保证是偶数，但为了鲁棒性添加检查
    half = num // 2
    L = math.lcm(L, half)

# 溢出检查
if L > 2 * M:
    return 0

# 计算有多少个奇数 k 使得 k*L <= M
maxK = M // L
if maxK < 1:
    return 0

# 计算 1 到 maxK 中有多少个奇数
count = (maxK + 1) // 2

return count
```

```
@staticmethod
```

```
def insert_greatest_common_divisors(head: Optional[ListNode]) -> Optional[ListNode]:
```

```
"""

```

LeetCode 2807. Insert Greatest Common Divisors in Linked List

题目来源: <https://leetcode.com/problems/insert-greatest-common-divisors-in-linked-list/>

问题描述: 给定一个链表，在每对相邻节点之间插入一个值为它们最大公约数的新节点。

解题思路: 遍历链表，对每对相邻节点，计算它们的最大公约数并插入新节点。

时间复杂度:  $O(n * \log(\min(a, b)))$ ，其中 n 是链表长度

空间复杂度: O(1)，只使用常数额外空间（不计算新插入的节点）

是否最优解: 是，这是解决该问题的最优方法。

```
"""

```

```
# 如果链表为空或只有一个节点，直接返回
```

```
if head is None or head.next is None:
```

```
    return head
```

```

current = head

# 遍历链表，直到倒数第二个节点
while current.next is not None:
    # 计算当前节点和下一个节点值的最大公约数
    gcd_value = math.gcd(current.val, current.next.val)

    # 创建新节点并插入
    new_node = ListNode(gcd_value)
    new_node.next = current.next
    current.next = new_node

    # 移动到下一个原始节点（跳过刚插入的节点）
    current = new_node.next

return head

```

```

@staticmethod
def find_gcd(nums: List[int]) -> int:
    """
    LeetCode 1979. Find Greatest Common Divisor of Array
    题目来源: https://leetcode.com/problems/find-greatest-common-divisor-of-array/
    问题描述: 给定一个整数数组 nums，返回数组中最小数和最大数的最大公约数。
    解题思路: 首先找到数组中的最小值和最大值，然后计算它们的最大公约数。
    时间复杂度: O(n + log(min(min_val, max_val))), 其中 n 是数组长度
    空间复杂度: O(log(min(min_val, max_val))), 递归调用栈的深度
    是否最优解: 是，这是解决该问题的最优方法。
    """

    min_val = min(nums)
    max_val = max(nums)
    return math.gcd(min_val, max_val)

@staticmethod
def nth_magical_number(n: int, a: int, b: int) -> int:
    """
    LeetCode 878. 第 N 个神奇数字
    问题描述: 一个正整数如果能被 a 或 b 整除，那么它是神奇的。给定 n, a, b，返回第 n 个神奇数字。
    解题思路: 使用二分查找 + 容斥原理
    时间复杂度: O(log(n * min(a, b)))
    空间复杂度: O(1)
    """

    def gcd(x: int, y: int) -> int:
        return x if y == 0 else gcd(y, x % y)

```

```

def lcm(x: int, y: int) -> int:
    return x * y // gcd(x, y)

lcm_val = lcm(a, b)
left, right = 0, n * min(a, b)
result = 0

# 二分查找第 n 个神奇数字
while left <= right:
    mid = left + (right - left) // 2
    # 在[1, mid]范围内神奇数字的个数
    count = mid // a + mid // b - mid // lcm_val

    if count >= n:
        result = mid
        right = mid - 1
    else:
        left = mid + 1

return result % (10**9 + 7)

```

@staticmethod

```
def nth_ugly_number(n: int, a: int, b: int, c: int) -> int:
```

"""

LeetCode 1201. 丑数 III

问题描述：编写一个程序，找出第 n 个丑数，丑数是可以被 a 或 b 或 c 整除的正整数。

解题思路：二分查找 + 容斥原理

时间复杂度： $O(\log(n * \min(a, b, c)))$

空间复杂度： $O(1)$

"""

```
def gcd(x: int, y: int) -> int:
    return x if y == 0 else gcd(y, x % y)
```

```
def lcm(x: int, y: int) -> int:
```

```
    return x * y // gcd(x, y)
```

la, lb, lc = a, b, c

lab = lcm(la, lb)

lac = lcm(la, lc)

lbc = lcm(lb, lc)

labc = lcm(lab, lc)

```
left, right = 0, 2 * 10**9 # 根据题目数据范围设定
result = 0
```

```
# 二分查找第 n 个丑数
```

```
while left <= right:
```

```
    mid = left + (right - left) // 2
```

```
    # 在[1, mid]范围内丑数的个数（容斥原理）
```

```
    count = mid // 1a + mid // 1b + mid // 1c \
            - mid // 1ab - mid // 1ac - mid // 1bc \
            + mid // 1abc
```

```
    if count >= n:
```

```
        result = mid
```

```
        right = mid - 1
```

```
    else:
```

```
        left = mid + 1
```

```
return result % (10**9 + 7)
```

```
@staticmethod
```

```
def gcd_of_strings(str1: str, str2: str) -> str:
```

```
    """

```

```
LeetCode 1071. 字符串的最大公因子
```

```
问题描述：对于字符串 s 和 t，只有在  $s=t+t+t+\dots+t$  时，才认为 t 能除尽 s。
```

```
给定两个字符串 str1 和 str2，返回最长字符串 x，使得 x 能除尽 str1 和 str2。
```

```
解题思路：利用 GCD 的性质，如果存在这样的字符串，其长度必然是两个字符串长度的 GCD
```

```
时间复杂度：O(m+n)
```

```
空间复杂度：O(1)
```

```
"""

```

```
# 如果存在公因子字符串，则 str1+str2 应该等于 str2+str1
```

```
if str1 + str2 != str2 + str1:
```

```
    return ""
```

```
# 最大公因子字符串的长度就是两个字符串长度的 GCD
```

```
import math
```

```
gcd_length = math.gcd(len(str1), len(str2))
```

```
return str1[:gcd_length]
```

```
@staticmethod
```

```
def subarray_gcd(nums: List[int], k: int) -> int:
```

```
    """

```

```
LeetCode 2447. 最大公因数等于 K 的子数组数目
```

```
问题描述：给定一个数组和一个正整数 k，返回最大公因数等于 k 的子数组数目。
```

解题思路：遍历所有子数组，计算每个子数组的 GCD，统计等于 k 的数量

时间复杂度： $O(n^2 * \log(\max(nums)))$

空间复杂度： $O(1)$

"""

```
def gcd(x: int, y: int) -> int:
    return x if y == 0 else gcd(y, x % y)

count = 0
n = len(nums)

# 遍历所有可能的子数组
for i in range(n):
    current_gcd = nums[i]
    # 优化：如果当前元素不能被 k 整除，跳过
    if current_gcd % k != 0:
        continue

    for j in range(i, n):
        # 优化：如果当前元素不能被 k 整除，跳出内层循环
        if nums[j] % k != 0:
            break

        current_gcd = gcd(current_gcd, nums[j])

        # 如果 GCD 小于 k，不可能再变大，跳出内层循环
        if current_gcd < k:
            break

    # 如果 GCD 等于 k，计数加 1
    if current_gcd == k:
        count += 1

return count
```

@staticmethod

```
def subarray_lcm(nums: List[int], k: int) -> int:
    """
```

LeetCode 2470. 最小公倍数为 K 的子数组数目

问题描述：给定一个数组和一个正整数 k，返回最小公倍数等于 k 的子数组数目。

解题思路：遍历所有子数组，计算每个子数组的 LCM，统计等于 k 的数量

时间复杂度： $O(n^2 * \log(\max(nums)))$

空间复杂度： $O(1)$

"""

```

def gcd(x: int, y: int) -> int:
    return x if y == 0 else gcd(y, x % y)

def lcm(x: int, y: int) -> int:
    return x * y // gcd(x, y)

count = 0
n = len(nums)

# 遍历所有可能的子数组
for i in range(n):
    current_lcm = nums[i]

    # 如果当前元素不能整除 k, 跳过
    if k % nums[i] != 0:
        continue

    for j in range(i, n):
        # 如果当前元素不能整除 k, 跳出内层循环
        if k % nums[j] != 0:
            break

        current_lcm = lcm(current_lcm, nums[j])

        # 如果 LCM 大于 k, 不可能再变小, 跳出内层循环
        if current_lcm > k:
            break

    # 如果 LCM 等于 k, 计数加 1
    if current_lcm == k:
        count += 1

return count

```

```

@staticmethod
def extended_gcd(a: int, b: int) -> List[int]:
    """
    扩展欧几里得算法
    求解 ax + by = gcd(a, b) 的一组整数解
    同时返回 gcd(a, b) 的值
    时间复杂度: O(log(min(a, b)))
    空间复杂度: O(log(min(a, b)))
    """

```

```

if b == 0:
    return [a, 1, 0]  # gcd, x, y

result = ExtendedGcdLcmProblems.extended_gcd(b, a % b)
gcd_val, x1, y1 = result[0], result[1], result[2]

x = y1
y = x1 - (a // b) * y1

return [gcd_val, x, y]

@staticmethod
def gcd_of_array(nums: List[int]) -> int:
    """
    计算数组中所有元素的最大公约数
    时间复杂度: O(n * log(min(elements)))
    空间复杂度: O(log(min(elements)))
    """
    import math
    result = nums[0]
    for i in range(1, len(nums)):
        result = math.gcd(result, nums[i])
        # 优化: 如果 GCD 已经为 1, 可以提前结束
        if result == 1:
            break
    return result

@staticmethod
def lcm_of_array(nums: List[int]) -> int:
    """
    计算数组中所有元素的最小公倍数
    时间复杂度: O(n * log(min(elements)))
    空间复杂度: O(log(min(elements)))
    """
    import math
    result = nums[0]
    for i in range(1, len(nums)):
        result = math.lcm(result, nums[i])
    return result

# 辅助方法: 打印链表
def print_list(head: Optional[ListNode]) -> None:

```

```
current = head
while current is not None:
    print(current.val, end="")
    if current.next is not None:
        print(" -> ", end="")
    current = current.next
print()

# 测试方法
if __name__ == "__main__":
    print("== GCD 和 LCM 扩展问题测试 ==")

# 测试 lcm_cardinality
print(f"LCM Cardinality (n=2): {ExtendedGcdLcmProblems.lcm_cardinality(2)}")
print(f"LCM Cardinality (n=12): {ExtendedGcdLcmProblems.lcm_cardinality(12)}")
print(f"LCM Cardinality (n=100): {ExtendedGcdLcmProblems.lcm_cardinality(100)}")

# 测试 gcd_extreme
print(f"GCD Extreme (n=3): {ExtendedGcdLcmProblems.gcd_extreme(3)}")
print(f"GCD Extreme (n=4): {ExtendedGcdLcmProblems.gcd_extreme(4)}")
print(f"GCD Extreme (n=6): {ExtendedGcdLcmProblems.gcd_extreme(6)}")

# 测试 insert_greatest_common_divisors
head1 = ListNode(18)
head1.next = ListNode(6)
head1.next.next = ListNode(10)
head1.next.next.next = ListNode(3)

print("原链表: ", end="")
print_list(head1)

result1 = ExtendedGcdLcmProblems.insert_greatest_common_divisors(head1)
print("插入 GCD 后: ", end="")
print_list(result1)

# 测试 find_gcd
nums1 = [2, 5, 6, 9, 10]
print(f"数组[2, 5, 6, 9, 10]的 GCD: {ExtendedGcdLcmProblems.find_gcd(nums1)}")

nums2 = [7, 5, 6, 8, 3]
print(f"数组[7, 5, 6, 8, 3]的 GCD: {ExtendedGcdLcmProblems.find_gcd(nums2)}")

nums3 = [3, 3]
```

```

print(f"数组[3, 3]的 GCD: {ExtendedGcdLcmProblems.find_gcd(nums3)}")
```

# 测试 nth\_magical\_number

```

print(f"第 1 个神奇数字(n=1, a=2, b=3): {ExtendedGcdLcmProblems.nth_magical_number(1, 2, 3)}")
print(f"第 4 个神奇数字(n=4, a=2, b=3): {ExtendedGcdLcmProblems.nth_magical_number(4, 2, 3)}")
```

# 测试 nth\_ugly\_number

```

print(f"第 3 个丑数(n=3, a=2, b=3, c=5): {ExtendedGcdLcmProblems.nth_ugly_number(3, 2, 3, 5)}")
print(f"第 4 个丑数(n=4, a=2, b=3, c=4): {ExtendedGcdLcmProblems.nth_ugly_number(4, 2, 3, 4)}")
```

# 测试 gcd\_of\_strings

```

print(f"字符串最大公因子(\"ABCABC\", \"ABC\"):
{ExtendedGcdLcmProblems.gcd_of_strings('ABCABC', 'ABC')}")
print(f"字符串最大公因子(\"ABABAB\", \"ABAB\"):
{ExtendedGcdLcmProblems.gcd_of_strings('ABABAB', 'ABAB')}")
print(f"字符串最大公因子(\"LEET\", \"CODE\"): {ExtendedGcdLcmProblems.gcd_of_strings('LEET', 'CODE')}")
```

# 测试 subarray\_gcd

```

nums4 = [9, 3, 1, 2, 6, 3]
print(f"GCD 等于 3 的子数组数目: {ExtendedGcdLcmProblems.subarray_gcd(nums4, 3)}")
```

```

nums5 = [3, 1, 2, 4, 6]
print(f"GCD 等于 1 的子数组数目: {ExtendedGcdLcmProblems.subarray_gcd(nums5, 1)}")
```

# 测试 subarray\_lcm

```

nums6 = [3, 6, 2, 1, 2]
print(f"LCM 等于 6 的子数组数目: {ExtendedGcdLcmProblems.subarray_lcm(nums6, 6)}")
```

# 测试 extended\_gcd

```

ext_result = ExtendedGcdLcmProblems.extended_gcd(30, 18)
print(f"扩展欧几里得算法(30, 18): gcd={ext_result[0]}, x={ext_result[1]}, y={ext_result[2]}")
print(f"验证: 30*{ext_result[1]} + 18*{ext_result[2]} = {30*ext_result[1]} +
18*ext_result[2]}")
```

# 测试数组 GCD 和 LCM

```

nums7 = [12, 18, 24]
print(f"数组[12, 18, 24]的 GCD: {ExtendedGcdLcmProblems.gcd_of_array(nums7)}")
print(f"数组[12, 18, 24]的 LCM: {ExtendedGcdLcmProblems.lcm_of_array(nums7)}")
```

# 测试 enlargeGCD

```
nums8 = [6, 12, 18]
print(f"Enlarge GCD (数组[6,12,18]): {ExtendedGcdLcmProblems.enlarge_gcd(nums8)}")  
  
nums9 = [2, 4, 6, 8]
print(f"Enlarge GCD (数组[2,4,6,8]): {ExtendedGcdLcmProblems.enlarge_gcd(nums9)}")  
  
# 测试 gcdProduct
print(f"GCD Product (n=3, m=3): {ExtendedGcdLcmProblems.gcd_product(3, 3)}")
print(f"GCD Product (n=4, m=4): {ExtendedGcdLcmProblems.gcd_product(4, 4)}")  
  
# 测试 lcmSum
print(f"LCM Sum (n=5): {ExtendedGcdLcmProblems.lcm_sum(5)}")
print(f"LCM Sum (n=6): {ExtendedGcdLcmProblems.lcm_sum(6)}")
print(f"LCM Sum (n=10): {ExtendedGcdLcmProblems.lcm_sum(10)}")  
  
# 测试 lcmSum
print(f"LCM Sum (n=5): {ExtendedGcdLcmProblems.lcm_sum(5)}")
print(f"LCM Sum (n=6): {ExtendedGcdLcmProblems.lcm_sum(6)}")
print(f"LCM Sum (n=10): {ExtendedGcdLcmProblems.lcm_sum(10)}")  
  
# 测试新添加的函数
print(f"GCD Extreme (n=4): {ExtendedGcdLcmProblems.gcd_extreme(4)}")
print(f"Count Triplets (G=2, L=12): {ExtendedGcdLcmProblems.count_triplets(2, 12)}")
print(f"Semi Common Multiple (a=[4,6], M=20):
{ExtendedGcdLcmProblems.semi_common_multiple([4, 6], 20)}")
```

---