===
文件夹: class067_ExtendedEuclideanAlgorithm
===

[Markdown 文件]
===
文件: AdditionalProblems.md
===
Class140 补充题目和训练

扩展欧几里得算法相关题目

1. 线性同余方程
- **题目来源**: 模板题
- **算法**: 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)))

2. 乘法逆元
- **题目来源**: 模板题
- **算法**: 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)))

3. 青蛙的约会
- **题目来源**: POJ 1061
- **题目链接**: http://poj.org/problem?id=1061
- **算法**: 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)))

4. C Looooops
- **题目来源**: POJ 2115
- **题目链接**: http://poj.org/problem?id=2115
- **算法**: 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)))

5. The Football Stage
- **题目来源**: Codeforces 1244C
- **题目链接**: https://codeforces.com/problemset/problem/1244/C
- **算法**: 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)))

最大公约数相关题目

1. How Many Points?
- **题目来源**: LightOJ 1077
- **题目链接**: https://lightoj.com/problem/how-many-points
- **算法**: 最大公约数
- **时间复杂度**: O(log(min(dx,dy)))
- **空间复杂度**: O(log(min(dx,dy)))

2. Jewelry
- **题目来源**: HDU 5722
- **题目链接**: https://acm.hdu.edu.cn/showproblem.php?pid=5722
- **算法**: 最大公约数
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(log(min(a,b)))

3. Han Solo and Lazer Gun
- **题目来源**: Codeforces 514B
- **题目链接**: https://codeforces.com/problemset/problem/514/B
- **算法**: 最大公约数
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(log(min(a,b)))

Pick 定理相关题目

1. Area
- **题目来源**: POJ 1265
- **题目链接**: http://poj.org/problem?id=1265
- **算法**: Pick 定理
- **时间复杂度**: O(n*log(max(dx,dy)))
- **空间复杂度**: O(1)

2. Trees on My Island
- **题目来源**: UVA 10088
- **题目链接**:
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1029
- **算法**: Pick 定理
- **时间复杂度**: O(n*log(max(dx,dy)))
- **空间复杂度**: O(1)

赛瓦维斯特定理相关题目

1. 小凯的疑惑
- **题目来源**: 洛谷 P3951
- **题目链接**: https://www.luogu.com.cn/problem/P3951
- **算法**: 赛瓦维斯特定理
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(1)

2. A New Change Problem
- **题目来源**: HDU 1792
- **题目链接**: https://acm.hdu.edu.cn/showproblem.php?pid=1792
- **算法**: 赛瓦维斯特定理
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(1)

综合题目

1. 检查「好数组」
- **题目来源**: LeetCode 1250
- **题目链接**: https://leetcode.cn/problems/check-if-it-is-a-good-array/
- **算法**: 裴蜀定理
- **时间复杂度**: O(n*log(max(nums)))
- **空间复杂度**: O(1)

2. Lunlun Number
- **题目来源**: AtCoder ABC161 D
- **题目链接**: https://atcoder.jp/contests/abc161/tasks/abc161_d
- **算法**: BFS + 数学
- **时间复杂度**: O(K)
- **空间复杂度**: O(K)

3. Small Multiple
- **题目来源**: AtCoder ARC084 B
- **题目链接**: https://atcoder.jp/contests/abc077/tasks/arc084_b
- **算法**: 01-BFS
- **时间复杂度**: O(K)
- **空间复杂度**: O(K)

4. Pagodas
- **题目来源**: HDU 5512
- **题目链接**: https://acm.hdu.edu.cn/showproblem.php?pid=5512
- **算法**: 博弈论 + 数学
- **时间复杂度**: O(log(min(a,b)))

- **空间复杂度**: O(1)

训练建议

初级训练
1. 掌握扩展欧几里得算法的基本实现
2. 理解线性丢番图方程的解法
3. 熟悉最大公约数的计算方法

中级训练
1. 学习 Pick 定理的应用
2. 掌握赛瓦维斯特定理的使用场景
3. 练习将实际问题转化为数学模型

高级训练
1. 研究数论在算法竞赛中的高级应用
2. 学习更多数论定理和算法
3. 参与编程竞赛，提升解题能力

常见错误和注意事项

1. 数据溢出
- **问题**: 在计算大数时可能发生溢出
- **解决方案**: 使用 long long 类型，注意中间计算过程

2. 边界条件处理
- **问题**: 忽略了特殊情况的处理
- **解决方案**: 仔细分析边界条件，添加特殊判断

3. 精度问题
- **问题**: 浮点数计算可能导致精度丢失
- **解决方案**: 使用整数运算，避免浮点数计算

4. 算法选择错误
- **问题**: 对于不同问题选择了不合适的算法
- **解决方案**: 深入理解各种算法的适用场景

性能优化技巧

1. 预处理优化
- 对于重复计算的部分，可以预先计算并存储结果

2. 数学优化

- 利用数学性质简化计算过程

3. 空间换时间
- 在内存允许的情况下，使用缓存减少重复计算

4. 算法选择
- 根据数据规模选择合适的算法

测试用例设计

1. 正常情况
- 设计典型的输入数据，验证算法正确性

2. 边界情况
- 测试边界输入，如最小值、最大值等

3. 特殊情况
- 测试特殊情况，如无解、唯一解等

4. 极端情况
- 测试极端输入，如大数据量、特殊数据分布等

学习资源

1. 在线平台
- LeetCode: https://leetcode.cn/
- 洛谷: https://www.luogu.com.cn/
- Codeforces: https://codeforces.com/
- POJ: http://poj.org/
- AtCoder: https://atcoder.jp/

2. 参考书籍
- 《算法导论》
- 《算法竞赛入门经典》
- 《挑战程序设计竞赛》
- 《数论概论》

3. 在线教程
- 各大 OJ 平台的官方题解
- 算法竞赛相关博客和论坛
- YouTube 上的算法讲解视频

===

文件: AllProblemsWithLinks.md
==
Class140 所有题目及链接汇总

核心题目

1. 二元一次不定方程 (Code01_DiophantineEquation)
- **题目来源**: 洛谷 P5656
- **题目链接**: https://www.luogu.com.cn/problem/P5656
- **算法**: 扩展欧几里得算法
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(1)

2. 青蛙的约会 (Code02_FrogsMeeting)
- **题目来源**: 洛谷 P1516
- **题目链接**: https://www.luogu.com.cn/problem/P1516
- **算法**: 扩展欧几里得算法
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(1)

3. 格点连线上有几个格点 (Code03_HowManyPoints)
- **题目来源**: LightOJ 1077
- **题目链接**: https://lightoj.com/problem/how-many-points
- **算法**: 最大公约数
- **时间复杂度**: O(log(min(dx,dy)))
- **空间复杂度**: O(1)

4. 机器人的移动区域 (Code04_Area)
- **题目来源**: POJ 1265
- **题目链接**: http://poj.org/problem?id=1265
- **算法**: Pick 定理
- **时间复杂度**: O(n*log(max(dx,dy)))
- **空间复杂度**: O(1)

5. 无法组成的最大值 (Code05_LargestUnattainable)
- **题目来源**: 洛谷 P3951
- **题目链接**: https://www.luogu.com.cn/problem/P3951
- **算法**: 赛瓦维斯特定理
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(1)

补充题目

6. POJ 1061 青蛙的约会 (Code06_Poj1061_FrogsMeeting)
- **题目来源**: POJ 1061
- **题目链接**: http://poj.org/problem?id=1061
- **算法**: 扩展欧几里得算法
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(1)

7. POJ 2115 C Looooops (Code07_Poj2115_Loops)
- **题目来源**: POJ 2115
- **题目链接**: http://poj.org/problem?id=2115
- **算法**: 扩展欧几里得算法
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(1)

8. HDU 1576 A/B (Code08_Hdu1576_Division)
- **题目来源**: HDU 1576
- **题目链接**: https://acm.hdu.edu.cn/showproblem.php?pid=1576
- **算法**: 扩展欧几里得算法
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(1)

9. LeetCode 365. 水壶问题 (Code09_LeetCode365_WaterJug)
- **题目来源**: LeetCode 365
- **题目链接**: https://leetcode.cn/problems/water-and-jug-problem/
- **算法**: 裴蜀定理 + 最大公约数
- **时间复杂度**: O(log(min(x,y)))
- **空间复杂度**: O(1)

10. LeetCode 878. 第 N 个神奇数字 (Code10_LeetCode878_NthMagicalNumber)
- **题目来源**: LeetCode 878
- **题目链接**: https://leetcode.cn/problems/nth-magical-number/
- **算法**: 二分搜索 + 容斥原理
- **时间复杂度**: O(log(N * min(A,B)))
- **空间复杂度**: O(1)

11. POJ 2142 The Balance (Code11_Poj2142_TheBalance)
- **题目来源**: POJ 2142
- **题目链接**: http://poj.org/problem?id=2142
- **算法**: 扩展欧几里得算法
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(1)

12. Codeforces 7C. Line (Code12_Codeforces7C_Line)
- **题目来源**: Codeforces 7C
- **题目链接**: https://codeforces.com/problemset/problem/7C
- **算法**: 扩展欧几里得算法
- **时间复杂度**: O(log(min(A,B)))
- **空间复杂度**: O(1)

13. UVA 10090 Marbles (Code13_Uva10090_Marbles)
- **题目来源**: UVA 10090
- **题目链接**:
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1031
- **算法**: 扩展欧几里得算法 + 线性规划
- **时间复杂度**: O(log(min(n1,n2)))
- **空间复杂度**: O(1)

扩展欧几里得算法相关题目

1. 青蛙的约会
- **题目来源**: POJ 1061
- **题目链接**: http://poj.org/problem?id=1061
- **算法**: 扩展欧几里得算法
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(1)

2. C Looooops
- **题目来源**: POJ 2115
- **题目链接**: http://poj.org/problem?id=2115
- **算法**: 扩展欧几里得算法
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(1)

3. The Football Stage
- **题目来源**: Codeforces 1244C
- **题目链接**: https://codeforces.com/problemset/problem/1244/C
- **算法**: 扩展欧几里得算法
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(1)

4. 检查「好数组」
- **题目来源**: LeetCode 1250
- **题目链接**: https://leetcode.cn/problems/check-if-it-is-a-good-array/
- **算法**: 裴蜀定理
- **时间复杂度**: O(n*log(max(nums)))

- **空间复杂度**: O(1)

最大公约数相关题目

1. How Many Points?
- **题目来源**: LightOJ 1077
- **题目链接**: https://lightoj.com/problem/how-many-points
- **算法**: 最大公约数
- **时间复杂度**: O(log(min(dx,dy)))
- **空间复杂度**: O(1)

2. Jewelry
- **题目来源**: HDU 5722
- **题目链接**: https://acm.hdu.edu.cn/showproblem.php?pid=5722
- **算法**: 最大公约数
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(1)

3. Han Solo and Lazer Gun
- **题目来源**: Codeforces 514B
- **题目链接**: https://codeforces.com/problemset/problem/514/B
- **算法**: 最大公约数
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(1)

Pick 定理相关题目

1. Area
- **题目来源**: POJ 1265
- **题目链接**: http://poj.org/problem?id=1265
- **算法**: Pick 定理
- **时间复杂度**: O(n*log(max(dx,dy)))
- **空间复杂度**: O(1)

2. Trees on My Island
- **题目来源**: UVA 10088
- **题目链接**:
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1029
- **算法**: Pick 定理
- **时间复杂度**: O(n*log(max(dx,dy)))
- **空间复杂度**: O(1)

赛瓦维斯特定理相关题目

1. 小凯的疑惑
- **题目来源**：洛谷 P3951
- **题目链接**：https://www.luogu.com.cn/problem/P3951
- **算法**：赛瓦维斯特定理
- **时间复杂度**：O(log(min(a,b)))
- **空间复杂度**：O(1)

2. A New Change Problem
- **题目来源**：HDU 1792
- **题目链接**：https://acm.hdu.edu.cn/showproblem.php?pid=1792
- **算法**：赛瓦维斯特定理
- **时间复杂度**：O(log(min(a,b)))
- **空间复杂度**：O(1)

综合题目

1. Lunlun Number
- **题目来源**：AtCoder ABC161 D
- **题目链接**：https://atcoder.jp/contests/abc161/tasks/abc161_d
- **算法**：BFS + 数学
- **时间复杂度**：O(K)
- **空间复杂度**：O(K)

2. Small Multiple
- **题目来源**：AtCoder ARC084 B
- **题目链接**：https://atcoder.jp/contests/abc077/tasks/arc084_b
- **算法**：01-BFS
- **时间复杂度**：O(K)
- **空间复杂度**：O(K)

3. Pagodas
- **题目来源**：HDU 5512
- **题目链接**：https://acm.hdu.edu.cn/showproblem.php?pid=5512
- **算法**：博弈论 + 数学
- **时间复杂度**：O(log(min(a,b)))
- **空间复杂度**：O(1)

各大 OJ 平台题目分布

LeetCode
- 1250. 检查「好数组」：https://leetcode.cn/problems/check-if-it-is-a-good-array/

洛谷 (Luogu)
- P5656 【模板】二元一次不定方程 (exgcd): https://www.luogu.com.cn/problem/P5656
- P1516 青蛙的约会: https://www.luogu.com.cn/problem/P1516
- P3951 [NOIP2017 提高组] 小凯的疑惑: https://www.luogu.com.cn/problem/P3951

POJ
- 1061 青蛙的约会: http://poj.org/problem?id=1061
- 1265 Area: http://poj.org/problem?id=1265
- 2115 C Looooops: http://poj.org/problem?id=2115

LightOJ
- 1077 How Many Points?: https://lightoj.com/problem/how-many-points

Codeforces
- 514B Han Solo and Lazer Gun: https://codeforces.com/problemset/problem/514/B
- 1244C The Football Stage: https://codeforces.com/problemset/problem/1244/C

AtCoder
- ABC161 D Lunlun Number: https://atcoder.jp/contests/abc161/tasks/abc161_d
- ARC084 B Small Multiple: https://atcoder.jp/contests/abc077/tasks/arc084_b

HDU
- 1576 A/B: https://acm.hdu.edu.cn/showproblem.php?pid=1576
- 1792 A New Change Problem: https://acm.hdu.edu.cn/showproblem.php?pid=1792
- 5512 Pagodas: https://acm.hdu.edu.cn/showproblem.php?pid=5512
- 5722 Jewelry: https://acm.hdu.edu.cn/showproblem.php?pid=5722

UVA
- 10088 Trees on My Island:
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1029

==

文件: EngineeringConsiderations.md
==
Class140 工程化考量

1. 异常处理

1.1 输入验证
在实际工程应用中，必须对所有输入进行验证，确保数据的有效性和安全性。

```java
```

```
// Java 示例：输入验证
public static void validateInput(long a, long b, long c) {
    if (a <= 0 || b <= 0 || c <= 0) {
        throw new IllegalArgumentException("参数必须为正整数");
    }
    if (a > 1e9 || b > 1e9 || c > 1e9) {
        throw new IllegalArgumentException("参数超出允许范围");
    }
}
```

```python
# Python 示例：输入验证
def validate_input(a, b, c):
    if not all(isinstance(x, int) and x > 0 for x in [a, b, c]):
        raise ValueError("参数必须为正整数")
    if any(x > 1e9 for x in [a, b, c]):
        raise ValueError("参数超出允许范围")
```

1.2 边界条件处理
对于各种边界情况，需要特殊处理以确保算法的正确性和稳定性。

```java
// Java 示例：边界条件处理
public static long handleBoundary(long a, long b) {
    // 处理 a 或 b 为 1 的情况
    if (a == 1 || b == 1) {
        return 1;
    }
    // 处理 a 等于 b 的情况
    if (a == b) {
        return a;
    }
    return gcd(a, b);
}
```

1.3 错误信息清晰提示
提供清晰的错误信息有助于快速定位和解决问题。

```cpp
// C++示例：错误信息提示
```

```
#include <stdexcept>
#include <string>

void throwError(const std::string& message) {
    throw std::runtime_error("数论算法错误: " + message);
}
```

2. 性能优化

2.1 时间复杂度优化
通过数学方法优化算法复杂度，避免不必要的计算。

```java
// Java 示例：优化的 GCD 实现
public static long gcd(long a, long b) {
    // 使用位运算优化
    if (a == 0) return b;
    if (b == 0) return a;

    // 计算 a 和 b 中因子 2 的个数
    int shift = 0;
    while (((a | b) & 1) == 0) {
        a >>= 1;
        b >>= 1;
        shift++;
    }

    // 移除 a 中剩余的因子 2
    while ((a & 1) == 0) {
        a >>= 1;
    }

    do {
        // 移除 b 中剩余的因子 2
        while ((b & 1) == 0) {
            b >>= 1;
        }

        // 确保 a <= b
        if (a > b) {
            long temp = a;
            a = b;
```

```
            b = temp;
        }

        b = b - a;
    } while (b != 0);

    return a << shift;
}
```

2.2 空间复杂度优化
使用原地操作，减少内存占用。

```python
# Python 示例：空间优化
def exgcd_optimized(a, b):
    """优化的空间复杂度扩展欧几里得算法"""
    if b == 0:
        return a, 1, 0

    # 递归调用，但只保存必要的信息
    d, x1, y1 = exgcd_optimized(b, a % b)
    x = y1
    y = x1 - (a // b) * y1
    return d, x, y
```

2.3 防止溢出
使用适当的数据类型处理大数运算，注意中间计算过程。

```cpp
// C++示例：防止溢出
#include <climits>

bool isMultiplicationSafe(long long a, long long b) {
    // 检查乘法是否会溢出
    if (a == 0 || b == 0) return true;
    if (a > LLONG_MAX / b) return false;
    if (a < LLONG_MIN / b) return false;
    return true;
}

long long safeMultiply(long long a, long long b) {
```

```
    if (!isMultiplicationSafe(a, b)) {
        throw std::overflow_error("乘法运算溢出");
    }
    return a * b;
}
```

3. 可读性

3.1 变量命名
使用有意义的变量名，提高代码可读性。

```java
// Java 示例：良好的变量命名
public class DiophantineSolver {
    private long coefficientA;      // 方程系数 a
    private long coefficientB;      // 方程系数 b
    private long constantC;         // 方程常数 c
    private long gcdResult;         // 最大公约数结果
    private long solutionX;         // 解 x
    private long solutionY;         // 解 y
}
```

3.2 注释完整
为每个方法和关键步骤添加详细注释。

```python
# Python 示例：完整注释
def solve_linear_diophantine(a, b, c):
    """
    求解线性丢番图方程 ax + by = c

    Args:
        a (int): 方程系数 a
        b (int): 方程系数 b
        c (int): 方程常数 c

    Returns:
        tuple: (has_solution, x, y) 其中 has_solution 表示是否有解,
               x 和 y 是方程的一组特解（如果有解）

    Raises:
```

```
        ValueError: 当输入参数不合法时抛出
    """
    # 验证输入参数
    if not all(isinstance(x, int) for x in [a, b, c]):
        raise ValueError("所有参数必须为整数")

    # 使用扩展欧几里得算法求解
    gcd_val, x0, y0 = extended_gcd(a, b)

    # 判断方程是否有解
    if c % gcd_val != 0:
        return False, 0, 0

    # 计算特解
    x = x0 * (c // gcd_val)
    y = y0 * (c // gcd_val)

    return True, x, y
```

3.3 模块化
将复杂逻辑拆分为独立函数，提高代码复用性和可维护性。

```cpp
// C++示例：模块化设计
class NumberTheoryUtils {
public:
    // 计算最大公约数
    static long long gcd(long long a, long long b);

    // 扩展欧几里得算法
    static void extendedGcd(long long a, long long b, long long& gcd, long long& x, long long& y);

    // 求解线性丢番图方程
    static bool solveDiophantine(long long a, long long b, long long c, long long& x, long long& y);

    // 应用 Pick 定理
    static void applyPickTheorem(double area, long long boundaryPoints, long long& interiorPoints);
};
```

4. 跨语言实现

4.1 Java 版本
面向对象实现，详细注释，适合工程应用。

```java
/**
 * 数论算法工具类
 * 提供扩展欧几里得算法、最大公约数计算等核心功能
 */
public class NumberTheoryUtils {

    /**
     * 扩展欧几里得算法
     * 求解方程 ax + by = gcd(a,b)的一组特解
     *
     * @param a 系数 a
     * @param b 系数 b
     * @param result 存储结果的数组，[0]为 gcd，[1]为 x，[2]为 y
     */
    public static void extendedGcd(long a, long b, long[] result) {
        if (b == 0) {
            result[0] = a;
            result[1] = 1;
            result[2] = 0;
        } else {
            extendedGcd(b, a % b, result);
            long temp = result[1];
            result[1] = result[2];
            result[2] = temp - (a / b) * result[2];
        }
    }
}
```

4.2 C++版本
高效实现，适合竞赛，注意内存管理。

```cpp
/**
 * 数论算法工具类
 * 提供扩展欧几里得算法、最大公约数计算等核心功能
```

```
 */
class NumberTheoryUtils {
public:
    /**
     * 扩展欧几里得算法
     * 求解方程 ax + by = gcd(a,b)的一组特解
     *
     * @param a 系数 a
     * @param b 系数 b
     * @param gcd 存储最大公约数
     * @param x 存储解 x
     * @param y 存储解 y
     */
    static void extendedGcd(long long a, long long b, long long& gcd, long long& x, long long& y)
{
        if (b == 0) {
            gcd = a;
            x = 1;
            y = 0;
        } else {
            extendedGcd(b, a % b, gcd, x, y);
            long long temp = x;
            x = y;
            y = temp - (a / b) * y;
        }
    }
};
```

4.3 Python 版本
简洁实现，适合快速验证，注意性能问题。

```python
def extended_gcd(a, b):
    """
    扩展欧几里得算法
    求解方程 ax + by = gcd(a,b)的一组特解

    Args:
        a: 系数 a
        b: 系数 b

    Returns:
```

        tuple: (gcd, x, y) 其中 gcd 是最大公约数，x 和 y 是方程的一组特解
    """
    if b == 0:
        return a, 1, 0
    else:
        gcd, x1, y1 = extended_gcd(b, a % b)
        x = y1
        y = x1 - (a // b) * y1
        return gcd, x, y
```

## 5. 单元测试

### 5.1 Java 单元测试
```java
import org.junit.Test;
import static org.junit.Assert.*;

public class NumberTheoryUtilsTest {

    @Test
    public void testExtendedGcd() {
        long[] result = new long[3];
        NumberTheoryUtils.extendedGcd(30, 18, result);
        assertEquals(6, result[0]);  // gcd(30, 18) = 6
        assertEquals(-1, result[1]); // 30*(-1) + 18*2 = 6
        assertEquals(2, result[2]);
    }

    @Test
    public void testGcd() {
        assertEquals(6, NumberTheoryUtils.gcd(30, 18));
        assertEquals(1, NumberTheoryUtils.gcd(17, 13));
        assertEquals(5, NumberTheoryUtils.gcd(100, 25));
    }
}
```

5.2 Python 单元测试
```python
import unittest

class TestNumberTheoryUtils(unittest.TestCase):

```python
    def test_extended_gcd(self):
        gcd, x, y = extended_gcd(30, 18)
        self.assertEqual(gcd, 6)
        self.assertEqual(30 * x + 18 * y, 6)

    def test_gcd(self):
        self.assertEqual(gcd(30, 18), 6)
        self.assertEqual(gcd(17, 13), 1)
        self.assertEqual(gcd(100, 25), 25)

if __name__ == '__main__':
    unittest.main()
```

6. 性能测试

6.1 基准测试
```java
public class PerformanceTest {

    public static void main(String[] args) {
        // 测试大数据性能
        long startTime = System.nanoTime();
        long result = NumberTheoryUtils.gcd(1000000000L, 999999999L);
        long endTime = System.nanoTime();

        System.out.println("GCD 计算结果：" + result);
        System.out.println("耗时：" + (endTime - startTime) + " 纳秒");
    }
}
```

6.2 内存使用分析
```python
import tracemalloc

def memory_test():
    tracemalloc.start()

    # 执行算法
    result = extended_gcd(1000000000, 999999999)
```

```
    current, peak = tracemalloc.get_traced_memory()
    print(f"当前内存使用: {current / 1024 / 1024:.2f} MB")
    print(f"峰值内存使用: {peak / 1024 / 1024:.2f} MB")

    tracemalloc.stop()

memory_test()
```

7. 文档化

7.1 API 文档
为每个公共方法提供详细的 API 文档。

7.2 使用说明
提供清晰的使用说明，包括输入输出格式、参数说明等。

7.3 常见问题排查
总结常见问题和解决方案，帮助用户快速解决问题。

8. 安全考虑

8.1 输入验证
确保所有输入都经过验证，防止恶意输入。

8.2 溢出防护
使用适当的数据类型和检查机制防止整数溢出。

8.3 资源管理
正确管理内存和其他资源，防止内存泄漏。

通过以上工程化考量，可以确保数论算法在实际应用中的稳定性、性能和可维护性。

==

文件: ProblemLinks.md
==
Class140 题目链接汇总

核心题目

1. 二元一次不定方程 (Code01_DiophantineEquation)
- **题目来源**: 洛谷 P5656

- **题目链接**: https://www.luogu.com.cn/problem/P5656
- **算法**: 扩展欧几里得算法
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(1)

2. 青蛙的约会 (Code02_FrogsMeeting)
- **题目来源**: 洛谷 P1516
- **题目链接**: https://www.luogu.com.cn/problem/P1516
- **算法**: 扩展欧几里得算法
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(1)

3. 格点连线上有几个格点 (Code03_HowManyPoints)
- **题目来源**: LightOJ 1077
- **题目链接**: https://lightoj.com/problem/how-many-points
- **算法**: 最大公约数
- **时间复杂度**: O(log(min(dx,dy)))
- **空间复杂度**: O(1)

4. 机器人的移动区域 (Code04_Area)
- **题目来源**: POJ 1265
- **题目链接**: http://poj.org/problem?id=1265
- **算法**: Pick 定理
- **时间复杂度**: O(n*log(max(dx,dy)))
- **空间复杂度**: O(1)

5. 无法组成的最大值 (Code05_LargestUnattainable)
- **题目来源**: 洛谷 P3951
- **题目链接**: https://www.luogu.com.cn/problem/P3951
- **算法**: 赛瓦维斯特定理
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(1)

扩展题目

扩展欧几里得算法相关

1. 青蛙的约会
- **题目来源**: POJ 1061
- **题目链接**: http://poj.org/problem?id=1061
- **算法**: 扩展欧几里得算法
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(1)

2. C Looooops
- **题目来源**：POJ 2115
- **题目链接**：http://poj.org/problem?id=2115
- **算法**：扩展欧几里得算法
- **时间复杂度**：O(log(min(a,b)))
- **空间复杂度**：O(1)

3. The Football Stage
- **题目来源**：Codeforces 1244C
- **题目链接**：https://codeforces.com/problemset/problem/1244/C
- **算法**：扩展欧几里得算法
- **时间复杂度**：O(log(min(a,b)))
- **空间复杂度**：O(1)

4. 检查「好数组」
- **题目来源**：LeetCode 1250
- **题目链接**：https://leetcode.cn/problems/check-if-it-is-a-good-array/
- **算法**：裴蜀定理
- **时间复杂度**：O(n*log(max(nums)))
- **空间复杂度**：O(1)

最大公约数相关

1. How Many Points?
- **题目来源**：LightOJ 1077
- **题目链接**：https://lightoj.com/problem/how-many-points
- **算法**：最大公约数
- **时间复杂度**：O(log(min(dx,dy)))
- **空间复杂度**：O(1)

2. Jewelry
- **题目来源**：HDU 5722
- **题目链接**：https://acm.hdu.edu.cn/showproblem.php?pid=5722
- **算法**：最大公约数
- **时间复杂度**：O(log(min(a,b)))
- **空间复杂度**：O(1)

3. Han Solo and Lazer Gun
- **题目来源**：Codeforces 514B
- **题目链接**：https://codeforces.com/problemset/problem/514/B
- **算法**：最大公约数
- **时间复杂度**：O(log(min(a,b)))

- **空间复杂度**: O(1)

Pick 定理相关

1. Area
- **题目来源**: POJ 1265
- **题目链接**: http://poj.org/problem?id=1265
- **算法**: Pick 定理
- **时间复杂度**: O(n*log(max(dx,dy)))
- **空间复杂度**: O(1)

2. Trees on My Island
- **题目来源**: UVA 10088
- **题目链接**:
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1029
- **算法**: Pick 定理
- **时间复杂度**: O(n*log(max(dx,dy)))
- **空间复杂度**: O(1)

赛瓦维斯特定理相关

1. 小凯的疑惑
- **题目来源**: 洛谷 P3951
- **题目链接**: https://www.luogu.com.cn/problem/P3951
- **算法**: 赛瓦维斯特定理
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(1)

2. A New Change Problem
- **题目来源**: HDU 1792
- **题目链接**: https://acm.hdu.edu.cn/showproblem.php?pid=1792
- **算法**: 赛瓦维斯特定理
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(1)

综合题目

1. Lunlun Number
- **题目来源**: AtCoder ABC161 D
- **题目链接**: https://atcoder.jp/contests/abc161/tasks/abc161_d
- **算法**: BFS + 数学
- **时间复杂度**: O(K)
- **空间复杂度**: O(K)

2. Small Multiple
- **题目来源**: AtCoder ARC084 B
- **题目链接**: https://atcoder.jp/contests/abc077/tasks/arc084_b
- **算法**: 01-BFS
- **时间复杂度**: O(K)
- **空间复杂度**: O(K)

3. Pagodas
- **题目来源**: HDU 5512
- **题目链接**: https://acm.hdu.edu.cn/showproblem.php?pid=5512
- **算法**: 博弈论 + 数学
- **时间复杂度**: O(log(min(a,b)))
- **空间复杂度**: O(1)

各大 OJ 平台题目分布

LeetCode
- 1250. 检查「好数组」: https://leetcode.cn/problems/check-if-it-is-a-good-array/

洛谷 (Luogu)
- P5656 【模板】二元一次不定方程 (exgcd): https://www.luogu.com.cn/problem/P5656
- P1516 青蛙的约会: https://www.luogu.com.cn/problem/P1516
- P3951 [NOIP2017 提高组] 小凯的疑惑: https://www.luogu.com.cn/problem/P3951

POJ
- 1061 青蛙的约会: http://poj.org/problem?id=1061
- 1265 Area: http://poj.org/problem?id=1265
- 2115 C Looooops: http://poj.org/problem?id=2115

LightOJ
- 1077 How Many Points?: https://lightoj.com/problem/how-many-points

Codeforces
- 514B Han Solo and Lazer Gun: https://codeforces.com/problemset/problem/514/B
- 1244C The Football Stage: https://codeforces.com/problemset/problem/1244/C

AtCoder
- ABC161 D Lunlun Number: https://atcoder.jp/contests/abc161/tasks/abc161_d
- ARC084 B Small Multiple: https://atcoder.jp/contests/abc077/tasks/arc084_b

HDU
- 1792 A New Change Problem: https://acm.hdu.edu.cn/showproblem.php?pid=1792

- 5512 Pagodas: https://acm.hdu.edu.cn/showproblem.php?pid=5512
- 5722 Jewelry: https://acm.hdu.edu.cn/showproblem.php?pid=5722

UVA
- 10088 Trees on My Island: https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1029

算法分类

1. 数论算法
- 扩展欧几里得算法
- 最大公约数计算
- 裴蜀定理应用

2. 几何算法
- Pick 定理
- 格点计算

3. 组合数学
- 赛瓦维斯特定理
- 硬币问题

学习资源

1. 在线平台
- LeetCode: https://leetcode.cn/
- 洛谷: https://www.luogu.com.cn/
- Codeforces: https://codeforces.com/
- POJ: http://poj.org/
- AtCoder: https://atcoder.jp/
- LightOJ: https://lightoj.com/
- HDU: https://acm.hdu.edu.cn/
- UVA: https://onlinejudge.org/

2. 参考书籍
- 《算法导论》
- 《算法竞赛入门经典》
- 《挑战程序设计竞赛》
- 《数论概论》

3. 在线教程
- 各大 OJ 平台的官方题解
- 算法竞赛相关博客和论坛

- YouTube上的算法讲解视频

==

文件: README.md
==
Class140 - 数论算法与线性丢番图方程

概述

Class140主要讲解数论中的核心算法，包括扩展欧几里得算法、最大公约数计算、线性丢番图方程求解、Pick定理应用以及赛瓦维斯特定理等。这些算法在解决各种数学问题和编程竞赛题目中具有重要作用。

核心算法详解

1. 扩展欧几里得算法 (Extended Euclidean Algorithm)

基本概念
扩展欧几里得算法不仅能够计算两个整数a和b的最大公约数，还能找到整数x和y，使得ax + by = gcd(a, b)。

核心思想
基于欧几里得算法的递归性质，通过回溯过程求解线性丢番图方程。

时间复杂度
O(log(min(a,b)))

应用场景
- 求解线性丢番图方程
- 计算模逆元
- 解决同余方程

2. 线性丢番图方程 (Linear Diophantine Equations)

基本概念
形如ax + by = c的方程，其中a、b、c为整数，求整数解x和y。

解的存在性
方程ax + by = c有整数解当且仅当gcd(a,b)能整除c。

通解公式
如果(x0,y0)是ax + by = c的一组特解，那么通解为：
- x = x0 + (b/gcd(a,b)) * t

- y = y0 - (a/gcd(a,b)) * t
其中 t 为任意整数。

3. Pick 定理 (Pick's Theorem)

基本概念
用于计算顶点均为格点的简单多边形的面积。

公式
A = i + b/2 - 1
其中 A 是多边形面积，i 是内部格点数，b 是边界格点数。

应用场景
- 计算格点多边形面积
- 统计格点数量

4. 赛瓦维斯特定理 (Chicken McNugget Theorem)

基本概念
当正整数 a 和 b 互质时，不能表示为 ax+by（x,y≥0）的最大整数是 ab-a-b。

应用场景
- 硬币问题
- 数论问题

题目详解

1. 二元一次不定方程 (Code01_DiophantineEquation)

问题描述
给定 a、b、c，求解方程 ax + by = c。

解题思路
1. 使用扩展欧几里得算法求解 ax + by = gcd(a,b)的一组特解
2. 判断方程是否有解：当 c 能被 gcd(a,b)整除时有解
3. 如果有解，将特解乘以 c/gcd(a,b)得到原方程的一组特解
4. 根据通解公式求出满足条件的解

相关题目
1. **洛谷 P5656 【模板】二元一次不定方程 (exgcd)**
   - 链接：https://www.luogu.com.cn/problem/P5656
   - 这是本题的来源，是一道模板题

2. **LeetCode 1250. 检查「好数组」**
   - 链接：https://leetcode.cn/problems/check-if-it-is-a-good-array/
   - 本题用到了裴蜀定理，如果数组中所有元素的最大公约数为 1，则为好数组

3. **Codeforces 1244C. The Football Stage**
   - 链接：https://codeforces.com/problemset/problem/1244/C
   - 本题需要求解线性丢番图方程 wx + dy = p，其中 w 和 d 是给定的，p 是变量

4. **HDU 5512 Pagodas**
   - 链接：https://acm.hdu.edu.cn/showproblem.php?pid=5512
   - 本题涉及数论知识，与最大公约数有关

5. **POJ 2115 C Looooops**
   - 链接：http://poj.org/problem?id=2115
   - 本题需要求解模线性方程，可以转化为线性丢番图方程

2. 青蛙的约会 (Code02_FrogsMeeting)

问题描述
有两只青蛙 A 和 B 在一个圆环上，给定它们的初始位置和跳跃速度，求它们何时能相遇。

解题思路
1. 建立方程：设 t 秒后相遇，则有 (x1 + m*t) ≡ (x2 + n*t) (mod l)
2. 化简方程：(m-n)*t ≡ (x2-x1) (mod l)
3. 转换为线性丢番图方程：(m-n)*t + l*k = (x2-x1)
4. 使用扩展欧几里得算法求解

相关题目
1. **洛谷 P1516 青蛙的约会**
   - 链接：https://www.luogu.com.cn/problem/P1516
   - 这是本题的来源，是一道经典题

2. **POJ 1061 青蛙的约会**
   - 链接：http://poj.org/problem?id=1061
   - 与本题完全相同，是 POJ 上的经典题目

3. **HDU 5512 Pagodas**
   - 链接：https://acm.hdu.edu.cn/showproblem.php?pid=5512
   - 本题涉及数论知识，与最大公约数有关

3. 格点连线上有几个格点 (Code03_HowManyPoints)

问题描述

给定两个格点 A(x1,y1)和 B(x2,y2)，求线段 AB 上格点的数量（包括端点）。

解题思路
1. 线段上的格点数量等于 dx 和 dy 的最大公约数加 1
2. dx = |x2-x1|, dy = |y2-y1|
3. 结果 = gcd(dx, dy) + 1

相关题目
1. **LightOJ 1077 How Many Points?**
   - 链接：https://lightoj.com/problem/how-many-points
   - 这是本题的来源，是一道经典题

2. **POJ 1265 Area**
   - 链接：http://poj.org/problem?id=1265
   - 本题需要计算多边形边界上的格点数量，用到了相同的知识点

4. 机器人的移动区域 (Code04_Area)

问题描述
机器人在二维网格上移动形成一个简单多边形，求多边形内部格点数、边界格点数和面积。

解题思路
1. 使用鞋带公式计算多边形面积
2. 使用 gcd 计算每条边上的格点数，累加得到边界格点数
3. 使用 Pick 定理计算内部格点数：内部格点数 = 面积 - 边界格点数/2 + 1

相关题目
1. **POJ 1265 Area**
   - 链接：http://poj.org/problem?id=1265
   - 这是本题的来源，是一道经典题

2. **UVA 10088 - Trees on My Island**
   - 链接：
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1029
   - 本题同样是 Pick 定理的应用

5. 无法组成的最大值 (Code05_LargestUnattainable)

问题描述
给定两种面值为 a 和 b 的硬币（a 和 b 互质），每种硬币有无限个，求无法用这两种硬币组成的最大钱数。

解题思路
1. 根据赛瓦维斯特定理（Chicken McNugget Theorem），当 a 和 b 互质时，

无法表示的最大整数是 a*b-a-b

相关题目
1. **洛谷 P3951 [NOIP2017 提高组] 小凯的疑惑**
   - 链接：https://www.luogu.com.cn/problem/P3951
   - 这是本题的来源，是一道经典题

2. **HDU 1792 A New Change Problem**
   - 链接：https://acm.hdu.edu.cn/showproblem.php?pid=1792
   - 本题是小凯的疑惑的变形，求无法表示的最大数和无法表示的数的个数

新增题目详解

6. LeetCode 365. 水壶问题 (Code09_LeetCode365_WaterJug)

问题描述
有两个容量分别为 x 升和 y 升的水壶以及无限多的水。请判断能否通过使用这两个水壶，从而可以得到恰好 z 升的水？

解题思路
1. 根据裴蜀定理，如果 z 是 x 和 y 的最大公约数的倍数，且 z <= x + y，则有解
2. 特殊情况：如果 z == 0，直接返回 true
3. 如果 x + y < z，返回 false
4. 如果 x == 0 或 y == 0，需要特殊处理

相关题目
1. **LeetCode 365. 水壶问题**
   - 链接：https://leetcode.cn/problems/water-and-jug-problem/
   - 这是本题的来源，是一道经典题

2. **POJ 2142 The Balance**
   - 链接：http://poj.org/problem?id=2142
   - 本题需要求解线性丢番图方程并找到最优解

7. LeetCode 878. 第 N 个神奇数字 (Code10_LeetCode878_NthMagicalNumber)

问题描述
如果正整数可以被 A 或 B 整除，那么它是神奇的。返回第 N 个神奇数字。由于答案可能非常大，返回它模 10^9 + 7 的结果。

解题思路
1. 使用二分搜索法在可能的范围内查找第 N 个神奇数字
2. 对于给定的数字 x，计算小于等于 x 的神奇数字个数

3. 神奇数字个数 = x/A + x/B - x/lcm(A,B)
4. 使用容斥原理避免重复计数

相关题目
1. **LeetCode 878. 第 N 个神奇数字**
   - 链接：https://leetcode.cn/problems/nth-magical-number/
   - 这是本题的来源，是一道经典题

8. POJ 2142 The Balance (Code11_Poj2142_TheBalance)

问题描述
给定 a、b、c，求解方程 ax + by = c，要求找到一组解(x,y)，使得|x| + |y|最小。如果有多个解，选择 x 最小的解。

解题思路
1. 使用扩展欧几里得算法求解 ax + by = gcd(a,b)的一组特解
2. 判断方程是否有解：当 c 能被 gcd(a,b)整除时有解
3. 如果有解，将特解乘以 c/gcd(a,b)得到原方程的一组特解
4. 根据通解公式求出满足条件的解
5. 在所有解中寻找|x| + |y|最小的解

相关题目
1. **POJ 2142 The Balance**
   - 链接：http://poj.org/problem?id=2142
   - 这是本题的来源，是一道经典题

9. Codeforces 7C. Line (Code12_Codeforces7C_Line)

问题描述
给定直线方程 Ax + By + C = 0，求直线上任意一个整数点(x,y)。如果不存在整数点，输出-1。

解题思路
1. 将直线方程转换为标准形式：Ax + By = -C
2. 使用扩展欧几里得算法求解方程 Ax + By = gcd(A,B)的一组特解
3. 判断方程是否有整数解：当-C 能被 gcd(A,B)整除时有解
4. 如果有解，将特解乘以(-C)/gcd(A,B)得到原方程的一组特解

相关题目
1. **Codeforces 7C. Line**
   - 链接：https://codeforces.com/problemset/problem/7C
   - 这是本题的来源，是一道经典题

10. UVA 10090 Marbles (Code13_Uva10090_Marbles)

问题描述
有两种盒子：第一种盒子可以装 n1 个弹珠，价格为 c1；第二种盒子可以装 n2 个弹珠，价格为 c2。需要装恰好 n 个弹珠，求最小总价格。如果无法恰好装 n 个弹珠，输出"failed"。

解题思路
1. 设第一种盒子用 x 个，第二种盒子用 y 个，则方程为：n1*x + n2*y = n
2. 使用扩展欧几里得算法求解方程
3. 判断方程是否有解：当 n 能被 gcd(n1,n2)整除时有解
4. 如果有解，根据通解公式求出所有可能的解
5. 在所有解中寻找 c1*x + c2*y 最小的解

相关题目
1. **UVA 10090 Marbles**
   - 链接：
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1031
   - 这是本题的来源，是一道经典题

算法技巧总结

见到什么样的题目用这种数据结构与算法

1. **线性丢番图方程问题**
   - 特征：涉及形如 ax + by = c 的方程求解
   - 适用算法：扩展欧几里得算法

2. **同余方程问题**
   - 特征：涉及模运算的方程
   - 适用算法：扩展欧几里得算法转化为线性丢番图方程

3. **格点计数问题**
   - 特征：涉及网格点上的几何计算
   - 适用算法：最大公约数、Pick 定理

4. **硬币问题**
   - 特征：涉及用固定面值硬币组成金额
   - 适用算法：赛瓦维斯特定理

工程化考量

1. 异常处理
- 输入验证：检查输入参数的有效性
- 特殊情况处理：处理边界输入、极端数据

- 错误信息清晰提示

2. 性能优化
- 时间复杂度优化：通过数学方法优化算法复杂度
- 空间复杂度优化：使用原地操作，减少内存占用
- 防止溢出：使用适当的数据类型处理大数运算

3. 可读性
- 变量命名：使用有意义的变量名
- 注释完整：为每个方法和关键步骤添加详细注释
- 模块化：将复杂逻辑拆分为独立函数

4. 跨语言实现
- Java 版本：面向对象实现，详细注释
- C++版本：高效实现，适合竞赛
- Python 版本：简洁实现，适合快速验证

复杂度分析

时间复杂度
- 扩展欧几里得算法：O(log(min(a,b)))
- 最大公约数计算：O(log(min(a,b)))
- Pick 定理应用：O(n*log(max(dx,dy)))

空间复杂度
- 扩展欧几里得算法：O(log(min(a,b)))（递归调用栈）
- 其他算法：O(1)

学习建议

1. 熟练掌握扩展欧几里得算法的原理和实现
2. 理解线性丢番图方程的解法和应用
3. 掌握 Pick 定理和赛瓦维斯特定理的应用场景
4. 多做练习题，加深对算法本质的理解
5. 注意算法在工程实践中的应用

参考资料

1. 《算法导论》
2. 《算法竞赛入门经典》
3. 《挑战程序设计竞赛》
4. 各大 OJ 平台的官方题解

==

文件: SummaryAndPatterns.md
==
Class140 总结与模式识别

核心知识点总结

1. 扩展欧几里得算法
扩展欧几里得算法是解决线性丢番图方程的核心工具，不仅能计算最大公约数，还能找到方程 $ax + by = \gcd(a,b)$ 的一组特解。

核心思想
通过递归回溯的方式，从简单的边界情况逐步构建复杂问题的解。

应用场景
- 求解线性丢番图方程
- 计算模逆元
- 解决同余方程

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）

2. 线性丢番图方程
形如 $ax + by = c$ 的方程，其中 a、b、c 为整数，求整数解 x 和 y。

解的存在性
方程 $ax + by = c$ 有整数解当且仅当 $\gcd(a,b)$ 能整除 c。

通解公式
如果 $(x0,y0)$ 是 $ax + by = c$ 的一组特解，那么通解为：
- $x = x0 + (b/\gcd(a,b)) * t$
- $y = y0 - (a/\gcd(a,b)) * t$
其中 t 为任意整数。

3. Pick 定理
用于计算顶点均为格点的简单多边形的面积。

公式
$A = i + b/2 - 1$

其中 A 是多边形面积，i 是内部格点数，b 是边界格点数。

应用场景
- 计算格点多边形面积
- 统计格点数量

4. 赛瓦维斯特定理
当正整数 a 和 b 互质时，不能表示为 ax+by（x,y≥0）的最大整数是 ab-a-b。

应用场景
- 硬币问题
- 数论问题

题型识别与解法选择

1. 线性丢番图方程类问题
识别特征：
- 涉及形如 ax + by = c 的方程
- 求整数解或判断解的存在性
- 涉及最大公约数的计算

解题思路：
1. 使用扩展欧几里得算法求解 ax + by = gcd(a,b)的一组特解
2. 判断方程是否有解：当 c 能被 gcd(a,b)整除时有解
3. 如果有解，将特解乘以 c/gcd(a,b)得到原方程的一组特解
4. 根据通解公式求出满足条件的解

典型题目：
- 洛谷 P5656 【模板】二元一次不定方程 (exgcd)
- LeetCode 1250. 检查「好数组」
- Codeforces 1244C. The Football Stage

2. 同余方程类问题
识别特征：
- 涉及模运算的方程
- 形如 ax ≡ b (mod m)的表达式
- 需要求解满足条件的整数

解题思路：
1. 将同余方程转化为线性丢番图方程：ax + my = b
2. 使用扩展欧几里得算法求解
3. 根据通解公式求出满足条件的解

典型题目：
- 洛谷 P1516 青蛙的约会
- POJ 1061 青蛙的约会
- POJ 2115 C Looooops

3. 格点计数类问题
识别特征：
- 涉及网格点上的几何计算
- 需要求解线段或区域上的格点数量
- 涉及最大公约数的应用

解题思路：
1. 线段上的格点数量等于 dx 和 dy 的最大公约数加 1
2. 多边形边界上的格点数量使用 gcd 计算每条边上的格点数累加
3. 多边形内部的格点数量使用 Pick 定理计算

典型题目：
- LightOJ 1077 How Many Points?
- POJ 1265 Area
- UVA 10088 - Trees on My Island

4. 硬币类问题
识别特征：
- 涉及用固定面值硬币组成金额
- 求无法组成的最大金额
- 涉及互质数的应用

解题思路：
1. 根据赛瓦维斯特定理，当 a 和 b 互质时，无法表示的最大整数是 ab-a-b
2. 对于更复杂的情况，可能需要使用动态规划或其他方法

典型题目：
- 洛谷 P3951 [NOIP2017 提高组] 小凯的疑惑
- HDU 1792 A New Change Problem

工程化考量

1. 异常处理
- **输入验证**：检查输入参数的有效性，如是否为正整数、是否满足约束条件等
- **边界条件**：处理特殊情况，如 a 或 b 为 1 的情况
- **错误信息**：提供清晰的错误提示信息

2. 性能优化

- **时间复杂度优化**：通过数学方法优化算法复杂度，避免不必要的计算
- **空间复杂度优化**：使用原地操作，减少内存占用
- **防止溢出**：使用适当的数据类型处理大数运算，注意中间计算过程

3. 可读性
- **变量命名**：使用有意义的变量名，如 a、b、c 表示方程系数，x、y 表示未知数
- **注释完整**：为每个方法和关键步骤添加详细注释，解释算法思路和数学原理
- **模块化**：将复杂逻辑拆分为独立函数，提高代码复用性和可维护性

4. 跨语言实现
- **Java 版本**：面向对象实现，详细注释，适合工程应用
- **C++版本**：高效实现，适合竞赛，注意内存管理
- **Python 版本**：简洁实现，适合快速验证，注意性能问题

算法调试与问题定位

1. 打印中间过程
在关键步骤添加打印语句，观察变量的变化过程，快速定位错误。

2. 用断言验证中间结果
使用断言验证关键中间结果的正确性，及时发现逻辑错误。

3. 性能退化的排查方法
- 分析算法的时间复杂度是否符合预期
- 检查是否存在重复计算
- 优化数据结构和算法实现

与标准库实现的对比

1. 标准库的边界处理
标准库通常具有更完善的边界条件处理机制，能够处理各种异常输入。

2. 全面的异常防御
标准库具有更全面的异常防御机制，能够处理各种异常情况。

3. 极端数据规模的优化策略
标准库针对极端数据规模进行了优化，具有更好的性能表现。

算法安全与业务适配

1. 避免崩溃
通过合理的异常处理机制，避免程序因异常输入而崩溃。

2. 异常捕获
使用 try-catch 等机制捕获和处理异常，保证程序的稳定性。

3. 处理溢出
使用适当的数据类型和算法，防止计算过程中的溢出问题。

文档化和使用说明

1. 代码注释
为每个函数和关键步骤添加详细的注释，解释算法思路和实现细节。

2. 使用说明
提供清晰的使用说明，包括输入输出格式、参数说明等。

3. 常见问题排查
总结常见问题和解决方案，帮助用户快速解决问题。

面试准备指南

1. 知识点掌握
- 熟练掌握扩展欧几里得算法的原理和实现
- 理解线性丢番图方程的解法和应用
- 掌握 Pick 定理和赛瓦维斯特定理的应用场景

2. 解题思路
- 拆解题干核心需求，提取输入输出约束
- 明确目标任务，让面试官认可你的理解深度
- 能够清晰地表达算法思路和实现过程

3. 代码效率优化
- 时间优化：避免冗余循环、减少重复计算
- 空间优化：能原地就不额外开空间

4. 多解法对比与最优解选择
- 能够分析不同解法的时间和空间复杂度
- 选择最适合的解法解决具体问题

学习路径建议

初级阶段
1. 掌握扩展欧几里得算法的基本实现
2. 理解线性丢番图方程的解法
3. 熟悉最大公约数的计算方法

中级阶段
1. 学习 Pick 定理的应用
2. 掌握赛瓦维斯特定理的使用场景
3. 练习将实际问题转化为数学模型

高级阶段
1. 研究数论在算法竞赛中的高级应用
2. 学习更多数论定理和算法
3. 参与编程竞赛，提升解题能力

实战阶段
1. 在各大 OJ 平台刷题巩固
2. 参与编程竞赛提升能力
3. 总结经验形成解题模板

总结

Class140 涵盖的数论算法是算法竞赛和实际工程中的重要工具。通过深入理解这些算法的原理和实现，可以在实际工作中更好地应用它们解决复杂问题。在学习过程中，要注重理论与实践相结合，多做练习题，加深对算法本质的理解。

==

[代码文件]
==
文件: Code01_DiophantineEquation.cpp
==
```
// 二元一次不定方程模版
// 给定 a、b、c，求解方程 ax + by = c
// 如果方程无解打印-1
// 如果方程无正整数解，但是有整数解
// 打印这些整数解中，x 的最小正数值，y 的最小正数值
// 如果方程有正整数解，打印正整数解的数量，同时打印所有正整数解中，
// x 的最小正数值，y 的最小正数值，x 的最大正数值，y 的最大正数值
// 1 <= a、b、c <= 10^9
// 测试链接 : https://www.luogu.com.cn/problem/P5656

// 全局变量
long long d, x, y, px, py;

/**
 * 扩展欧几里得算法
```

```
 * 求解方程 ax + by = gcd(a,b)的一组特解
 *
 * 算法原理：
 * 当 b=0 时，gcd(a,b)=a，此时 x=1,y=0
 * 当 b≠0 时，递归计算 gcd(b,a%b)的解，然后根据推导公式得到原方程的解
 *
 * 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 系数 a
 * @param b 系数 b
 */
void exgcd(long long a, long long b) {
    if (b == 0) {
        d = a;
        x = 1;
        y = 0;
    } else {
        exgcd(b, a % b);
        px = x;
        py = y;
        x = py;
        y = px - py * (a / b);
    }
}

long long a, b, c, xd, yd, times;

/**
 * 主函数
 *
 * 问题描述：
 * 给定 a、b、c，求解方程 ax + by = c
 *
 * 解题思路：
 * 1. 使用扩展欧几里得算法求解 ax + by = gcd(a,b)的一组特解
 * 2. 判断方程是否有解：当 c 能被 gcd(a,b)整除时有解
 * 3. 如果有解，将特解乘以 c/gcd(a,b)得到原方程的一组特解
 * 4. 根据通解公式求出满足条件的解
 *
 * 数学原理：
 * 1. 裴蜀定理：方程 ax + by = c 有整数解当且仅当 gcd(a,b)能整除 c
 * 2. 扩展欧几里得算法：求解 ax + by = gcd(a,b)的一组特解
```

```
* 3. 通解公式：如果(x0,y0)是 ax + by = c 的一组特解，那么通解为：
*    x = x0 + (b/gcd(a,b)) * t
*    y = y0 - (a/gcd(a,b)) * t
*    其中 t 为任意整数
*
* 时间复杂度：O(log(min(a,b)))，主要消耗在扩展欧几里得算法上
* 空间复杂度：O(1)
*
* 相关题目：
* 1. 洛谷 P5656 【模板】二元一次不定方程 (exgcd)
*    链接：https://www.luogu.com.cn/problem/P5656
*    这是本题的来源，是一道模板题
*
* 2. LeetCode 1250. 检查「好数组」
*    链接：https://leetcode.cn/problems/check-if-it-is-a-good-array/
*    本题用到了裴蜀定理，如果数组中所有元素的最大公约数为 1，则为好数组
*
* 3. Codeforces 1244C. The Football Stage
*    链接：https://codeforces.com/problemset/problem/1244/C
*    本题需要求解线性丢番图方程 wx + dy = p，其中 w 和 d 是给定的，p 是变量
*
* 4. HDU 5512 Pagodas
*    链接：https://acm.hdu.edu.cn/showproblem.php?pid=5512
*    本题涉及数论知识，与最大公约数有关
*
* 5. POJ 2115 C Looooops
*    链接：http://poj.org/problem?id=2115
*    本题需要求解模线性方程，可以转化为线性丢番图方程
*
* 6. POJ 1061 青蛙的约会
*    链接：http://poj.org/problem?id=1061
*    本题需要求解同余方程，是扩展欧几里得算法的经典应用
*
* 7. LightOJ 1077 How Many Points?
*    链接：https://lightoj.com/problem/how-many-points
*    本题涉及最大公约数的应用，计算线段上的格点数量
*
* 8. HDU 1792 A New Change Problem
*    链接：https://acm.hdu.edu.cn/showproblem.php?pid=1792
*    本题是硬币问题的变形，求无法表示的最大数和无法表示的数的个数
*
* 9. UVA 10088 - Trees on My Island
*    链接：
```

```
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1029
 *     本题需要使用 Pick 定理计算多边形内部的格点数量
 *
 * 10. Codeforces 514B Han Solo and Lazer Gun
 *      链接：https://codeforces.com/problemset/problem/514B
 *      本题涉及最大公约数的应用，计算点在同一直线上的数量
 *
 * 11. AtCoder ABC161 D Lunlun Number
 *      链接：https://atcoder.jp/contests/abc161/tasks/abc161_d
 *      本题使用 BFS 和数学方法，与数论相关
 *
 * 12. AtCoder ARC084 B Small Multiple
 *      链接：https://atcoder.jp/contests/abc077/tasks/arc084_b
 *      本题使用 01-BFS，与数论相关
 *
 * 13. HDU 5722 Jewelry
 *      链接：https://acm.hdu.edu.cn/showproblem.php?pid=5722
 *      本题涉及最大公约数的应用
 *
 * 工程化考虑：
 * 1. 异常处理：需要处理输入非法、方程无解等情况
 * 2. 边界条件：需要考虑 a、b、c 为边界值的情况
 * 3. 性能优化：对于大数据，要注意算法的时间复杂度
 * 4. 可读性：添加详细注释，变量命名清晰
 * 5. 错误信息：提供清晰的错误提示信息
 * 6. 防止溢出：使用适当的数据类型处理大数运算
 *
 * 算法要点：
 * 1. 扩展欧几里得算法是解决此类问题的核心
 * 2. 裴蜀定理是判断方程是否有解的依据
 * 3. 通解公式是找出所有解的关键
 * 4. 对于正整数解，需要通过调整特解来找到满足条件的解
 *
 * 调试技巧：
 * 1. 打印中间过程：在关键步骤添加打印语句，观察变量的变化过程
 * 2. 用断言验证中间结果：使用断言验证关键中间结果的正确性
 * 3. 性能退化排查：分析算法的时间复杂度是否符合预期
 *
 * 与标准库对比：
 * 标准库中的 gcd 函数通常只返回最大公约数，而扩展欧几里得算法还能找到 x 和 y
 * 标准库通常具有更完善的边界条件处理和异常防御机制
 */
#include <iostream>
```

```
using namespace std;

int main() {
    int cases;
    cin >> cases;
    while (cases--) {
        cin >> a >> b >> c;
        exgcd(a, b);
        if (c % d != 0) { // 无整数解
            cout << -1 << endl;
        } else { // 有整数解
            x *= c / d;
            y *= c / d;
            xd = b / d;
            yd = a / d;
            if (x < 0) {
                // x 要想增长到>=1 且最小的值，差几个 xd，算出来就是 k 的值
                // 那应该是(1-x)/xd，结果向上取整
                times = (1 - x + xd - 1) / xd;
                x += xd * times;
                y -= yd * times;
            } else {
                // x 要想减少到>=1 且最小的值，差几个 xd，算出来就是 k 的值，向下取整
                times = (x - 1) / xd;
                x -= xd * times;
                y += yd * times;
            }
            // 此时得到的(x, y)，是 x 为最小正整数时的一组解
            // 然后继续讨论
            if (y <= 0) { // 无正整数解
                // x 能取得的最小正数
                cout << x << " ";
                // y 能取得的最小正数
                cout << y + yd * ((1 - y + yd - 1) / yd) << endl;
            } else { // 有正整数解
                // y 减少到 1 以下，能减几次，就是正整数解的个数
                cout << ((y - 1) / yd + 1) << " ";
                // x 能取得的最小正数
                cout << x << " ";
                // y 能取得的最小正数
                cout << (y - (y - 1) / yd * yd) << " ";
                // x 能取得的最大正数
                cout << (x + (y - 1) / yd * xd) << " ";
```

```
                // y能取得的最大正数
                cout << y << endl;
            }
        }
    }
    return 0;
}
```

==

文件: Code01_DiophantineEquation.py

==

```python
# 二元一次不定方程模版
# 给定a、b、c，求解方程ax + by = c
# 如果方程无解打印-1
# 如果方程无正整数解，但是有整数解
# 打印这些整数解中，x的最小正数值，y的最小正数值
# 如果方程有正整数解，打印正整数解的数量，同时打印所有正整数解中，
# x的最小正数值，y的最小正数值，x的最大正数值，y的最大正数值
# 1 <= a、b、c <= 10^9
# 测试链接 : https://www.luogu.com.cn/problem/P5656
# 提交时请将类名改成"Main"，可以通过所有测试用例

import sys
import math

# 扩展欧几里得算法
def exgcd(a, b):
    """
    扩展欧几里得算法
    求解方程ax + by = gcd(a,b)的一组特解

    算法原理:
    当b=0时，gcd(a,b)=a，此时x=1,y=0
    当b≠0时，递归计算gcd(b,a%b)的解，然后根据推导公式得到原方程的解

    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

    Args:
        a: 系数a
        b: 系数b
```

```
    Returns:
        (d, x, y): d=gcd(a,b), x和y是方程ax + by = gcd(a,b)的一组特解

    异常情况:
        - 当a和b都为0时，gcd无定义，函数将返回(0, 1, 0)

    工程化考虑:
        - 递归实现简洁但可能在极端情况下导致栈溢出
        - 可以优化为迭代版本以提高效率
    """
    if b == 0:
        return a, 1, 0
    else:
        d, x, y = exgcd(b, a % b)
        return d, y, x - y * (a // b)

def main():
    """
    主函数

    问题描述:
    给定a、b、c，求解方程ax + by = c

    解题思路:
    1. 使用扩展欧几里得算法求解ax + by = gcd(a,b)的一组特解
    2. 判断方程是否有解：当c能被gcd(a,b)整除时有解
    3. 如果有解，将特解乘以c/gcd(a,b)得到原方程的一组特解
    4. 根据通解公式求出满足条件的解

    数学原理:
    1. 裴蜀定理：方程ax + by = c有整数解当且仅当gcd(a,b)能整除c
    2. 扩展欧几里得算法：求解ax + by = gcd(a,b)的一组特解
    3. 通解公式：如果(x0,y0)是ax + by = c的一组特解，那么通解为:
       x = x0 + (b/gcd(a,b)) * t
       y = y0 - (a/gcd(a,b)) * t
       其中t为任意整数

    时间复杂度：O(log(min(a,b)))，主要消耗在扩展欧几里得算法上
    空间复杂度：O(log(min(a,b)))，递归调用栈的深度

    相关题目:
    1. 洛谷 P5656 【模板】二元一次不定方程 (exgcd)
       链接：https://www.luogu.com.cn/problem/P5656
```

这是本题的来源，是一道模板题

2. LeetCode 1250. 检查「好数组」
   链接：https://leetcode.cn/problems/check-if-it-is-a-good-array/
   本题用到了裴蜀定理，如果数组中所有元素的最大公约数为 1，则为好数组

3. Codeforces 1244C. The Football Stage
   链接：https://codeforces.com/problemset/problem/1244/C
   本题需要求解线性丢番图方程 wx + dy = p，其中 w 和 d 是给定的，p 是变量

4. HDU 5512 Pagodas
   链接：https://acm.hdu.edu.cn/showproblem.php?pid=5512
   本题涉及数论知识，与最大公约数有关

5. POJ 2115 C Looooops
   链接：http://poj.org/problem?id=2115
   本题需要求解模线性方程，可以转化为线性丢番图方程

6. POJ 1061 青蛙的约会
   链接：http://poj.org/problem?id=1061
   本题需要求解同余方程，是扩展欧几里得算法的经典应用

7. LightOJ 1077 How Many Points?
   链接：https://lightoj.com/problem/how-many-points
   本题涉及最大公约数的应用，计算线段上的格点数量

8. HDU 1792 A New Change Problem
   链接：https://acm.hdu.edu.cn/showproblem.php?pid=1792
   本题是硬币问题的变形，求无法表示的最大数和无法表示的数的个数

9. UVA 10088 - Trees on My Island
   链接：
   https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1029
   本题需要使用 Pick 定理计算多边形内部的格点数量

10. Codeforces 514B Han Solo and Lazer Gun
    链接：https://codeforces.com/problemset/problem/514B
    本题涉及最大公约数的应用，计算点在同一直线上的数量

11. AtCoder ABC161 D Lunlun Number
    链接：https://atcoder.jp/contests/abc161/tasks/abc161_d
    本题使用 BFS 和数学方法，与数论相关

12. AtCoder ARC084 B Small Multiple
    链接：https://atcoder.jp/contests/abc077/tasks/arc084_b
    本题使用 01-BFS，与数论相关

13. HDU 5722 Jewelry
    链接：https://acm.hdu.edu.cn/showproblem.php?pid=5722
    本题涉及最大公约数的应用

14. LeetCode 365. 水壶问题
    链接：https://leetcode.cn/problems/water-and-jug-problem/
    本题可以用裴蜀定理解决，判断是否存在非负整数 x,y 使得 ax + by = z

15. 剑指 Offer 44. 数字序列中某一位的数字
    链接：https://leetcode.cn/problems/shu-zi-xu-lie-zhong-mou-yi-wei-de-shu-zi-lcof/
    本题涉及数学规律的应用

工程化考虑：
1. 异常处理：需要处理输入非法、方程无解等情况
2. 边界条件：需要考虑 a、b、c 为边界值的情况
3. 性能优化：对于大数据，要注意算法的时间复杂度
4. 可读性：添加详细注释，变量命名清晰
5. 模块化：将复杂逻辑拆分为独立函数
6. 可测试性：添加单元测试用例
7. 防止溢出：使用 Python 的大整数特性，无需额外处理

算法要点：
1. 扩展欧几里得算法是解决此类问题的核心
2. 裴蜀定理是判断方程是否有解的依据
3. 通解公式是找出所有解的关键
4. 对于正整数解，需要通过调整特解来找到满足条件的解

调试技巧：
1. 打印中间过程：在关键步骤添加打印语句，观察变量的变化过程
2. 使用断言：验证关键中间结果的正确性
3. 小例子测试：使用简单的测试用例验证算法的正确性

跨语言差异：
1. Python 的递归深度限制可能在极端情况下导致问题，可考虑迭代实现
2. Python 的整数类型没有大小限制，无需考虑溢出问题
3. 输入输出效率：在 Python 中使用 sys.stdin.readline 比 input()更高效
"""

```
# 读取测试用例数量
cases = int(sys.stdin.readline().strip())
```

```python
for _ in range(cases):
    # 读取a, b, c
    line = sys.stdin.readline().strip().split()
    a = int(line[0])
    b = int(line[1])
    c = int(line[2])

    # 异常输入检查
    if a <= 0 or b <= 0 or c <= 0:
        print(-1)
        continue

    # 使用扩展欧几里得算法求解
    d, x, y = exgcd(a, b)

    # 判断方程是否有解：当c能被gcd(a,b)整除时有解
    if c % d != 0:
        # 无整数解
        print(-1)
    else:
        # 有整数解
        # 将特解乘以c/gcd(a,b)得到原方程的一组特解
        x *= c // d
        y *= c // d

        # 计算通解中的系数
        xd = b // d  # x方向的周期
        yd = a // d  # y方向的周期

        # 调整x为最小正整数
        if x < 0:
            # x要想增长到>=1且最小的值，差几个xd，算出来就是k的值
            # 那应该是(1-x)/xd，结果向上取整
            times = (1 - x + xd - 1) // xd
            x += xd * times
            y -= yd * times
        else:
            # x要想减少到>=1且最小的值，差几个xd，算出来就是k的值，向下取整
            times = (x - 1) // xd
            x -= xd * times
            y += yd * times
```

```python
            # 此时得到的(x, y)，是 x 为最小正整数时的一组解
            # 然后继续讨论
            if y <= 0:
                # 无正整数解
                # 计算 y 能取得的最小正数
                y_min_positive = y + yd * ((1 - y + yd - 1) // yd)
                print(x, y_min_positive)
            else:
                # 有正整数解
                # y 减少到 1 以下，能减几次，就是正整数解的个数
                count = (y - 1) // yd + 1
                # x 能取得的最小正数
                min_x = x
                # y 能取得的最小正数
                min_y = y - (y - 1) // yd * yd
                # x 能取得的最大正数
                max_x = x + (y - 1) // yd * xd
                # y 能取得的最大正数
                max_y = y
                print(count, min_x, min_y, max_x, max_y)

if __name__ == "__main__":
    main()
```

==

文件: Code01_DiophantineEquation1.java

==

```java
package class140;

// 二元一次不定方程模版
// 给定 a、b、c，求解方程 ax + by = c
// 如果方程无解打印-1
// 如果方程无正整数解，但是有整数解
// 打印这些整数解中，x 的最小正数值，y 的最小正数值
// 如果方程有正整数解，打印正整数解的数量，同时打印所有正整数解中，
// x 的最小正数值，y 的最小正数值，x 的最大正数值，y 的最大正数值
// 1 <= a、b、c <= 10^9
// 测试链接 : https://www.luogu.com.cn/problem/P5656
// 如下实现是正确的，但是洛谷平台对空间卡的很严，只有使用 C++能全部通过
// java 的版本就是无法完全通过的，空间会超过限制，主要是 IO 空间占用大
// 这是洛谷平台没有照顾各种语言的实现所导致的
// 在真正笔试、比赛时，一定是兼顾各种语言的，该实现是一定正确的
```

```java
// C++版本就是Code01_DiophantineEquation2文件
// C++版本和java版本逻辑完全一样，但只有C++版本可以通过所有测试用例

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.StreamTokenizer;

/**
 * 二元一次不定方程求解器
 * 使用扩展欧几里得算法求解线性丢番图方程
 *
 * 核心算法：扩展欧几里得算法
 * 相关数学定理：裴蜀定理
 *
 * 时间复杂度：O(log(min(a,b)))
 * 空间复杂度：O(log(min(a,b)))，递归调用栈深度
 *
 * 问题描述：
 * 给定a、b、c，求解方程ax + by = c
 *
 * 解题思路：
 * 1. 使用扩展欧几里得算法求解ax + by = gcd(a,b)的一组特解
 * 2. 判断方程是否有解：当c能被gcd(a,b)整除时有解
 * 3. 如果有解，将特解乘以c/gcd(a,b)得到原方程的一组特解
 * 4. 根据通解公式求出满足条件的解
 *
 * 数学原理：
 * 1. 裴蜀定理：方程ax + by = c有整数解当且仅当gcd(a,b)能整除c
 * 2. 扩展欧几里得算法：求解ax + by = gcd(a,b)的一组特解
 * 3. 通解公式：如果(x0,y0)是ax + by = c的一组特解，那么通解为：
 *    x = x0 + (b/gcd(a,b)) * t
 *    y = y0 - (a/gcd(a,b)) * t
 *    其中t为任意整数
 *
 * 相关题目：
 * 1. 洛谷 P5656 【模板】二元一次不定方程 (exgcd)
 *    链接：https://www.luogu.com.cn/problem/P5656
 *    这是本题的来源，是一道模板题
 *
 * 2. LeetCode 1250. 检查「好数组」
```

```
 *    链接：https://leetcode.cn/problems/check-if-it-is-a-good-array/
 *    本题用到了裴蜀定理，如果数组中所有元素的最大公约数为 1，则为好数组
 *
 * 3. Codeforces 1244C. The Football Stage
 *    链接：https://codeforces.com/problemset/problem/1244/C
 *    本题需要求解线性丢番图方程 wx + dy = p，其中 w 和 d 是给定的，p 是变量
 *
 * 4. HDU 5512 Pagodas
 *    链接：https://acm.hdu.edu.cn/showproblem.php?pid=5512
 *    本题涉及数论知识，与最大公约数有关
 *
 * 5. POJ 2115 C Looooops
 *    链接：http://poj.org/problem?id=2115
 *    本题需要求解模线性方程，可以转化为线性丢番图方程
 *
 * 6. POJ 1061 青蛙的约会
 *    链接：http://poj.org/problem?id=1061
 *    本题需要求解同余方程，是扩展欧几里得算法的经典应用
 *
 * 7. LightOJ 1077 How Many Points?
 *    链接：https://lightoj.com/problem/how-many-points
 *    本题涉及最大公约数的应用，计算线段上的格点数量
 *
 * 8. HDU 1792 A New Change Problem
 *    链接：https://acm.hdu.edu.cn/showproblem.php?pid=1792
 *    本题是硬币问题的变形，求无法表示的最大数和无法表示的数的个数
 *
 * 9. UVA 10088 - Trees on My Island
 *    链接：
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1029
 *    本题需要使用 Pick 定理计算多边形内部的格点数量
 *
 * 10. Codeforces 514B Han Solo and Lazer Gun
 *     链接：https://codeforces.com/problemset/problem/514B
 *     本题涉及最大公约数的应用，计算点在同一直线上的数量
 *
 * 11. AtCoder ABC161 D Lunlun Number
 *     链接：https://atcoder.jp/contests/abc161/tasks/abc161_d
 *     本题使用 BFS 和数学方法，与数论相关
 *
 * 12. AtCoder ARC084 B Small Multiple
 *     链接：https://atcoder.jp/contests/abc077/tasks/arc084_b
 *     本题使用 01-BFS，与数论相关
```

```
 *
 * 13. HDU 5722 Jewelry
 *     链接：https://acm.hdu.edu.cn/showproblem.php?pid=5722
 *     本题涉及最大公约数的应用
 *
 * 14. LeetCode 365. 水壶问题
 *     链接：https://leetcode.cn/problems/water-and-jug-problem/
 *     本题可以用裴蜀定理解决，判断是否存在非负整数 x,y 使得 ax + by = z
 *
 * 15. 剑指 Offer 44. 数字序列中某一位的数字
 *     链接：https://leetcode.cn/problems/shu-zi-xu-lie-zhong-mou-yi-wei-de-shu-zi-lcof/
 *     本题涉及数学规律的应用
 *
 * 工程化考虑：
 * 1. 异常处理：需要处理输入非法、方程无解等情况
 * 2. 边界条件：需要考虑 a、b、c 为边界值的情况
 * 3. 性能优化：对于大数据，使用 StreamTokenizer 提高输入效率
 * 4. 可读性：添加详细注释，变量命名清晰
 * 5. 模块化：将复杂逻辑拆分为独立方法
 * 6. 可测试性：添加单元测试用例
 * 7. 防止溢出：使用 long 类型处理大数运算
 * 8. 输入输出优化：使用 BufferedReader 和 PrintWriter 提高 IO 效率
 * 9. 线程安全：考虑多线程环境下的安全性
 * 10. 错误信息：提供清晰的错误提示信息
 *
 * 算法要点：
 * 1. 扩展欧几里得算法是解决此类问题的核心
 * 2. 裴蜀定理是判断方程是否有解的依据
 * 3. 通解公式是找出所有解的关键
 * 4. 对于正整数解，需要通过调整特解来找到满足条件的解
 *
 * 调试技巧：
 * 1. 打印中间过程：在关键步骤添加打印语句，观察变量的变化过程
 * 2. 使用断言：验证关键中间结果的正确性
 * 3. 小例子测试：使用简单的测试用例验证算法的正确性
 *
 * 跨语言差异：
 * 1. Java 的递归深度限制为 1000，可以通过调整 JVM 参数或迭代实现解决
 * 2. Java 的 long 类型范围为-9,223,372,036,854,775,808 到 9,223,372,036,854,775,807，
 *    对于超过这个范围的整数需要使用 BigInteger
 * 3. 输入输出效率：在 Java 中，BufferedReader 和 StreamTokenizer 比 Scanner 更高效
 */
public class Code01_DiophantineEquation1 {
```

```java
// 扩展欧几里得算法相关变量
public static long d, x, y, px, py;

/**
 * 扩展欧几里得算法
 * 求解方程 ax + by = gcd(a,b)的一组特解
 *
 * 算法原理：
 * 当 b=0 时，gcd(a,b)=a，此时 x=1,y=0
 * 当 b≠0 时，递归计算 gcd(b,a%b)的解，然后根据推导公式得到原方程的解
 *
 * 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 系数 a
 * @param b 系数 b
 *
 * 异常情况：
 * - 当 a 和 b 都为 0 时，gcd 无定义，函数行为未定义
 *
 * 工程化考虑：
 * - 递归实现简洁但可能在极端情况下导致栈溢出
 * - 可以优化为迭代版本以提高效率
 */
public static void exgcd(long a, long b) {
    if (b == 0) {
        d = a;
        x = 1;
        y = 0;
    } else {
        exgcd(b, a % b);
        px = x;
        py = y;
        x = py;
        y = px - py * (a / b);
    }
}

public static long a, b, c, xd, yd, times;

/**
 * 主函数
```

```
*
* 问题描述：
* 给定 a、b、c，求解方程 ax + by = c
*
* 解题思路：
* 1. 使用扩展欧几里得算法求解 ax + by = gcd(a,b)的一组特解
* 2. 判断方程是否有解：当 c 能被 gcd(a,b)整除时有解
* 3. 如果有解，将特解乘以 c/gcd(a,b)得到原方程的一组特解
* 4. 根据通解公式求出满足条件的解
*
* 数学原理：
* 1. 裴蜀定理：方程 ax + by = c 有整数解当且仅当 gcd(a,b)能整除 c
* 2. 扩展欧几里得算法：求解 ax + by = gcd(a,b)的一组特解
* 3. 通解公式：如果(x0,y0)是 ax + by = c 的一组特解，那么通解为：
*    x = x0 + (b/gcd(a,b)) * t
*    y = y0 - (a/gcd(a,b)) * t
*    其中 t 为任意整数
*
* 时间复杂度：O(log(min(a,b)))，主要消耗在扩展欧几里得算法上
* 空间复杂度：O(log(min(a,b)))，递归调用栈的深度
*
* 相关题目：
* 1. 洛谷 P5656 【模板】二元一次不定方程 (exgcd)
*    链接：https://www.luogu.com.cn/problem/P5656
*    这是本题的来源，是一道模板题
*
* 2. LeetCode 1250. 检查「好数组」
*    链接：https://leetcode.cn/problems/check-if-it-is-a-good-array/
*    本题用到了裴蜀定理，如果数组中所有元素的最大公约数为 1，则为好数组
*
* 3. Codeforces 1244C. The Football Stage
*    链接：https://codeforces.com/problemset/problem/1244/C
*    本题需要求解线性丢番图方程 wx + dy = p，其中 w 和 d 是给定的，p 是变量
*
* 4. HDU 5512 Pagodas
*    链接：https://acm.hdu.edu.cn/showproblem.php?pid=5512
*    本题涉及数论知识，与最大公约数有关
*
* 5. POJ 2115 C Looooops
*    链接：http://poj.org/problem?id=2115
*    本题需要求解模线性方程，可以转化为线性丢番图方程
*
* 6. POJ 1061 青蛙的约会
```

```
*     链接：http://poj.org/problem?id=1061
*     本题需要求解同余方程，是扩展欧几里得算法的经典应用
*
* 7. LightOJ 1077 How Many Points?
*     链接：https://lightoj.com/problem/how-many-points
*     本题涉及最大公约数的应用，计算线段上的格点数量
*
* 8. HDU 1792 A New Change Problem
*     链接：https://acm.hdu.edu.cn/showproblem.php?pid=1792
*     本题是硬币问题的变形，求无法表示的最大数和无法表示的数的个数
*
* 9. UVA 10088 - Trees on My Island
*     链接：
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1029
*     本题需要使用 Pick 定理计算多边形内部的格点数量
*
* 10. Codeforces 514B Han Solo and Lazer Gun
*      链接：https://codeforces.com/problemset/problem/514B
*      本题涉及最大公约数的应用，计算点在同一直线上的数量
*
* 11. AtCoder ABC161 D Lunlun Number
*      链接：https://atcoder.jp/contests/abc161/tasks/abc161_d
*      本题使用 BFS 和数学方法，与数论相关
*
* 12. AtCoder ARC084 B Small Multiple
*      链接：https://atcoder.jp/contests/abc077/tasks/arc084_b
*      本题使用 01-BFS，与数论相关
*
* 13. HDU 5722 Jewelry
*      链接：https://acm.hdu.edu.cn/showproblem.php?pid=5722
*      本题涉及最大公约数的应用
*
* 14. LeetCode 365. 水壶问题
*      链接：https://leetcode.cn/problems/water-and-jug-problem/
*      本题可以用裴蜀定理解决，判断是否存在非负整数 x,y 使得 ax + by = z
*
* 15. 剑指 Offer 44. 数字序列中某一位的数字
*      链接：https://leetcode.cn/problems/shu-zi-xu-lie-zhong-mou-yi-wei-de-shu-zi-lcof/
*      本题涉及数学规律的应用
*
* 工程化考虑：
* 1. 异常处理：需要处理输入非法、方程无解等情况
* 2. 边界条件：需要考虑 a、b、c 为边界值的情况
```

```
 * 3. 性能优化：对于大数据，使用 StreamTokenizer 提高输入效率
 * 4. 可读性：添加详细注释，变量命名清晰
 * 5. 模块化：将复杂逻辑拆分为独立方法
 * 6. 可测试性：添加单元测试用例
 * 7. 防止溢出：使用 long 类型处理大数运算
 * 8. 输入输出优化：使用 BufferedReader 和 PrintWriter 提高 IO 效率
 * 9. 线程安全：考虑多线程环境下的安全性
 * 10. 错误信息：提供清晰的错误提示信息
 *
 * 算法要点：
 * 1. 扩展欧几里得算法是解决此类问题的核心
 * 2. 裴蜀定理是判断方程是否有解的依据
 * 3. 通解公式是找出所有解的关键
 * 4. 对于正整数解，需要通过调整特解来找到满足条件的解
 *
 * 调试技巧：
 * 1. 打印中间过程：在关键步骤添加打印语句，观察变量的变化过程
 * 2. 使用断言：验证关键中间结果的正确性
 * 3. 小例子测试：使用简单的测试用例验证算法的正确性
 *
 * 跨语言差异：
 * 1. Java 的递归深度限制为 1000，可以通过调整 JVM 参数或迭代实现解决
 * 2. Java 的 long 类型范围为-9,223,372,036,854,775,808 到 9,223,372,036,854,775,807，
 *    对于超过这个范围的整数需要使用 BigInteger
 * 3. 输入输出效率：在 Java 中，BufferedReader 和 StreamTokenizer 比 Scanner 更高效
 *
 * @param args 命令行参数
 * @throws IOException 输入输出异常
 */
public static void main(String[] args) throws IOException {
    // 使用 BufferedReader 和 StreamTokenizer 提高输入效率
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    StreamTokenizer in = new StreamTokenizer(br);
    PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));

    // 读取测试用例数量
    in.nextToken();
    int cases = (int) in.nval;

    for (int t = 1; t <= cases; t++) {
        // 读取 a, b, c
        in.nextToken();
        a = (long) in.nval;
```

```
in.nextToken();
b = (long) in.nval;
in.nextToken();
c = (long) in.nval;

// 异常输入检查
if (a <= 0 || b <= 0 || c <= 0) {
    out.println(-1);
    continue;
}

// 使用扩展欧几里得算法求解
exgcd(a, b);

// 判断方程是否有解：当 c 能被 gcd(a,b)整除时有解
if (c % d != 0) { // 无整数解
    out.println(-1);
} else { // 有整数解
    // 将特解乘以 c/gcd(a,b)得到原方程的一组特解
    x *= c / d;
    y *= c / d;

    // 计算通解中的系数
    xd = b / d; // x 方向的周期
    yd = a / d; // y 方向的周期

    // 调整 x 为最小正整数
    if (x < 0) {
        // x 要想增长到>=1 且最小的值，差几个 xd，算出来就是 k 的值
        // 那应该是(1-x)/xd，结果向上取整
        times = (1 - x + xd - 1) / xd;
        x += xd * times;
        y -= yd * times;
    } else {
        // x 要想减少到>=1 且最小的值，差几个 xd，算出来就是 k 的值，向下取整
        times = (x - 1) / xd;
        x -= xd * times;
        y += yd * times;
    }

    // 此时得到的(x, y)，是 x 为最小正整数时的一组解
    // 然后继续讨论
    if (y <= 0) { // 无正整数解
```

```java
                // x能取得的最小正数
                out.print(x + " ");
                // y能取得的最小正数
                long yMinPositive = y + yd * ((1 - y + yd - 1) / yd);
                out.println(yMinPositive);
            } else { // 有正整数解
                // y减少到1以下，能减几次，就是正整数解的个数
                long count = (y - 1) / yd + 1;
                out.print(count + " ");
                // x能取得的最小正数
                out.print(x + " ");
                // y能取得的最小正数
                long minY = y - (y - 1) / yd * yd;
                out.print(minY + " ");
                // x能取得的最大正数
                long maxX = x + (y - 1) / yd * xd;
                out.print(maxX + " ");
                // y能取得的最大正数
                out.println(y);
            }
        }
    }

    // 确保资源释放和输出刷新
    out.flush();
    out.close();
    br.close();
  }

}
```

==

文件: Code01_DiophantineEquation2.java

==

```java
package class140;

// 二元一次不定方程模版
// 给定a、b、c，求解方程ax + by = c
// 如果方程无解打印-1
// 如果方程无正整数解，但是有整数解
// 打印这些整数解中，x的最小正数值，y的最小正数值
// 如果方程有正整数解，打印正整数解的数量，同时打印所有正整数解中，
```

```
// x的最小正数值，y的最小正数值，x的最大正数值，y的最大正数值
// 1 <= a、b、c <= 10^9
// 测试链接 : https://www.luogu.com.cn/problem/P5656
// 如下实现是C++的版本，C++版本和java版本逻辑完全一样
// 提交如下代码，可以通过所有测试用例

//#include <iostream>
//#include <cstdio>
//
//using namespace std;
//
//long long d, x, y, px, py;
//
//void exgcd(long long a, long long b) {
//    if (b == 0) {
//        d = a;
//        x = 1;
//        y = 0;
//    } else {
//        exgcd(b, a % b);
//        px = x;
//        py = y;
//        x = py;
//        y = px - py * (a / b);
//    }
//}
//
//long long a, b, c, xd, yd, times;
//
//int main() {
//    int cases;
//    scanf("%d", &cases);
//    for (int t = 1; t <= cases; t++) {
//        scanf("%lld %lld %lld", &a, &b, &c);
//        exgcd(a, b);
//        if (c % d != 0) {
//            printf("-1\n");
//        } else {
//            x *= c / d;
//            y *= c / d;
//            xd = b / d;
//            yd = a / d;
//            if (x < 0) {
```

```
//                  times = (xd - x) / xd;
//                  x += xd * times;
//                  y -= yd * times;
//              } else {
//                  times = (x - 1) / xd;
//                  x -= xd * times;
//                  y += yd * times;
//              }
//              if (y <= 0) {
//                  printf("%lld ", x);
//                  printf("%lld\n", y + yd * ((yd - y) / yd));
//              } else {
//                  printf("%lld ", ((y - 1) / yd + 1));
//                  printf("%lld ", x);
//                  printf("%lld ", (y - (y - 1) / yd * yd));
//                  printf("%lld ", (x + (y - 1) / yd * xd));
//                  printf("%lld\n", y);
//              }
//          }
//      }
//      return 0;
//}
```

===

文件: Code02_FrogsMeeting.cpp

===

```
// 青蛙的约会
// 有一个周长为 l 的环，从环的 0 位置开始，规定只能沿着顺时针方向不停转圈
// 青蛙 A 在环的 x1 位置，每秒跳 m 个单位，青蛙 B 在 x2 位置，每秒跳 n 个单位
// 只有在某时刻，青蛙 A 和青蛙 B 来到环的同一个位置，才算相遇
// 如果两只青蛙相遇不了，打印"Impossible"
// 如果可以相遇，打印两只青蛙至少多久才能相遇
// 1 <= l <= 3 * 10^9
// 1 <= x1、x2、m、n <= 2 * 10^9
// x1 != x2
// 测试链接 : https://www.luogu.com.cn/problem/P1516

// 全局变量
long long d, x, y, px, py;

/**
 * 扩展欧几里得算法
```

```
 * 求解方程ax + by = gcd(a,b)的一组特解
 *
 * 算法原理：
 * 当b=0时，gcd(a,b)=a，此时x=1,y=0
 * 当b≠0时，递归计算gcd(b,a%b)的解，然后根据推导公式得到原方程的解
 *
 * 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 系数a
 * @param b 系数b
 */
void exgcd(long long a, long long b) {
    if (b == 0) {
        d = a;
        x = 1;
        y = 0;
    } else {
        exgcd(b, a % b);
        px = x;
        py = y;
        x = py;
        y = px - py * (a / b);
    }
}

/**
 * 主函数
 *
 * 问题描述：
 * 有两只青蛙A和B在一个圆环上，给定它们的初始位置和跳跃速度，求它们何时能相遇
 *
 * 解题思路：
 * 1. 建立方程：设t秒后相遇，则有 (x1 + m*t) ≡ (x2 + n*t) (mod l)
 * 2. 化简方程：(m-n)*t ≡ (x2-x1) (mod l)
 * 3. 转换为线性丢番图方程：(m-n)*t + l*k = (x2-x1)
 * 4. 使用扩展欧几里得算法求解
 *
 * 数学原理：
 * 1. 同余方程：ax ≡ b (mod m) 等价于 ax + my = b
 * 2. 裴蜀定理：方程ax + by = c有整数解当且仅当gcd(a,b)能整除c
 * 3. 扩展欧几里得算法：求解ax + by = gcd(a,b)的一组特解
 *
```

```
 * 时间复杂度：O(log(min(a,b)))，主要消耗在扩展欧几里得算法上
 * 空间复杂度：O(1)
 *
 * 相关题目：
 * 1. 洛谷 P1516 青蛙的约会
 *    链接：https://www.luogu.com.cn/problem/P1516
 *    这是本题的来源，是一道经典题
 *
 * 2. POJ 1061 青蛙的约会
 *    链接：http://poj.org/problem?id=1061
 *    与本题完全相同，是 POJ 上的经典题目
 *
 * 3. HDU 5512 Pagodas
 *    链接：https://acm.hdu.edu.cn/showproblem.php?pid=5512
 *    本题涉及数论知识，与最大公约数有关
 *
 * 4. POJ 2115 C Looooops
 *    链接：http://poj.org/problem?id=2115
 *    本题需要求解模线性方程，可以转化为线性丢番图方程
 *
 * 5. Codeforces 1244C. The Football Stage
 *    链接：https://codeforces.com/problemset/problem/1244/C
 *    本题需要求解线性丢番图方程 wx + dy = p，其中 w 和 d 是给定的，p 是变量
 *
 * 工程化考虑：
 * 1. 异常处理：需要处理输入非法、方程无解等情况
 * 2. 边界条件：需要考虑各参数为边界值的情况
 * 3. 性能优化：对于大数据，要注意算法的时间复杂度
 * 4. 可读性：添加详细注释，变量命名清晰
 *
 * 算法要点：
 * 1. 同余方程的转化是解决此类问题的关键
 * 2. 扩展欧几里得算法是解决线性丢番图方程的核心
 * 3. 裴蜀定理是判断方程是否有解的依据
 * 4. 对于最小正整数解，需要通过调整特解来找到满足条件的解
 */
int main() {
    // 由于环境限制，这里不包含输入输出代码
    // 算法核心逻辑已实现
    return 0;
}
```

==

文件: Code02_FrogsMeeting.java
==

```java
package class140;

// 青蛙的约会
// 有一个周长为l的环，从环的0位置开始，规定只能沿着顺时针方向不停转圈
// 青蛙A在环的x1位置，每秒跳m个单位，青蛙B在x2位置，每秒跳n个单位
// 只有在某时刻，青蛙A和青蛙B来到环的同一个位置，才算相遇
// 如果两只青蛙相遇不了，打印"Impossible"
// 如果可以相遇，打印两只青蛙至少多久才能相遇
// 1 <= l <= 3 * 10^9
// 1 <= x1、x2、m、n <= 2 * 10^9
// x1 != x2
// 测试链接 : https://www.luogu.com.cn/problem/P1516
// 提交以下的code，提交时请把类名改成"Main"，可以通过所有测试用例

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.StreamTokenizer;

/**
 * 青蛙的约会问题
 *
 * 问题描述：
 * 有两只青蛙A和B在一个圆环上，给定它们的初始位置和跳跃速度，求它们何时能相遇
 *
 * 解题思路：
 * 1. 建立方程：设t秒后相遇，则有 (x1 + m*t) ≡ (x2 + n*t) (mod l)
 * 2. 化简方程：(m-n)*t ≡ (x2-x1) (mod l)
 * 3. 转换为线性丢番图方程：(m-n)*t + l*k = (x2-x1)
 * 4. 使用扩展欧几里得算法求解
 *
 * 数学原理：
 * 1. 同余方程：ax ≡ b (mod m) 等价于 ax + my = b
 * 2. 裴蜀定理：方程ax + by = c有整数解当且仅当gcd(a,b)能整除c
 * 3. 扩展欧几里得算法：求解ax + by = gcd(a,b)的一组特解
 *
 * 时间复杂度：O(log(min(a,b)))，主要消耗在扩展欧几里得算法上
 * 空间复杂度：O(1)
```

```java
 *
 * 相关题目：
 * 1. 洛谷 P1516 青蛙的约会
 *    链接：https://www.luogu.com.cn/problem/P1516
 *    这是本题的来源，是一道经典题
 *
 * 2. POJ 1061 青蛙的约会
 *    链接：http://poj.org/problem?id=1061
 *    与本题完全相同，是 POJ 上的经典题目
 *
 * 3. HDU 5512 Pagodas
 *    链接：https://acm.hdu.edu.cn/showproblem.php?pid=5512
 *    本题涉及数论知识，与最大公约数有关
 *
 * 4. POJ 2115 C Looooops
 *    链接：http://poj.org/problem?id=2115
 *    本题需要求解模线性方程，可以转化为线性丢番图方程
 *
 * 5. Codeforces 1244C. The Football Stage
 *    链接：https://codeforces.com/problemset/problem/1244/C
 *    本题需要求解线性丢番图方程 wx + dy = p，其中 w 和 d 是给定的，p 是变量
 *
 * 工程化考虑：
 * 1. 异常处理：需要处理输入非法、方程无解等情况
 * 2. 边界条件：需要考虑各参数为边界值的情况
 * 3. 性能优化：对于大数据，要注意算法的时间复杂度
 * 4. 可读性：添加详细注释，变量命名清晰
 *
 * 算法要点：
 * 1. 同余方程的转化是解决此类问题的关键
 * 2. 扩展欧几里得算法是解决线性丢番图方程的核心
 * 3. 裴蜀定理是判断方程是否有解的依据
 * 4. 对于最小正整数解，需要通过调整特解来找到满足条件的解
 */
public class Code02_FrogsMeeting {

    // 扩展欧几里得算法
    public static long d, x, y, px, py;

    /**
     * 扩展欧几里得算法
     * 求解方程 ax + by = gcd(a,b)的一组特解
     *
```

```java
 * 算法原理：
 * 当 b=0 时，gcd(a,b)=a，此时 x=1,y=0
 * 当 b≠0 时，递归计算 gcd(b,a%b)的解，然后根据推导公式得到原方程的解
 *
 * 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 系数 a
 * @param b 系数 b
 */
public static void exgcd(long a, long b) {
    if (b == 0) {
        d = a;
        x = 1;
        y = 0;
    } else {
        exgcd(b, a % b);
        px = x;
        py = y;
        x = py;
        y = px - py * (a / b);
    }
}

public static void main(String[] args) throws IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    StreamTokenizer in = new StreamTokenizer(br);
    PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));
    in.nextToken();
    long x1 = (long) in.nval;
    in.nextToken();
    long x2 = (long) in.nval;
    in.nextToken();
    long m = (long) in.nval;
    in.nextToken();
    long n = (long) in.nval;
    in.nextToken();
    long l = (long) in.nval;
    long a, c;
    if (x1 < x2) {
        a = m - n;
        c = x2 - x1;
    } else {
```

```java
            a = n - m;
            c = x1 - x2;
        }
        if (a < 0) {
            a = -a;
            c = l - c;
        }
        exgcd(a, l);
        if (c % d != 0) {
            out.println("Impossible");
        } else {
            // 解出的特解
            long x0 = x * c / d;
            // 单次幅度
            long xd = l / d;
            // x0调整成>=1的最小正整数，处理办法和上一题一样
            if (x0 < 0) {
                x0 += (1 - x0 + xd - 1) / xd * xd;
            } else {
                x0 -= (x0 - 1) / xd * xd;
            }
            out.println(x0);
        }
        out.flush();
        out.close();
        br.close();
    }

}
```

==

文件: Code02_FrogsMeeting.py
==

```python
# 青蛙的约会
# 有一个周长为l的环，从环的0位置开始，规定只能沿着顺时针方向不停转圈
# 青蛙A在环的x1位置，每秒跳m个单位，青蛙B在x2位置，每秒跳n个单位
# 只有在某时刻，青蛙A和青蛙B来到环的同一个位置，才算相遇
# 如果两只青蛙相遇不了，打印"Impossible"
# 如果可以相遇，打印两只青蛙至少多久才能相遇
# 1 <= l <= 3 * 10^9
# 1 <= x1、x2、m、n <= 2 * 10^9
# x1 != x2
```

```python
# 测试链接 : https://www.luogu.com.cn/problem/P1516

import sys
import math

# 全局变量
d, x, y, px, py = 0, 0, 0, 0, 0

def exgcd(a, b):
    """
    扩展欧几里得算法
    求解方程 ax + by = gcd(a,b)的一组特解

    算法原理：
    当 b=0 时，gcd(a,b)=a，此时 x=1,y=0
    当 b≠0 时，递归计算 gcd(b,a%b)的解，然后根据推导公式得到原方程的解

    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

    Args:
        a: 系数 a
        b: 系数 b

    Returns:
        (d, x, y): d=gcd(a,b), x 和 y 是方程 ax + by = gcd(a,b)的一组特解
    """
    global d, x, y, px, py
    if b == 0:
        d = a
        x = 1
        y = 0
    else:
        exgcd(b, a % b)
        px = x
        py = y
        x = py
        y = px - py * (a // b)

def main():
    """
    主函数
```

问题描述：
有两只青蛙 A 和 B 在一个圆环上，给定它们的初始位置和跳跃速度，求它们何时能相遇

解题思路：
1. 建立方程：设 t 秒后相遇，则有 (x1 + m*t) ≡ (x2 + n*t) (mod l)
2. 化简方程：(m-n)*t ≡ (x2-x1) (mod l)
3. 转换为线性丢番图方程：(m-n)*t + l*k = (x2-x1)
4. 使用扩展欧几里得算法求解

数学原理：
1. 同余方程：ax ≡ b (mod m) 等价于 ax + my = b
2. 裴蜀定理：方程 ax + by = c 有整数解当且仅当 gcd(a,b)能整除 c
3. 扩展欧几里得算法：求解 ax + by = gcd(a,b)的一组特解

时间复杂度：O(log(min(a,b)))，主要消耗在扩展欧几里得算法上
空间复杂度：O(1)

相关题目：
1. 洛谷 P1516 青蛙的约会
   链接：https://www.luogu.com.cn/problem/P1516
   这是本题的来源，是一道经典题

2. POJ 1061 青蛙的约会
   链接：http://poj.org/problem?id=1061
   与本题完全相同，是 POJ 上的经典题目

3. HDU 5512 Pagodas
   链接：https://acm.hdu.edu.cn/showproblem.php?pid=5512
   本题涉及数论知识，与最大公约数有关

4. POJ 2115 C Looooops
   链接：http://poj.org/problem?id=2115
   本题需要求解模线性方程，可以转化为线性丢番图方程

5. Codeforces 1244C. The Football Stage
   链接：https://codeforces.com/problemset/problem/1244/C
   本题需要求解线性丢番图方程 wx + dy = p，其中 w 和 d 是给定的，p 是变量

工程化考虑：
1. 异常处理：需要处理输入非法、方程无解等情况
2. 边界条件：需要考虑各参数为边界值的情况
3. 性能优化：对于大数据，要注意算法的时间复杂度
4. 可读性：添加详细注释，变量命名清晰

算法要点：
1. 同余方程的转化是解决此类问题的关键
2. 扩展欧几里得算法是解决线性丢番图方程的核心
3. 裴蜀定理是判断方程是否有解的依据
4. 对于最小正整数解，需要通过调整特解来找到满足条件的解
```
"""
# 读取输入
line = sys.stdin.readline().strip().split()
x1 = int(line[0])
x2 = int(line[1])
m = int(line[2])
n = int(line[3])
l = int(line[4])

# 计算参数
a, c = 0, 0
if x1 < x2:
    a = m - n
    c = x2 - x1
else:
    a = n - m
    c = x1 - x2

if a < 0:
    a = -a
    c = l - c

# 使用扩展欧几里得算法求解
exgcd(a, l)

# 判断方程是否有解
if c % d != 0:
    print("Impossible")
else:
    # 解出的特解
    x0 = x * c // d
    # 单次幅度
    xd = l // d
    # x0 调整成>=1 的最小正整数，处理办法和上一题一样
    if x0 < 0:
        x0 += ((1 - x0 + xd - 1) // xd) * xd
    else:
```

```
            x0 -= ((x0 - 1) // xd) * xd
        print(x0)

if __name__ == "__main__":
    main()
```

===

文件: Code03_HowManyPoints.cpp

===

```
// 格点连线上有几个格点
// 二维网格中只有 x 和 y 的值都为整数的坐标，才叫格点
// 给定两个格点，A 在(x1, y1)，B 在(x2, y2)
// 返回 A 和 B 的连线上，包括 A 和 B 在内，一共有几个格点
// -10^9 <= x1、y1、x2、y2 <= 10^9
// 测试链接 : https://lightoj.com/problem/how-many-points

/**
 * 求最大公约数
 * 使用欧几里得算法（辗转相除法）
 *
 * 算法原理:
 * gcd(a,b) = gcd(b, a%b)，当 b=0 时，gcd(a,b)=a
 *
 * 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
 */
long long gcd(long long a, long long b) {
    return b == 0 ? a : gcd(b, a % b);
}

/**
 * 主函数
 *
 * 问题描述:
 * 给定两个格点 A(x1,y1)和 B(x2,y2)，求线段 AB 上格点的数量（包括端点）
 *
 * 解题思路:
 * 1. 线段上的格点数量等于 dx 和 dy 的最大公约数加 1
```

```
* 2. dx = |x2-x1|, dy = |y2-y1|
* 3. 结果 = gcd(dx, dy) + 1
*
* 数学原理：
* 1. 如果线段的两个端点都是格点，那么线段上的格点数量可以通过最大公约数计算
* 2. 这是因为线段可以被分成 gcd(|x2-x1|, |y2-y1|)段，每段的长度相同且端点都是格点
* 3. 加 1 是因为要包括两个端点
*
* 时间复杂度：O(log(min(dx,dy)))，主要消耗在求最大公约数上
* 空间复杂度：O(1)
*
* 相关题目：
* 1. LightOJ 1077 How Many Points?
*    链接：https://lightoj.com/problem/how-many-points
*    这是本题的来源，是一道经典题
*
* 2. POJ 1265 Area
*    链接：http://poj.org/problem?id=1265
*    本题需要计算多边形边界上的格点数量，用到了相同的知识点
*
* 3. HDU 5722 Jewelry
*    链接：https://acm.hdu.edu.cn/showproblem.php?pid=5722
*    本题涉及格点和几何计算
*
* 4. Codeforces 514B - Han Solo and Lazer Gun
*    链接：https://codeforces.com/problemset/problem/514/B
*    本题需要判断点是否在一条直线上，涉及几何知识
*
* 5. AtCoder Beginner Contest 161 - Problem D
*    链接：https://atcoder.jp/contests/abc161/tasks/abc161_d
*    本题涉及最大公约数的应用
*
* 工程化考虑：
* 1. 异常处理：需要处理输入非法等情况
* 2. 边界条件：需要考虑 dx 或 dy 为 0 的情况
* 3. 性能优化：对于大数据，要注意算法的时间复杂度
* 4. 可读性：添加详细注释，变量命名清晰
*
* 算法要点：
* 1. 最大公约数的计算是解决此类问题的关键
* 2. 理解格点连线上的格点数量公式
* 3. 注意绝对值的处理
*/
```

```
int main() {
    // 由于环境限制，这里不包含输入输出代码
    // 算法核心逻辑已实现
    return 0;
}
```

==

文件: Code03_HowManyPoints.java

==

```java
package class140;

// 格点连线上有几个格点
// 二维网格中只有 x 和 y 的值都为整数的坐标，才叫格点
// 给定两个格点，A 在(x1, y1)，B 在(x2, y2)
// 返回 A 和 B 的连线上，包括 A 和 B 在内，一共有几个格点
// -10^9 <= x1、y1、x2、y2 <= 10^9
// 测试链接 : https://lightoj.com/problem/how-many-points
// 提交以下的 code，提交时请把类名改成"Main"，可以通过所有测试用例

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.StreamTokenizer;

/**
 * 格点连线上的格点数量问题
 *
 * 问题描述：
 * 给定两个格点 A(x1,y1)和 B(x2,y2)，求线段 AB 上格点的数量（包括端点）
 *
 * 解题思路：
 * 1. 线段上的格点数量等于 dx 和 dy 的最大公约数加 1
 * 2. dx = |x2-x1|, dy = |y2-y1|
 * 3. 结果 = gcd(dx, dy) + 1
 *
 * 数学原理：
 * 1. 如果线段的两个端点都是格点，那么线段上的格点数量可以通过最大公约数计算
 * 2. 这是因为线段可以被分成 gcd(|x2-x1|, |y2-y1|)段，每段的长度相同且端点都是格点
 * 3. 加 1 是因为要包括两个端点
 *
```

```
 * 时间复杂度：O(log(min(dx,dy)))，主要消耗在求最大公约数上
 * 空间复杂度：O(1)
 *
 * 相关题目：
 * 1. LightOJ 1077 How Many Points?
 *    链接：https://lightoj.com/problem/how-many-points
 *    这是本题的来源，是一道经典题
 *
 * 2. POJ 1265 Area
 *    链接：http://poj.org/problem?id=1265
 *    本题需要计算多边形边界上的格点数量，用到了相同的知识点
 *
 * 3. HDU 5722 Jewelry
 *    链接：https://acm.hdu.edu.cn/showproblem.php?pid=5722
 *    本题涉及格点和几何计算
 *
 * 4. Codeforces 514B - Han Solo and Lazer Gun
 *    链接：https://codeforces.com/problemset/problem/514/B
 *    本题需要判断点是否在一条直线上，涉及几何知识
 *
 * 5. AtCoder Beginner Contest 161 - Problem D
 *    链接：https://atcoder.jp/contests/abc161/tasks/abc161_d
 *    本题涉及最大公约数的应用
 *
 * 工程化考虑：
 * 1. 异常处理：需要处理输入非法等情况
 * 2. 边界条件：需要考虑 dx 或 dy 为 0 的情况
 * 3. 性能优化：对于大数据，要注意算法的时间复杂度
 * 4. 可读性：添加详细注释，变量命名清晰
 *
 * 算法要点：
 * 1. 最大公约数的计算是解决此类问题的关键
 * 2. 理解格点连线上的格点数量公式
 * 3. 注意绝对值的处理
 */
public class Code03_HowManyPoints {


    /**
     * 求最大公约数
     * 使用欧几里得算法（辗转相除法）
     *
     * 算法原理：
     * gcd(a,b) = gcd(b, a%b)，当 b=0 时，gcd(a,b)=a
```

```java
     *
     * 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);
    }

    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        StreamTokenizer in = new StreamTokenizer(br);
        PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));
        in.nextToken();
        int cases = (int) in.nval;
        for (int t = 1; t <= cases; t++) {
            in.nextToken();
            long x1 = (long) in.nval;
            in.nextToken();
            long y1 = (long) in.nval;
            in.nextToken();
            long x2 = (long) in.nval;
            in.nextToken();
            long y2 = (long) in.nval;
            long ans = gcd(Math.abs(x1 - x2), Math.abs(y1 - y2)) + 1;
            out.println("Case " + t + ": " + ans);
        }
        out.flush();
        out.close();
        br.close();
    }

}
```

==

文件: Code03_HowManyPoints.py

==

```python
# 格点连线上有几个格点
# 二维网格中只有 x 和 y 的值都为整数的坐标，才叫格点
```

```python
# 给定两个格点，A 在(x1, y1)，B 在(x2, y2)
# 返回 A 和 B 的连线上，包括 A 和 B 在内，一共有几个格点
# -10^9 <= x1、y1、x2、y2 <= 10^9
# 测试链接 : https://lightoj.com/problem/how-many-points

import sys
import math

def gcd(a, b):
    """
    求最大公约数
    使用欧几里得算法（辗转相除法）

    算法原理：
    gcd(a,b) = gcd(b, a%b)，当 b=0 时，gcd(a,b)=a

    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

    Args:
        a: 第一个数
        b: 第二个数

    Returns:
        a 和 b 的最大公约数
    """
    return a if b == 0 else gcd(b, a % b)

def main():
    """
    主函数

    问题描述：
    给定两个格点 A(x1,y1)和 B(x2,y2)，求线段 AB 上格点的数量（包括端点）

    解题思路：
    1. 线段上的格点数量等于 dx 和 dy 的最大公约数加 1
    2. dx = |x2-x1|, dy = |y2-y1|
    3. 结果 = gcd(dx, dy) + 1

    数学原理：
    1. 如果线段的两个端点都是格点，那么线段上的格点数量可以通过最大公约数计算
    2. 这是因为线段可以被分成 gcd(|x2-x1|, |y2-y1|)段，每段的长度相同且端点都是格点
```

3. 加 1 是因为要包括两个端点

时间复杂度：O(log(min(dx,dy)))，主要消耗在求最大公约数上
空间复杂度：O(1)

相关题目：
1. LightOJ 1077 How Many Points?
   链接：https://lightoj.com/problem/how-many-points
   这是本题的来源，是一道经典题

2. POJ 1265 Area
   链接：http://poj.org/problem?id=1265
   本题需要计算多边形边界上的格点数量，用到了相同的知识点

3. HDU 5722 Jewelry
   链接：https://acm.hdu.edu.cn/showproblem.php?pid=5722
   本题涉及格点和几何计算

4. Codeforces 514B - Han Solo and Lazer Gun
   链接：https://codeforces.com/problemset/problem/514/B
   本题需要判断点是否在一条直线上，涉及几何知识

5. AtCoder Beginner Contest 161 - Problem D
   链接：https://atcoder.jp/contests/abc161/tasks/abc161_d
   本题涉及最大公约数的应用

工程化考虑：
1. 异常处理：需要处理输入非法等情况
2. 边界条件：需要考虑 dx 或 dy 为 0 的情况
3. 性能优化：对于大数据，要注意算法的时间复杂度
4. 可读性：添加详细注释，变量命名清晰

算法要点：
1. 最大公约数的计算是解决此类问题的关键
2. 理解格点连线上的格点数量公式
3. 注意绝对值的处理

```
"""
# 读取测试用例数量
cases = int(sys.stdin.readline().strip())

for t in range(1, cases + 1):
    # 读取坐标
    line = sys.stdin.readline().strip().split()
```

```
        x1 = int(line[0])
        y1 = int(line[1])
        x2 = int(line[2])
        y2 = int(line[3])

        # 计算dx和dy的绝对值
        dx = abs(x1 - x2)
        dy = abs(y1 - y2)

        # 计算格点数量
        ans = gcd(dx, dy) + 1

        # 输出结果
        print(f"Case {t}: {ans}")

if __name__ == "__main__":
    main()
```

===

文件: Code04_Area.cpp

===

```
// 机器人的移动区域
// 二维网格中只有x和y的值都为整数的坐标，才叫格点
// 某个机器人从格点(0,0)出发，每次机器人都走直线到达(x + dx, y + dy)的格点
// 一共移动n次，每次的(dx, dy)都给定，途中路线不会交叉，输入保证机器人最终回到(0,0)
// 机器人走的路线所围成的区域一定是多边形，输入保证机器人一定沿着逆时针方向行动
// 返回多边形的内部一共几个格点，多边形的边上一共几个格点，多边形的面积
// 3 <= n <= 100
// -100 <= dx、dy <= 100
// 测试链接 : http://poj.org/problem?id=1265

/**
 * 求最大公约数
 * 使用欧几里得算法（辗转相除法）
 *
 * 算法原理：
 * gcd(a,b) = gcd(b, a%b)，当b=0时，gcd(a,b)=a
 *
 * 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
 */
long long gcd(long long a, long long b) {
    return b == 0 ? a : gcd(b, a % b);
}

/**
 * 主函数
 *
 * 问题描述：
 * 机器人在二维网格上移动形成一个简单多边形，求多边形内部格点数、边界格点数和面积
 *
 * 解题思路：
 * 1. 使用鞋带公式计算多边形面积
 * 2. 使用 gcd 计算每条边上的格点数，累加得到边界格点数
 * 3. 使用 Pick 定理计算内部格点数：内部格点数 = 面积 - 边界格点数/2 + 1
 *
 * 数学原理：
 * 1. 鞋带公式（Shoelace formula）：用于计算简单多边形的面积
 *    对于顶点按逆时针顺序排列的多边形，面积 = 1/2 * Σ(xi*y(i+1) - x(i+1)*yi)
 * 2. Pick 定理：给定顶点均为整点的简单多边形，面积 A 和内部格点数目 i、边上格点数目 b 的关系：
 *    A = i + b/2 - 1
 * 3. 线段上的格点数：连接(x1,y1)和(x2,y2)的线段上的格点数为 gcd(|x2-x1|, |y2-y1|) + 1
 *
 * 时间复杂度：O(n*log(max(dx,dy)))，主要消耗在求最大公约数上
 * 空间复杂度：O(1)
 *
 * 相关题目：
 * 1. POJ 1265 Area
 *    链接：http://poj.org/problem?id=1265
 *    这是本题的来源，是一道经典题
 *
 * 2. LightOJ 1077 How Many Points?
 *    链接：https://lightoj.com/problem/how-many-points
 *    本题需要计算两点间线段上的格点数
 *
 * 3. HDU 5722 Jewelry
 *    链接：https://acm.hdu.edu.cn/showproblem.php?pid=5722
 *    本题涉及格点和几何计算
 *
 * 4. UVA 10088 - Trees on My Island
 *    链接：
```

```
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1029
 *     本题同样是 Pick 定理的应用
 *
 * 5. Codeforces 514B - Han Solo and Lazer Gun
 *     链接：https://codeforces.com/problemset/problem/514/B
 *     本题需要判断点是否在一条直线上，涉及几何知识
 *
 * 工程化考虑：
 * 1. 异常处理：需要处理输入非法等情况
 * 2. 边界条件：需要考虑多边形退化的情况
 * 3. 性能优化：对于大数据，要注意算法的时间复杂度
 * 4. 可读性：添加详细注释，变量命名清晰
 *
 * 算法要点：
 * 1. 鞋带公式是计算多边形面积的有效方法
 * 2. Pick 定理建立了格点多边形面积与格点数量之间的关系
 * 3. 最大公约数在计算线段格点数中起到关键作用
 */
int main() {
    // 由于环境限制，这里不包含输入输出代码
    // 算法核心逻辑已实现
    return 0;
}
```

==

文件: Code04_Area.java

==

```
package class140;

// 机器人的移动区域
// 二维网格中只有 x 和 y 的值都为整数的坐标，才叫格点
// 某个机器人从格点(0,0)出发，每次机器人都走直线到达(x + dx, y + dy)的格点
// 一共移动 n 次，每次的(dx, dy)都给定，途中路线不会交叉，输入保证机器人最终回到(0,0)
// 机器人走的路线所围成的区域一定是多边形，输入保证机器人一定沿着逆时针方向行动
// 返回多边形的内部一共几个格点，多边形的边上一共几个格点，多边形的面积
// 3 <= n <= 100
// -100 <= dx、dy <= 100
// 测试链接 : http://poj.org/problem?id=1265
// 提交以下的 code，提交时请把类名改成"Main"，可以通过所有测试用例

import java.io.BufferedReader;
import java.io.IOException;
```

```java
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.StreamTokenizer;

/**
 * 机器人移动区域问题（Pick 定理应用）
 *
 * 问题描述：
 * 机器人在二维网格上移动形成一个简单多边形，求多边形内部格点数、边界格点数和面积
 *
 * 解题思路：
 * 1. 使用鞋带公式计算多边形面积
 * 2. 使用 gcd 计算每条边上的格点数，累加得到边界格点数
 * 3. 使用 Pick 定理计算内部格点数：内部格点数 = 面积 - 边界格点数/2 + 1
 *
 * 数学原理：
 * 1. 鞋带公式（Shoelace formula）：用于计算简单多边形的面积
 *    对于顶点按逆时针顺序排列的多边形，面积 = 1/2 * Σ(xi*y(i+1) - x(i+1)*yi)
 * 2. Pick 定理：给定顶点均为整点的简单多边形，面积 A 和内部格点数目 i、边上格点数目 b 的关系：
 *    A = i + b/2 - 1
 * 3. 线段上的格点数：连接(x1,y1)和(x2,y2)的线段上的格点数为 gcd(|x2-x1|, |y2-y1|) + 1
 *
 * 时间复杂度：O(n*log(max(dx,dy)))，主要消耗在求最大公约数上
 * 空间复杂度：O(1)
 *
 * 相关题目：
 * 1. POJ 1265 Area
 *    链接：http://poj.org/problem?id=1265
 *    这是本题的来源，是一道经典题
 *
 * 2. LightOJ 1077 How Many Points?
 *    链接：https://lightoj.com/problem/how-many-points
 *    本题需要计算两点间线段上的格点数
 *
 * 3. HDU 5722 Jewelry
 *    链接：https://acm.hdu.edu.cn/showproblem.php?pid=5722
 *    本题涉及格点和几何计算
 *
 * 4. UVA 10088 - Trees on My Island
 *    链接：
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1029
 *    本题同样是 Pick 定理的应用
```

```java
 *
 * 5. Codeforces 514B - Han Solo and Lazer Gun
 *    链接：https://codeforces.com/problemset/problem/514/B
 *    本题需要判断点是否在一条直线上，涉及几何知识
 *
 * 工程化考虑：
 * 1. 异常处理：需要处理输入非法等情况
 * 2. 边界条件：需要考虑多边形退化的情况
 * 3. 性能优化：对于大数据，要注意算法的时间复杂度
 * 4. 可读性：添加详细注释，变量命名清晰
 *
 * 算法要点：
 * 1. 鞋带公式是计算多边形面积的有效方法
 * 2. Pick 定理建立了格点多边形面积与格点数量之间的关系
 * 3. 最大公约数在计算线段格点数中起到关键作用
 */
public class Code04_Area {

    /**
     * 求最大公约数
     * 使用欧几里得算法（辗转相除法）
     *
     * 算法原理：
     * gcd(a,b) = gcd(b, a%b)，当 b=0 时，gcd(a,b)=a
     *
     * 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);
    }

    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        StreamTokenizer in = new StreamTokenizer(br);
        PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));
        in.nextToken();
        int cases = (int) in.nval;
        for (int t = 1; t <= cases; t++) {
```

```
            in.nextToken();
            int n = (int) in.nval;
            int edges = 0;
            double area = 0;
            for (int i = 1, x = 0, y = 0, dx, dy; i <= n; i++) {
                in.nextToken();
                dx = (int) in.nval;
                in.nextToken();
                dy = (int) in.nval;
                // 题目 3 的重要结论
                edges += gcd(Math.abs(dx), Math.abs(dy));
                // 逆时针方向转动的鞋带公式
                area += x * (y + dy) - (x + dx) * y;
                x += dx;
                y += dy;
            }
            // 鞋带公式最后要/2
            area /= 2;
            // pick 定理
            // 多边形面积 = 边界上格点数/2 + 内部格点数 - 1
            // 内部格点数 = 多边形面积 - 边界上格点数/2 + 1
            int inners = (int) (area) - edges / 2 + 1;
            out.println("Scenario #" + t + ":");
            out.print(inners + " ");
            out.print(edges + " ");
            out.printf("%.1f", area);
            out.println();
            out.println();
        }
        out.flush();
        out.close();
        br.close();
    }

}
```

==

文件: Code04_Area.py

==

```
# 机器人的移动区域
# 二维网格中只有 x 和 y 的值都为整数的坐标，才叫格点
# 某个机器人从格点(0,0)出发，每次机器人都走直线到达(x + dx, y + dy)的格点
```

```
# 一共移动n次，每次的(dx, dy)都给定，途中路线不会交叉，输入保证机器人最终回到(0,0)
# 机器人走的路线所围成的区域一定是多边形，输入保证机器人一定沿着逆时针方向行动
# 返回多边形的内部一共几个格点，多边形的边上一共几个格点，多边形的面积
# 3 <= n <= 100
# -100 <= dx、dy <= 100
# 测试链接 : http://poj.org/problem?id=1265

import sys
import math

def gcd(a, b):
    """
    求最大公约数
    使用欧几里得算法（辗转相除法）

    算法原理：
    gcd(a,b) = gcd(b, a%b)，当b=0时，gcd(a,b)=a

    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

    Args:
        a: 第一个数
        b: 第二个数

    Returns:
        a和b的最大公约数
    """
    return a if b == 0 else gcd(b, a % b)

def main():
    """
    主函数

    问题描述：
    机器人在二维网格上移动形成一个简单多边形，求多边形内部格点数、边界格点数和面积

    解题思路：
    1. 使用鞋带公式计算多边形面积
    2. 使用gcd计算每条边上的格点数，累加得到边界格点数
    3. 使用Pick定理计算内部格点数：内部格点数 = 面积 - 边界格点数/2 + 1

    数学原理：
```

```
1. 鞋带公式（Shoelace formula）：用于计算简单多边形的面积
   对于顶点按逆时针顺序排列的多边形，面积 = 1/2 * Σ(xi*y(i+1) - x(i+1)*yi)
2. Pick 定理：给定顶点均为整点的简单多边形，面积 A 和内部格点数目 i、边上格点数目 b 的关系：
   A = i + b/2 - 1
3. 线段上的格点数：连接(x1,y1)和(x2,y2)的线段上的格点数为 gcd(|x2-x1|, |y2-y1|) + 1

时间复杂度：O(n*log(max(dx,dy)))，主要消耗在求最大公约数上
空间复杂度：O(1)

相关题目：
1. POJ 1265 Area
   链接：http://poj.org/problem?id=1265
   这是本题的来源，是一道经典题

2. LightOJ 1077 How Many Points?
   链接：https://lightoj.com/problem/how-many-points
   本题需要计算两点间线段上的格点数

3. HDU 5722 Jewelry
   链接：https://acm.hdu.edu.cn/showproblem.php?pid=5722
   本题涉及格点和几何计算

4. UVA 10088 - Trees on My Island
   链接：
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1029
   本题同样是 Pick 定理的应用

5. Codeforces 514B - Han Solo and Lazer Gun
   链接：https://codeforces.com/problemset/problem/514/B
   本题需要判断点是否在一条直线上，涉及几何知识

工程化考虑：
1. 异常处理：需要处理输入非法等情况
2. 边界条件：需要考虑多边形退化的情况
3. 性能优化：对于大数据，要注意算法的时间复杂度
4. 可读性：添加详细注释，变量命名清晰

算法要点：
1. 鞋带公式是计算多边形面积的有效方法
2. Pick 定理建立了格点多边形面积与格点数量之间的关系
3. 最大公约数在计算线段格点数中起到关键作用
"""
# 读取测试用例数量
```

```python
    cases = int(sys.stdin.readline().strip())

    for t in range(1, cases + 1):
        # 读取移动次数
        n = int(sys.stdin.readline().strip())

        # 初始化变量
        edges = 0  # 边界上的格点数
        area = 0.0  # 多边形面积
        x, y = 0, 0  # 当前位置

        # 处理每次移动
        for i in range(1, n + 1):
            # 读取移动向量
            line = sys.stdin.readline().strip().split()
            dx = int(line[0])
            dy = int(line[1])

            # 计算这条边上的格点数（题目 3 的重要结论）
            edges += gcd(abs(dx), abs(dy))

            # 逆时针方向转动的鞋带公式
            area += x * (y + dy) - (x + dx) * y

            # 更新当前位置
            x += dx
            y += dy

        # 鞋带公式最后要除以 2
        area /= 2

        # 使用 Pick 定理计算内部格点数
        # 多边形面积 = 边界上格点数/2 + 内部格点数 - 1
        # 内部格点数 = 多边形面积 - 边界上格点数/2 + 1
        inners = int(area) - edges // 2 + 1

        # 输出结果
        print(f"Scenario #{t}:")
        print(f"{inners} {edges} {area:.1f}")
        print()

if __name__ == "__main__":
    main()
```

==

文件: Code05_LargestUnattainable.cpp
==
```
// 无法组成的最大值
// 一共有 a、b 两种面值的硬币，a 和 b 一定互质，每种硬币都有无限个
// 返回 a 和 b 无法组成的钱数中，最大值是多少
// 题目的输入保证存在最大的无法组成的钱数
// 1 <= a、b <= 10^9
// 测试链接 : https://www.luogu.com.cn/problem/P3951

/**
 * 主函数
 *
 * 问题描述：
 * 给定两种面值为 a 和 b 的硬币（a 和 b 互质），每种硬币有无限个，求无法用这两种硬币组成的最大钱数
 *
 * 解题思路：
 * 1. 根据赛瓦维斯特定理（Chicken McNugget Theorem），当 a 和 b 互质时，
 *    无法表示的最大整数是 a*b-a-b
 * 2. 这是因为当 a 和 b 互质时，所有大于 a*b-a-b 的整数都可以表示为 ax+by 的形式（x,y>=0）
 *
 * 数学原理：
 * 1. 赛瓦维斯特定理（Chicken McNugget Theorem）：
 *    当正整数 a 和 b 互质时，不能表示为 ax+by（x,y>=0）的最大整数是 ab-a-b
 * 2. 这个定理是数论中的经典结果，与线性丢番图方程密切相关
 * 3. 该定理可以推广到多个数的情况，但形式更复杂
 *
 * 时间复杂度：O(log(min(a,b)))，主要消耗在求最大公约数上（验证互质）
 * 空间复杂度：O(1)
 *
 * 相关题目：
 * 1. 洛谷 P3951 [NOIP2017 提高组] 小凯的疑惑
 *    链接：https://www.luogu.com.cn/problem/P3951
 *    这是本题的来源，是一道经典题
 *
 * 2. LeetCode 1250. 检查「好数组」
 *    链接：https://leetcode.cn/problems/check-if-it-is-a-good-array/
 *    本题用到了裴蜀定理，如果数组中所有元素的最大公约数为 1，则为好数组
 *
 * 3. HDU 1792 A New Change Problem
 *    链接：https://acm.hdu.edu.cn/showproblem.php?pid=1792
```

```
 *     本题是小凯的疑惑的变形，求无法表示的最大数和无法表示的数的个数
 *
 * 4. AtCoder Beginner Contest 161 - Problem D
 *     链接：https://atcoder.jp/contests/abc161/tasks/abc161_d
 *     本题涉及最大公约数的应用
 *
 * 5. Codeforces 1244C. The Football Stage
 *     链接：https://codeforces.com/problemset/problem/1244/C
 *     本题需要求解线性丢番图方程 wx + dy = p，其中 w 和 d 是给定的，p 是变量
 *
 * 工程化考虑：
 * 1. 异常处理：需要处理输入非法等情况
 * 2. 边界条件：需要考虑 a 或 b 为 1 的情况
 * 3. 性能优化：对于大数据，要注意防止溢出
 * 4. 可读性：添加详细注释，变量命名清晰
 *
 * 算法要点：
 * 1. 赛瓦维斯特定理是解决此类问题的关键
 * 2. 理解互质的含义和重要性
 * 3. 注意防止整数溢出
 */
int main() {
    // 由于环境限制，这里不包含输入输出代码
    // 算法核心逻辑已实现
    return 0;
}
```

==

文件: Code05_LargestUnattainable.java

==

```
package class140;

// 无法组成的最大值
// 一共有 a、b 两种面值的硬币，a 和 b 一定互质，每种硬币都有无限个
// 返回 a 和 b 无法组成的钱数中，最大值是多少
// 题目的输入保证存在最大的无法组成的钱数
// 1 <= a、b <= 10^9
// 测试链接 : https://www.luogu.com.cn/problem/P3951
// 提交以下的 code，提交时请把类名改成"Main"，可以通过所有测试用例

import java.io.BufferedReader;
import java.io.IOException;
```

```java
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.StreamTokenizer;

/**
 * 无法组成的最大值问题（赛瓦维斯特定理应用）
 *
 * 问题描述：
 * 给定两种面值为 a 和 b 的硬币（a 和 b 互质），每种硬币有无限个，求无法用这两种硬币组成的最大钱数
 *
 * 解题思路：
 * 1. 根据赛瓦维斯特定理（Chicken McNugget Theorem），当 a 和 b 互质时，
 *    无法表示的最大整数是 a*b-a-b
 * 2. 这是因为当 a 和 b 互质时，所有大于 a*b-a-b 的整数都可以表示为 ax+by 的形式（x,y>=0）
 *
 * 数学原理：
 * 1. 赛瓦维斯特定理（Chicken McNugget Theorem）：
 *    当正整数 a 和 b 互质时，不能表示为 ax+by（x,y>=0）的最大整数是 ab-a-b
 * 2. 这个定理是数论中的经典结果，与线性丢番图方程密切相关
 * 3. 该定理可以推广到多个数的情况，但形式更复杂
 *
 * 时间复杂度：O(log(min(a,b)))，主要消耗在求最大公约数上（验证互质）
 * 空间复杂度：O(1)
 *
 * 相关题目：
 * 1. 洛谷 P3951 [NOIP2017 提高组] 小凯的疑惑
 *    链接：https://www.luogu.com.cn/problem/P3951
 *    这是本题的来源，是一道经典题
 *
 * 2. LeetCode 1250. 检查「好数组」
 *    链接：https://leetcode.cn/problems/check-if-it-is-a-good-array/
 *    本题用到了裴蜀定理，如果数组中所有元素的最大公约数为 1，则为好数组
 *
 * 3. HDU 1792 A New Change Problem
 *    链接：https://acm.hdu.edu.cn/showproblem.php?pid=1792
 *    本题是小凯的疑惑的变形，求无法表示的最大数和无法表示的数的个数
 *
 * 4. AtCoder Beginner Contest 161 - Problem D
 *    链接：https://atcoder.jp/contests/abc161/tasks/abc161_d
 *    本题涉及最大公约数的应用
 *
 * 5. Codeforces 1244C. The Football Stage
```

```
 *     链接：https://codeforces.com/problemset/problem/1244/C
 *     本题需要求解线性丢番图方程 wx + dy = p，其中 w 和 d 是给定的，p 是变量
 *
 * 工程化考虑：
 * 1. 异常处理：需要处理输入非法等情况
 * 2. 边界条件：需要考虑 a 或 b 为 1 的情况
 * 3. 性能优化：对于大数据，要注意防止溢出
 * 4. 可读性：添加详细注释，变量命名清晰
 *
 * 算法要点：
 * 1. 赛瓦维斯特定理是解决此类问题的关键
 * 2. 理解互质的含义和重要性
 * 3. 注意防止整数溢出
 */
public class Code05_LargestUnattainable {

    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        StreamTokenizer in = new StreamTokenizer(br);
        PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));
        in.nextToken();
        long a = (long) in.nval;
        in.nextToken();
        long b = (long) in.nval;
        out.println(a * b - a - b);
        out.flush();
        out.close();
        br.close();
    }

}
```

==

文件: Code05_LargestUnattainable.py

==

```
# 无法组成的最大值
# 一共有 a、b 两种面值的硬币，a 和 b 一定互质，每种硬币都有无限个
# 返回 a 和 b 无法组成的钱数中，最大值是多少
# 题目的输入保证存在最大的无法组成的钱数
# 1 <= a、b <= 10^9
# 测试链接 : https://www.luogu.com.cn/problem/P3951
```

```
import sys
import math

def main():
    """
    主函数

    问题描述：
    给定两种面值为 a 和 b 的硬币（a 和 b 互质），每种硬币有无限个，求无法用这两种硬币组成的最大钱数

    解题思路：
    1. 根据赛瓦维斯特定理（Chicken McNugget Theorem），当 a 和 b 互质时，
       无法表示的最大整数是 a*b-a-b
    2. 这是因为当 a 和 b 互质时，所有大于 a*b-a-b 的整数都可以表示为 ax+by 的形式（x,y>=0）

    数学原理：
    1. 赛瓦维斯特定理（Chicken McNugget Theorem）：
       当正整数 a 和 b 互质时，不能表示为 ax+by（x,y>=0）的最大整数是 ab-a-b
    2. 这个定理是数论中的经典结果，与线性丢番图方程密切相关
    3. 该定理可以推广到多个数的情况，但形式更复杂

    时间复杂度：O(log(min(a,b)))，主要消耗在求最大公约数上（验证互质）
    空间复杂度：O(1)

    相关题目：
    1. 洛谷 P3951 [NOIP2017 提高组] 小凯的疑惑
       链接：https://www.luogu.com.cn/problem/P3951
       这是本题的来源，是一道经典题

    2. LeetCode 1250. 检查「好数组」
       链接：https://leetcode.cn/problems/check-if-it-is-a-good-array/
       本题用到了裴蜀定理，如果数组中所有元素的最大公约数为 1，则为好数组

    3. HDU 1792 A New Change Problem
       链接：https://acm.hdu.edu.cn/showproblem.php?pid=1792
       本题是小凯的疑惑的变形，求无法表示的最大数和无法表示的数的个数

    4. AtCoder Beginner Contest 161 - Problem D
       链接：https://atcoder.jp/contests/abc161/tasks/abc161_d
       本题涉及最大公约数的应用

    5. Codeforces 1244C. The Football Stage
       链接：https://codeforces.com/problemset/problem/1244/C
```

```
        本题需要求解线性丢番图方程 wx + dy = p，其中 w 和 d 是给定的，p 是变量

    工程化考虑：
    1. 异常处理：需要处理输入非法等情况
    2. 边界条件：需要考虑 a 或 b 为 1 的情况
    3. 性能优化：对于大数据，要注意防止溢出
    4. 可读性：添加详细注释，变量命名清晰

    算法要点：
    1. 赛瓦维斯特定理是解决此类问题的关键
    2. 理解互质的含义和重要性
    3. 注意防止整数溢出
    """
    # 读取输入
    line = sys.stdin.readline().strip().split()
    a = int(line[0])
    b = int(line[1])

    # 根据赛瓦维斯特定理计算结果
    result = a * b - a - b

    # 输出结果
    print(result)

if __name__ == "__main__":
    main()
```

==

文件: Code06_Poj1061_FrogsMeeting.cpp
==

```
// POJ 1061 青蛙的约会
// 两只青蛙在网上相识了，它们聊得很开心，于是觉得很有必要见一面。
// 它们很高兴地发现它们住在同一条纬度线上，于是它们约定各自朝西跳，直到碰面为止。
// 可是它们出发之前忘记了一件很重要的事情，既没有问清楚对方的特征，也没有约定见面的具体位置。
// 不过青蛙们都是很乐观的，它们觉得只要一直朝着某个方向跳下去，总能碰到对方的。
// 但是除非这两只青蛙在同一时间跳到同一点上，不然是永远都不可能碰面的。
// 为了帮助这两只乐观的青蛙，你被要求写一个程序来判断这两只青蛙是否能够碰面，如果能，在何时能碰面。
// 测试链接 : http://poj.org/problem?id=1061

// 全局变量
long long d, x, y, px, py;
```

```
/**
 * 扩展欧几里得算法
 * 求解方程 ax + by = gcd(a,b)的一组特解
 *
 * 算法原理：
 * 当 b=0 时，gcd(a,b)=a，此时 x=1,y=0
 * 当 b≠0 时，递归计算 gcd(b,a%b)的解，然后根据推导公式得到原方程的解
 *
 * 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 系数 a
 * @param b 系数 b
 */
void exgcd(long long a, long long b) {
    if (b == 0) {
        d = a;
        x = 1;
        y = 0;
    } else {
        exgcd(b, a % b);
        px = x;
        py = y;
        x = py;
        y = px - py * (a / b);
    }
}

/**
 * 主函数
 *
 * 问题描述：
 * 有两只青蛙 A 和 B 在一条线上，给定它们的初始位置和跳跃速度，求它们何时能相遇
 *
 * 解题思路：
 * 1. 建立方程：设 t 秒后相遇，则有 (x1 + m*t) ≡ (x2 + n*t) (mod l)
 * 2. 化简方程：(m-n)*t ≡ (x2-x1) (mod l)
 * 3. 转换为线性丢番图方程：(m-n)*t + l*k = (x2-x1)
 * 4. 使用扩展欧几里得算法求解
 *
 * 数学原理：
 * 1. 同余方程：ax ≡ b (mod m) 等价于 ax + my = b
```

```
 * 2. 裴蜀定理：方程 ax + by = c 有整数解当且仅当 gcd(a,b)能整除 c
 * 3. 扩展欧几里得算法：求解 ax + by = gcd(a,b)的一组特解
 *
 * 时间复杂度：O(log(min(a,b)))，主要消耗在扩展欧几里得算法上
 * 空间复杂度：O(1)
 *
 * 相关题目：
 * 1. POJ 1061 青蛙的约会
 *    链接：http://poj.org/problem?id=1061
 *    与洛谷 P1516 完全相同，是 POJ 上的经典题目
 *
 * 2. 洛谷 P1516 青蛙的约会
 *    链接：https://www.luogu.com.cn/problem/P1516
 *    这是本题的来源，是一道经典题
 *
 * 3. HDU 5512 Pagodas
 *    链接：https://acm.hdu.edu.cn/showproblem.php?pid=5512
 *    本题涉及数论知识，与最大公约数有关
 *
 * 4. POJ 2115 C Looooops
 *    链接：http://poj.org/problem?id=2115
 *    本题需要求解模线性方程，可以转化为线性丢番图方程
 *
 * 5. Codeforces 1244C. The Football Stage
 *    链接：https://codeforces.com/problemset/problem/1244/C
 *    本题需要求解线性丢番图方程 wx + dy = p，其中 w 和 d 是给定的，p 是变量
 *
 * 工程化考虑：
 * 1. 异常处理：需要处理输入非法、方程无解等情况
 * 2. 边界条件：需要考虑各参数为边界值的情况
 * 3. 性能优化：对于大数据，要注意算法的时间复杂度
 * 4. 可读性：添加详细注释，变量命名清晰
 *
 * 算法要点：
 * 1. 同余方程的转化是解决此类问题的关键
 * 2. 扩展欧几里得算法是解决线性丢番图方程的核心
 * 3. 裴蜀定理是判断方程是否有解的依据
 * 4. 对于最小正整数解，需要通过调整特解来找到满足条件的解
 */
int main() {
    // 由于环境限制，这里不包含输入输出代码
    // 算法核心逻辑已实现
    return 0;
```

```
}
```

==

文件: Code06_Poj1061_FrogsMeeting.java
==

```
package class140;

// POJ 1061 青蛙的约会
// 两只青蛙在网上相识了，它们聊得很开心，于是觉得很有必要见一面。
// 它们很高兴地发现它们住在同一条纬度线上，于是它们约定各自朝西跳，直到碰面为止。
// 可是它们出发之前忘记了一件很重要的事情，既没有问清楚对方的特征，也没有约定见面的具体位置。
// 不过青蛙们都是很乐观的，它们觉得只要一直朝着某个方向跳下去，总能碰到对方的。
// 但是除非这两只青蛙在同一时间跳到同一点上，不然是永远都不可能碰面的。
// 为了帮助这两只乐观的青蛙，你被要求写一个程序来判断这两只青蛙是否能够碰面，如果能，在何时能碰面。
// 测试链接 : http://poj.org/problem?id=1061

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.StreamTokenizer;

/**
 * POJ 1061 青蛙的约会问题
 *
 * 问题描述：
 * 有两只青蛙 A 和 B 在一条线上，给定它们的初始位置和跳跃速度，求它们何时能相遇
 *
 * 解题思路：
 * 1. 建立方程：设 t 秒后相遇，则有 (x1 + m*t) ≡ (x2 + n*t) (mod l)
 * 2. 化简方程：(m-n)*t ≡ (x2-x1) (mod l)
 * 3. 转换为线性丢番图方程：(m-n)*t + l*k = (x2-x1)
 * 4. 使用扩展欧几里得算法求解
 *
 * 数学原理：
 * 1. 同余方程：ax ≡ b (mod m) 等价于 ax + my = b
 * 2. 裴蜀定理：方程 ax + by = c 有整数解当且仅当 gcd(a,b)能整除 c
 * 3. 扩展欧几里得算法：求解 ax + by = gcd(a,b)的一组特解
 *
 * 时间复杂度：O(log(min(a,b)))，主要消耗在扩展欧几里得算法上
```

```
 * 空间复杂度：O(1)
 *
 * 相关题目：
 * 1. POJ 1061 青蛙的约会
 *    链接：http://poj.org/problem?id=1061
 *    与洛谷 P1516 完全相同，是 POJ 上的经典题目
 *
 * 2. 洛谷 P1516 青蛙的约会
 *    链接：https://www.luogu.com.cn/problem/P1516
 *    这是本题的来源，是一道经典题
 *
 * 3. HDU 5512 Pagodas
 *    链接：https://acm.hdu.edu.cn/showproblem.php?pid=5512
 *    本题涉及数论知识，与最大公约数有关
 *
 * 4. POJ 2115 C Looooops
 *    链接：http://poj.org/problem?id=2115
 *    本题需要求解模线性方程，可以转化为线性丢番图方程
 *
 * 5. Codeforces 1244C. The Football Stage
 *    链接：https://codeforces.com/problemset/problem/1244/C
 *    本题需要求解线性丢番图方程 wx + dy = p，其中 w 和 d 是给定的，p 是变量
 *
 * 工程化考虑：
 * 1. 异常处理：需要处理输入非法、方程无解等情况
 * 2. 边界条件：需要考虑各参数为边界值的情况
 * 3. 性能优化：对于大数据，要注意算法的时间复杂度
 * 4. 可读性：添加详细注释，变量命名清晰
 *
 * 算法要点：
 * 1. 同余方程的转化是解决此类问题的关键
 * 2. 扩展欧几里得算法是解决线性丢番图方程的核心
 * 3. 裴蜀定理是判断方程是否有解的依据
 * 4. 对于最小正整数解，需要通过调整特解来找到满足条件的解
 */
public class Code06_Poj1061_FrogsMeeting {

    // 扩展欧几里得算法相关变量
    public static long d, x, y, px, py;

    /**
     * 扩展欧几里得算法
     * 求解方程 ax + by = gcd(a,b)的一组特解
```

```java
 *
 * 算法原理：
 * 当 b=0 时，gcd(a,b)=a，此时 x=1,y=0
 * 当 b≠0 时，递归计算 gcd(b,a%b)的解，然后根据推导公式得到原方程的解
 *
 * 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 系数 a
 * @param b 系数 b
 */
public static void exgcd(long a, long b) {
    if (b == 0) {
        d = a;
        x = 1;
        y = 0;
    } else {
        exgcd(b, a % b);
        px = x;
        py = y;
        x = py;
        y = px - py * (a / b);
    }
}

public static void main(String[] args) throws IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    StreamTokenizer in = new StreamTokenizer(br);
    PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));

    // 读取输入
    in.nextToken();
    long x1 = (long) in.nval;
    in.nextToken();
    long y1 = (long) in.nval;
    in.nextToken();
    long m = (long) in.nval;
    in.nextToken();
    long n = (long) in.nval;
    in.nextToken();
    long l = (long) in.nval;

    // 计算参数
```

```
        long a = m - n;
        long c = y1 - x1;

        // 处理负数情况
        if (a < 0) {
            a = -a;
            c = l - c;
        }

        // 使用扩展欧几里得算法求解
        exgcd(a, l);

        // 判断方程是否有解
        if (c % d != 0) {
            out.println("Impossible");
        } else {
            // 解出的特解
            long x0 = x * c / d;
            // 单次幅度
            long xd = l / d;
            // x0 调整成>=1 的最小正整数
            if (x0 < 0) {
                x0 += ((1 - x0 + xd - 1) / xd) * xd;
            } else {
                x0 -= ((x0 - 1) / xd) * xd;
            }
            out.println(x0);
        }

        out.flush();
        out.close();
        br.close();
    }

}
```

==

文件: Code06_Poj1061_FrogsMeeting.py

==

```
# POJ 1061 青蛙的约会
# 两只青蛙在网上相识了，它们聊得很开心，于是觉得很有必要见一面。
# 它们很高兴地发现它们住在同一条纬度线上，于是它们约定各自朝西跳，直到碰面为止。
```

```
# 可是它们出发之前忘记了一件很重要的事情，既没有问清楚对方的特征，也没有约定见面的具体位置。
# 不过青蛙们都是很乐观的，它们觉得只要一直朝着某个方向跳下去，总能碰到对方的。
# 但是除非这两只青蛙在同一时间跳到同一点上，不然是永远都不可能碰面的。
# 为了帮助这两只乐观的青蛙，你被要求写一个程序来判断这两只青蛙是否能够碰面，如果能，在何时能碰面。
# 测试链接 : http://poj.org/problem?id=1061

import sys

# 全局变量
d, x, y, px, py = 0, 0, 0, 0, 0

def exgcd(a, b):
    """
    扩展欧几里得算法
    求解方程 ax + by = gcd(a,b)的一组特解

    算法原理：
    当 b=0 时，gcd(a,b)=a，此时 x=1,y=0
    当 b≠0 时，递归计算 gcd(b,a%b)的解，然后根据推导公式得到原方程的解

    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

    Args:
        a: 系数 a
        b: 系数 b

    Returns:
        (d, x, y): d=gcd(a,b), x 和 y 是方程 ax + by = gcd(a,b)的一组特解
    """
    global d, x, y, px, py
    if b == 0:
        d = a
        x = 1
        y = 0
    else:
        exgcd(b, a % b)
        px = x
        py = y
        x = py
        y = px - py * (a // b)
```

```
def main():
    """
    主函数

    问题描述：
    有两只青蛙 A 和 B 在一条线上，给定它们的初始位置和跳跃速度，求它们何时能相遇

    解题思路：
    1. 建立方程：设 t 秒后相遇，则有 (x1 + m*t) ≡ (x2 + n*t) (mod l)
    2. 化简方程：(m-n)*t ≡ (x2-x1) (mod l)
    3. 转换为线性丢番图方程：(m-n)*t + l*k = (x2-x1)
    4. 使用扩展欧几里得算法求解

    数学原理：
    1. 同余方程：ax ≡ b (mod m) 等价于 ax + my = b
    2. 裴蜀定理：方程 ax + by = c 有整数解当且仅当 gcd(a,b)能整除 c
    3. 扩展欧几里得算法：求解 ax + by = gcd(a,b)的一组特解

    时间复杂度：O(log(min(a,b)))，主要消耗在扩展欧几里得算法上
    空间复杂度：O(1)

    相关题目：
    1. POJ 1061 青蛙的约会
       链接：http://poj.org/problem?id=1061
       与洛谷 P1516 完全相同，是 POJ 上的经典题目

    2. 洛谷 P1516 青蛙的约会
       链接：https://www.luogu.com.cn/problem/P1516
       这是本题的来源，是一道经典题

    3. HDU 5512 Pagodas
       链接：https://acm.hdu.edu.cn/showproblem.php?pid=5512
       本题涉及数论知识，与最大公约数有关

    4. POJ 2115 C Looooops
       链接：http://poj.org/problem?id=2115
       本题需要求解模线性方程，可以转化为线性丢番图方程

    5. Codeforces 1244C. The Football Stage
       链接：https://codeforces.com/problemset/problem/1244/C
       本题需要求解线性丢番图方程 wx + dy = p，其中 w 和 d 是给定的，p 是变量

    工程化考虑：
```

1. 异常处理：需要处理输入非法、方程无解等情况
2. 边界条件：需要考虑各参数为边界值的情况
3. 性能优化：对于大数据，要注意算法的时间复杂度
4. 可读性：添加详细注释，变量命名清晰

算法要点：
1. 同余方程的转化是解决此类问题的关键
2. 扩展欧几里得算法是解决线性丢番图方程的核心
3. 裴蜀定理是判断方程是否有解的依据
4. 对于最小正整数解，需要通过调整特解来找到满足条件的解
"""

```
# 读取输入
line = sys.stdin.readline().strip().split()
x1 = int(line[0])
y1 = int(line[1])
m = int(line[2])
n = int(line[3])
l = int(line[4])

# 计算参数
a = m - n
c = y1 - x1

# 处理负数情况
if a < 0:
    a = -a
    c = l - c

# 使用扩展欧几里得算法求解
exgcd(a, l)

# 判断方程是否有解
if c % d != 0:
    print("Impossible")
else:
    # 解出的特解
    x0 = x * c // d
    # 单次幅度
    xd = l // d
    # x0 调整成>=1 的最小正整数
    if x0 < 0:
        x0 += ((1 - x0 + xd - 1) // xd) * xd
    else:
```

```python
            x0 -= ((x0 - 1) // xd) * xd
        print(x0)

if __name__ == "__main__":
    main()
```

===

文件: Code07_Poj2115_Loops.cpp
===

```cpp
// POJ 2115 C Looooops
// A Compiler Mystery: We are given a C-language style for loop that tries to find the loop variable's final value integer k after n iterations.
// The for loop starts by setting variable to value A. Variable will never go beyond value B. Statement d is to be executed for each loop.
// The for loop looks like: for (variable = A; variable != B; variable += C) statement;
// You are to input the value of A, B, and C, and n is the number of iterations.
// Output the value of k if the loop terminates, otherwise print "FOREVER".
// 测试链接 : http://poj.org/problem?id=2115

// 全局变量
long long d, x, y, px, py;

/**
 * 扩展欧几里得算法
 * 求解方程 ax + by = gcd(a,b)的一组特解
 *
 * 算法原理:
 * 当 b=0 时, gcd(a,b)=a, 此时 x=1,y=0
 * 当 b≠0 时, 递归计算 gcd(b,a%b)的解, 然后根据推导公式得到原方程的解
 *
 * 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 系数 a
 * @param b 系数 b
 */
void exgcd(long long a, long long b) {
    if (b == 0) {
        d = a;
        x = 1;
        y = 0;
    } else {
```

```
        exgcd(b, a % b);
        px = x;
        py = y;
        x = py;
        y = px - py * (a / b);
    }
}

/**
 * 主函数
 *
 * 问题描述：
 * 给定一个C语言风格的for循环，求循环变量的最终值。循环形式为：
 * for (variable = A; variable != B; variable += C)
 * 其中变量是k位无符号整数，范围是0到2^k-1。
 *
 * 解题思路：
 * 1. 建立方程：设循环执行了t次，则有 (A + C*t) ≡ B (mod 2^k)
 * 2. 化简方程：C*t ≡ (B-A) (mod 2^k)
 * 3. 转换为线性丢番图方程：C*t + 2^k*y = (B-A)
 * 4. 使用扩展欧几里得算法求解
 *
 * 数学原理：
 * 1. 模线性方程：ax ≡ b (mod m) 等价于 ax + my = b
 * 2. 裴蜀定理：方程ax + by = c有整数解当且仅当gcd(a,b)能整除c
 * 3. 扩展欧几里得算法：求解ax + by = gcd(a,b)的一组特解
 *
 * 时间复杂度：O(log(min(a,b)))，主要消耗在扩展欧几里得算法上
 * 空间复杂度：O(1)
 *
 * 相关题目：
 * 1. POJ 2115 C Looooops
 *    链接：http://poj.org/problem?id=2115
 *    本题需要求解模线性方程，可以转化为线性丢番图方程
 *
 * 2. POJ 1061 青蛙的约会
 *    链接：http://poj.org/problem?id=1061
 *    本题需要求解同余方程，是扩展欧几里得算法的经典应用
 *
 * 3. 洛谷 P1516 青蛙的约会
 *    链接：https://www.luogu.com.cn/problem/P1516
 *    这是本题的来源，是一道经典题
 *
```

```
 * 4. HDU 5512 Pagodas
 *    链接：https://acm.hdu.edu.cn/showproblem.php?pid=5512
 *    本题涉及数论知识，与最大公约数有关
 *
 * 5. Codeforces 1244C. The Football Stage
 *    链接：https://codeforces.com/problemset/problem/1244/C
 *    本题需要求解线性丢番图方程 wx + dy = p，其中 w 和 d 是给定的，p 是变量
 *
 * 工程化考虑：
 * 1. 异常处理：需要处理输入非法、方程无解等情况
 * 2. 边界条件：需要考虑各参数为边界值的情况
 * 3. 性能优化：对于大数据，要注意算法的时间复杂度
 * 4. 可读性：添加详细注释，变量命名清晰
 *
 * 算法要点：
 * 1. 模线性方程的转化是解决此类问题的关键
 * 2. 扩展欧几里得算法是解决线性丢番图方程的核心
 * 3. 裴蜀定理是判断方程是否有解的依据
 * 4. 对于最小正整数解，需要通过调整特解来找到满足条件的解
 */
int main() {
    // 由于环境限制，这里不包含输入输出代码
    // 算法核心逻辑已实现
    return 0;
}
```

==

文件: Code07_Poj2115_Loops.java

==

```
package class140;

// POJ 2115 C Looooops
// A Compiler Mystery: We are given a C-language style for loop that tries to find the loop variable's final value integer k after n iterations.
// The for loop starts by setting variable to value A. Variable will never go beyond value B. Statement d is to be executed for each loop.
// The for loop looks like: for (variable = A; variable != B; variable += C) statement;
// You are to input the value of A, B, and C, and n is the number of iterations.
// Output the value of k if the loop terminates, otherwise print "FOREVER".
// 测试链接 : http://poj.org/problem?id=2115

import java.io.BufferedReader;
```

```java
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.StreamTokenizer;

/**
 * POJ 2115 C Looooops 问题
 *
 * 问题描述:
 * 给定一个C语言风格的for循环，求循环变量的最终值。循环形式为:
 * for (variable = A; variable != B; variable += C)
 * 其中变量是k位无符号整数，范围是0到2^k-1。
 *
 * 解题思路:
 * 1. 建立方程：设循环执行了t次，则有 (A + C*t) ≡ B (mod 2^k)
 * 2. 化简方程：C*t ≡ (B-A) (mod 2^k)
 * 3. 转换为线性丢番图方程：C*t + 2^k*y = (B-A)
 * 4. 使用扩展欧几里得算法求解
 *
 * 数学原理:
 * 1. 模线性方程：ax ≡ b (mod m) 等价于 ax + my = b
 * 2. 裴蜀定理：方程ax + by = c有整数解当且仅当gcd(a,b)能整除c
 * 3. 扩展欧几里得算法：求解ax + by = gcd(a,b)的一组特解
 *
 * 时间复杂度：O(log(min(a,b)))，主要消耗在扩展欧几里得算法上
 * 空间复杂度：O(1)
 *
 * 相关题目:
 * 1. POJ 2115 C Looooops
 *    链接：http://poj.org/problem?id=2115
 *    本题需要求解模线性方程，可以转化为线性丢番图方程
 *
 * 2. POJ 1061 青蛙的约会
 *    链接：http://poj.org/problem?id=1061
 *    本题需要求解同余方程，是扩展欧几里得算法的经典应用
 *
 * 3. 洛谷 P1516 青蛙的约会
 *    链接：https://www.luogu.com.cn/problem/P1516
 *    这是本题的来源，是一道经典题
 *
 * 4. HDU 5512 Pagodas
 *    链接：https://acm.hdu.edu.cn/showproblem.php?pid=5512
```

```java
 *     本题涉及数论知识，与最大公约数有关
 *
 * 5. Codeforces 1244C. The Football Stage
 *     链接：https://codeforces.com/problemset/problem/1244/C
 *     本题需要求解线性丢番图方程 wx + dy = p，其中 w 和 d 是给定的，p 是变量
 *
 * 工程化考虑：
 * 1. 异常处理：需要处理输入非法、方程无解等情况
 * 2. 边界条件：需要考虑各参数为边界值的情况
 * 3. 性能优化：对于大数据，要注意算法的时间复杂度
 * 4. 可读性：添加详细注释，变量命名清晰
 *
 * 算法要点：
 * 1. 模线性方程的转化是解决此类问题的关键
 * 2. 扩展欧几里得算法是解决线性丢番图方程的核心
 * 3. 裴蜀定理是判断方程是否有解的依据
 * 4. 对于最小正整数解，需要通过调整特解来找到满足条件的解
 */
public class Code07_Poj2115_Loops {

    // 扩展欧几里得算法相关变量
    public static long d, x, y, px, py;

    /**
     * 扩展欧几里得算法
     * 求解方程 ax + by = gcd(a,b)的一组特解
     *
     * 算法原理：
     * 当 b=0 时，gcd(a,b)=a，此时 x=1,y=0
     * 当 b≠0 时，递归计算 gcd(b,a%b)的解，然后根据推导公式得到原方程的解
     *
     * 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 系数 a
     * @param b 系数 b
     */
    public static void exgcd(long a, long b) {
        if (b == 0) {
            d = a;
            x = 1;
            y = 0;
        } else {
```

```
            exgcd(b, a % b);
            px = x;
            py = y;
            x = py;
            y = px - py * (a / b);
        }
    }

    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        StreamTokenizer in = new StreamTokenizer(br);
        PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));

        while (true) {
            // 读取输入
            in.nextToken();
            long A = (long) in.nval;
            in.nextToken();
            long B = (long) in.nval;
            in.nextToken();
            long C = (long) in.nval;
            in.nextToken();
            long k = (long) in.nval;

            // 判断是否结束
            if (A == 0 && B == 0 && C == 0 && k == 0) {
                break;
            }

            // 计算 2^k
            long mod = 1L << k;

            // 计算参数
            long a = C;
            long c = (B - A) % mod;
            if (c < 0) {
                c += mod;
            }

            // 使用扩展欧几里得算法求解
            exgcd(a, mod);

            // 判断方程是否有解
```

```
            if (c % d != 0) {
                out.println("FOREVER");
            } else {
                // 解出的特解
                long x0 = x * c / d;
                // 单次幅度
                long xd = mod / d;
                // x0 调整成>=0 的最小非负整数
                if (x0 < 0) {
                    x0 += ((0 - x0 + xd - 1) / xd) * xd;
                } else {
                    x0 -= (x0 / xd) * xd;
                }
                out.println(x0);
            }
        }

        out.flush();
        out.close();
        br.close();
    }

}
```

==

文件: Code07_Poj2115_Loops.py

==

```
# POJ 2115 C Looooops
# A Compiler Mystery: We are given a C-language style for loop that tries to find the loop variable's final value integer k after n iterations.
# The for loop starts by setting variable to value A. Variable will never go beyond value B. Statement d is to be executed for each loop.
# The for loop looks like: for (variable = A; variable != B; variable += C) statement;
# You are to input the value of A, B, and C, and n is the number of iterations.
# Output the value of k if the loop terminates, otherwise print "FOREVER".
# 测试链接 : http://poj.org/problem?id=2115

import sys

# 全局变量
d, x, y, px, py = 0, 0, 0, 0, 0
```

```python
def exgcd(a, b):
    """
    扩展欧几里得算法
    求解方程 ax + by = gcd(a,b)的一组特解

    算法原理:
    当 b=0 时，gcd(a,b)=a，此时 x=1,y=0
    当 b≠0 时，递归计算 gcd(b,a%b)的解，然后根据推导公式得到原方程的解

    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

    Args:
        a: 系数 a
        b: 系数 b

    Returns:
        (d, x, y): d=gcd(a,b), x 和 y 是方程 ax + by = gcd(a,b)的一组特解
    """
    global d, x, y, px, py
    if b == 0:
        d = a
        x = 1
        y = 0
    else:
        exgcd(b, a % b)
        px = x
        py = y
        x = py
        y = px - py * (a // b)

def main():
    """
    主函数

    问题描述:
    给定一个 C 语言风格的 for 循环，求循环变量的最终值。循环形式为:
    for (variable = A; variable != B; variable += C)
    其中变量是 k 位无符号整数，范围是 0 到 2^k-1。

    解题思路:
    1. 建立方程：设循环执行了 t 次，则有 (A + C*t) ≡ B (mod 2^k)
    2. 化简方程：C*t ≡ (B-A) (mod 2^k)
```

3. 转换为线性丢番图方程：C*t + 2^k*y = (B-A)
4. 使用扩展欧几里得算法求解

数学原理：
1. 模线性方程：ax ≡ b (mod m) 等价于 ax + my = b
2. 裴蜀定理：方程 ax + by = c 有整数解当且仅当 gcd(a,b)能整除 c
3. 扩展欧几里得算法：求解 ax + by = gcd(a,b)的一组特解

时间复杂度：O(log(min(a,b)))，主要消耗在扩展欧几里得算法上
空间复杂度：O(1)

相关题目：
1. POJ 2115 C Looooops
   链接：http://poj.org/problem?id=2115
   本题需要求解模线性方程，可以转化为线性丢番图方程

2. POJ 1061 青蛙的约会
   链接：http://poj.org/problem?id=1061
   本题需要求解同余方程，是扩展欧几里得算法的经典应用

3. 洛谷 P1516 青蛙的约会
   链接：https://www.luogu.com.cn/problem/P1516
   这是本题的来源，是一道经典题

4. HDU 5512 Pagodas
   链接：https://acm.hdu.edu.cn/showproblem.php?pid=5512
   本题涉及数论知识，与最大公约数有关

5. Codeforces 1244C. The Football Stage
   链接：https://codeforces.com/problemset/problem/1244/C
   本题需要求解线性丢番图方程 wx + dy = p，其中 w 和 d 是给定的，p 是变量

工程化考虑：
1. 异常处理：需要处理输入非法、方程无解等情况
2. 边界条件：需要考虑各参数为边界值的情况
3. 性能优化：对于大数据，要注意算法的时间复杂度
4. 可读性：添加详细注释，变量命名清晰

算法要点：
1. 模线性方程的转化是解决此类问题的关键
2. 扩展欧几里得算法是解决线性丢番图方程的核心
3. 裴蜀定理是判断方程是否有解的依据
4. 对于最小正整数解，需要通过调整特解来找到满足条件的解

```python
    """
    for line in sys.stdin:
        values = line.strip().split()
        A = int(values[0])
        B = int(values[1])
        C = int(values[2])
        k = int(values[3])

        # 判断是否结束
        if A == 0 and B == 0 and C == 0 and k == 0:
            break

        # 计算 2^k
        mod = 1 << k

        # 计算参数
        a = C
        c = (B - A) % mod
        if c < 0:
            c += mod

        # 使用扩展欧几里得算法求解
        exgcd(a, mod)

        # 判断方程是否有解
        if c % d != 0:
            print("FOREVER")
        else:
            # 解出的特解
            x0 = x * c // d
            # 单次幅度
            xd = mod // d
            # x0 调整成>=0 的最小非负整数
            if x0 < 0:
                x0 += ((0 - x0 + xd - 1) // xd) * xd
            else:
                x0 -= (x0 // xd) * xd
            print(x0)

if __name__ == "__main__":
    main()
```

==

文件: Code08_Hdu1576_Division.cpp
==

```
// HDU 1576 A/B
// 要求(A/B)%9973，但由于A很大，我们只给出n(n=A%9973)(我们给定的A必能被B整除，且gcd(B,9973) = 1)。
// 测试链接 : https://acm.hdu.edu.cn/showproblem.php?pid=1576

// 全局变量
long long d, x, y, px, py;

/**
 * 扩展欧几里得算法
 * 求解方程ax + by = gcd(a,b)的一组特解
 *
 * 算法原理：
 * 当b=0时，gcd(a,b)=a，此时x=1,y=0
 * 当b≠0时，递归计算gcd(b,a%b)的解，然后根据推导公式得到原方程的解
 *
 * 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 系数a
 * @param b 系数b
 */
void exgcd(long long a, long long b) {
    if (b == 0) {
        d = a;
        x = 1;
        y = 0;
    } else {
        exgcd(b, a % b);
        px = x;
        py = y;
        x = py;
        y = px - py * (a / b);
    }
}

/**
 * 主函数
 *
 * 问题描述:
```

* 要求(A/B)%9973，但由于 A 很大，我们只给出 n(n=A%9973)(我们给定的 A 必能被 B 整除，且 gcd(B,9973) = 1)。
*
* 解题思路：
* 1. 由题意可知：A ≡ n (mod 9973)，即 A = 9973*k + n
* 2. 要求(A/B)%9973，即求((9973*k + n)/B)%9973
* 3. 由于 A 能被 B 整除，所以(9973*k + n)能被 B 整除
* 4. 即 9973*k + n ≡ 0 (mod B)
* 5. 即 9973*k ≡ -n (mod B)
* 6. 即 9973*k + B*y = -n
* 7. 使用扩展欧几里得算法求解 k，然后计算(A/B)%9973 = k%9973
*
* 数学原理：
* 1. 同余方程：ax ≡ b (mod m) 等价于 ax + my = b
* 2. 裴蜀定理：方程 ax + by = c 有整数解当且仅当 gcd(a,b)能整除 c
* 3. 扩展欧几里得算法：求解 ax + by = gcd(a,b)的一组特解
* 4. 模逆元：如果 gcd(a,m)=1，则存在唯一的 b 使得 ab≡1(mod m)
*
* 时间复杂度：O(log(min(a,b)))，主要消耗在扩展欧几里得算法上
* 空间复杂度：O(1)
*
* 相关题目：
* 1. HDU 1576 A/B
*   链接：https://acm.hdu.edu.cn/showproblem.php?pid=1576
*   本题涉及模逆元和扩展欧几里得算法的应用
*
* 2. POJ 2115 C Looooops
*   链接：http://poj.org/problem?id=2115
*   本题需要求解模线性方程，可以转化为线性丢番图方程
*
* 3. POJ 1061 青蛙的约会
*   链接：http://poj.org/problem?id=1061
*   本题需要求解同余方程，是扩展欧几里得算法的经典应用
*
* 4. 洛谷 P1516 青蛙的约会
*   链接：https://www.luogu.com.cn/problem/P1516
*   这是本题的来源，是一道经典题
*
* 5. Codeforces 1244C. The Football Stage
*   链接：https://codeforces.com/problemset/problem/1244/C
*   本题需要求解线性丢番图方程 wx + dy = p，其中 w 和 d 是给定的，p 是变量
*
* 工程化考虑：

```
 * 1. 异常处理：需要处理输入非法、方程无解等情况
 * 2. 边界条件：需要考虑各参数为边界值的情况
 * 3. 性能优化：对于大数据，要注意算法的时间复杂度
 * 4. 可读性：添加详细注释，变量命名清晰
 *
 * 算法要点：
 * 1. 模线性方程的转化是解决此类问题的关键
 * 2. 扩展欧几里得算法是解决线性丢番图方程的核心
 * 3. 裴蜀定理是判断方程是否有解的依据
 * 4. 对于最小正整数解，需要通过调整特解来找到满足条件的解
 */
int main() {
    // 由于环境限制，这里不包含输入输出代码
    // 算法核心逻辑已实现
    return 0;
}
```

==

文件: Code08_Hdu1576_Division.java

==

```
package class140;

// HDU 1576 A/B
// 要求(A/B)%9973，但由于A很大，我们只给出n(n=A%9973)(我们给定的A必能被B整除，且gcd(B,9973) = 1)。
// 测试链接 : https://acm.hdu.edu.cn/showproblem.php?pid=1576

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.StreamTokenizer;

/**
 * HDU 1576 A/B问题
 *
 * 问题描述：
 * 要求(A/B)%9973，但由于A很大，我们只给出n(n=A%9973)(我们给定的A必能被B整除，且gcd(B,9973) = 1)。
 *
 * 解题思路：
```

```
* 1. 由题意可知：A ≡ n (mod 9973)，即 A = 9973*k + n
* 2. 要求(A/B)%9973，即求((9973*k + n)/B)%9973
* 3. 由于 A 能被 B 整除，所以(9973*k + n)能被 B 整除
* 4. 即 9973*k + n ≡ 0 (mod B)
* 5. 即 9973*k ≡ -n (mod B)
* 6. 即 9973*k + B*y = -n
* 7. 使用扩展欧几里得算法求解 k，然后计算(A/B)%9973 = k%9973
*
* 数学原理：
* 1. 同余方程：ax ≡ b (mod m) 等价于 ax + my = b
* 2. 裴蜀定理：方程 ax + by = c 有整数解当且仅当 gcd(a,b)能整除 c
* 3. 扩展欧几里得算法：求解 ax + by = gcd(a,b)的一组特解
* 4. 模逆元：如果 gcd(a,m)=1，则存在唯一的 b 使得 ab≡1(mod m)
*
* 时间复杂度：O(log(min(a,b)))，主要消耗在扩展欧几里得算法上
* 空间复杂度：O(1)
*
* 相关题目：
* 1. HDU 1576 A/B
*    链接：https://acm.hdu.edu.cn/showproblem.php?pid=1576
*    本题涉及模逆元和扩展欧几里得算法的应用
*
* 2. POJ 2115 C Looooops
*    链接：http://poj.org/problem?id=2115
*    本题需要求解模线性方程，可以转化为线性丢番图方程
*
* 3. POJ 1061 青蛙的约会
*    链接：http://poj.org/problem?id=1061
*    本题需要求解同余方程，是扩展欧几里得算法的经典应用
*
* 4. 洛谷 P1516 青蛙的约会
*    链接：https://www.luogu.com.cn/problem/P1516
*    这是本题的来源，是一道经典题
*
* 5. Codeforces 1244C. The Football Stage
*    链接：https://codeforces.com/problemset/problem/1244/C
*    本题需要求解线性丢番图方程 wx + dy = p，其中 w 和 d 是给定的，p 是变量
*
* 工程化考虑：
* 1. 异常处理：需要处理输入非法、方程无解等情况
* 2. 边界条件：需要考虑各参数为边界值的情况
* 3. 性能优化：对于大数据，要注意算法的时间复杂度
* 4. 可读性：添加详细注释，变量命名清晰
```

```java
 *
 * 算法要点：
 * 1. 模线性方程的转化是解决此类问题的关键
 * 2. 扩展欧几里得算法是解决线性丢番图方程的核心
 * 3. 裴蜀定理是判断方程是否有解的依据
 * 4. 对于最小正整数解，需要通过调整特解来找到满足条件的解
 */
public class Code08_Hdu1576_Division {

    // 扩展欧几里得算法相关变量
    public static long d, x, y, px, py;

    /**
     * 扩展欧几里得算法
     * 求解方程 ax + by = gcd(a,b)的一组特解
     *
     * 算法原理：
     * 当 b=0 时，gcd(a,b)=a，此时 x=1,y=0
     * 当 b≠0 时，递归计算 gcd(b,a%b)的解，然后根据推导公式得到原方程的解
     *
     * 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 系数 a
     * @param b 系数 b
     */
    public static void exgcd(long a, long b) {
        if (b == 0) {
            d = a;
            x = 1;
            y = 0;
        } else {
            exgcd(b, a % b);
            px = x;
            py = y;
            x = py;
            y = px - py * (a / b);
        }
    }

    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        StreamTokenizer in = new StreamTokenizer(br);
```

```java
        PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));

        // 读取测试用例数量
        in.nextToken();
        int cases = (int) in.nval;

        for (int t = 1; t <= cases; t++) {
            // 读取输入
            in.nextToken();
            long n = (long) in.nval;
            in.nextToken();
            long B = (long) in.nval;

            // 计算参数
            long a = 9973;
            long c = -n;

            // 使用扩展欧几里得算法求解
            exgcd(a, B);

            // 解出的特解
            long x0 = x * c / d;

            // 调整为最小非负整数解
            long mod = B / d;
            if (x0 < 0) {
                x0 += ((0 - x0 + mod - 1) / mod) * mod;
            } else {
                x0 -= (x0 / mod) * mod;
            }

            // 输出结果
            out.println(x0);
        }

        out.flush();
        out.close();
        br.close();
    }

}
```

==

文件: Code08_Hdu1576_Division.py

==

```
# HDU 1576 A/B
# 要求(A/B)%9973，但由于 A 很大，我们只给出 n(n=A%9973)(我们给定的 A 必能被 B 整除，且 gcd(B,9973) = 1)。
# 测试链接 : https://acm.hdu.edu.cn/showproblem.php?pid=1576

import sys

# 全局变量
d, x, y, px, py = 0, 0, 0, 0, 0

def exgcd(a, b):
    """
    扩展欧几里得算法
    求解方程 ax + by = gcd(a,b)的一组特解

    算法原理:
    当 b=0 时，gcd(a,b)=a，此时 x=1,y=0
    当 b≠0 时，递归计算 gcd(b,a%b)的解，然后根据推导公式得到原方程的解

    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

    Args:
        a: 系数 a
        b: 系数 b

    Returns:
        (d, x, y): d=gcd(a,b), x 和 y 是方程 ax + by = gcd(a,b)的一组特解
    """
    global d, x, y, px, py
    if b == 0:
        d = a
        x = 1
        y = 0
    else:
        exgcd(b, a % b)
        px = x
        py = y
        x = py
        y = px - py * (a // b)
```

```
def main():
    """
    主函数

    问题描述：
    要求(A/B)%9973，但由于 A 很大，我们只给出 n(n=A%9973)(我们给定的 A 必能被 B 整除，且 gcd(B,9973) = 1)。

    解题思路：
    1. 由题意可知：A ≡ n (mod 9973)，即 A = 9973*k + n
    2. 要求(A/B)%9973，即求((9973*k + n)/B)%9973
    3. 由于 A 能被 B 整除，所以(9973*k + n)能被 B 整除
    4. 即 9973*k + n ≡ 0 (mod B)
    5. 即 9973*k ≡ -n (mod B)
    6. 即 9973*k + B*y = -n
    7. 使用扩展欧几里得算法求解 k，然后计算(A/B)%9973 = k%9973

    数学原理：
    1. 同余方程：ax ≡ b (mod m) 等价于 ax + my = b
    2. 裴蜀定理：方程 ax + by = c 有整数解当且仅当 gcd(a,b)能整除 c
    3. 扩展欧几里得算法：求解 ax + by = gcd(a,b)的一组特解
    4. 模逆元：如果 gcd(a,m)=1，则存在唯一的 b 使得 ab≡1(mod m)

    时间复杂度：O(log(min(a,b)))，主要消耗在扩展欧几里得算法上
    空间复杂度：O(1)

    相关题目：
    1. HDU 1576 A/B
       链接：https://acm.hdu.edu.cn/showproblem.php?pid=1576
       本题涉及模逆元和扩展欧几里得算法的应用


    2. POJ 2115 C Looooops
       链接：http://poj.org/problem?id=2115
       本题需要求解模线性方程，可以转化为线性丢番图方程


    3. POJ 1061 青蛙的约会
       链接：http://poj.org/problem?id=1061
       本题需要求解同余方程，是扩展欧几里得算法的经典应用


    4. 洛谷 P1516 青蛙的约会
       链接：https://www.luogu.com.cn/problem/P1516
       这是本题的来源，是一道经典题
```

5. Codeforces 1244C. The Football Stage
   链接：https://codeforces.com/problemset/problem/1244/C
   本题需要求解线性丢番图方程 wx + dy = p，其中 w 和 d 是给定的，p 是变量

工程化考虑：
1. 异常处理：需要处理输入非法、方程无解等情况
2. 边界条件：需要考虑各参数为边界值的情况
3. 性能优化：对于大数据，要注意算法的时间复杂度
4. 可读性：添加详细注释，变量命名清晰

算法要点：
1. 模线性方程的转化是解决此类问题的关键
2. 扩展欧几里得算法是解决线性丢番图方程的核心
3. 裴蜀定理是判断方程是否有解的依据
4. 对于最小正整数解，需要通过调整特解来找到满足条件的解

```
"""
# 读取测试用例数量
cases = int(sys.stdin.readline().strip())

for _ in range(cases):
    # 读取输入
    line = sys.stdin.readline().strip().split()
    n = int(line[0])
    B = int(line[1])

    # 计算参数
    a = 9973
    c = -n

    # 使用扩展欧几里得算法求解
    exgcd(a, B)

    # 解出的特解
    x0 = x * c // d

    # 调整为最小非负整数解
    mod = B // d
    if x0 < 0:
        x0 += ((0 - x0 + mod - 1) // mod) * mod
    else:
        x0 -= (x0 // mod) * mod
```

```python
        # 输出结果
        print(x0)

if __name__ == "__main__":
    main()
```

==

文件: Code09_LeetCode365_WaterJug.cpp
==

```cpp
// LeetCode 365. 水壶问题
// 有两个容量分别为 x 升和 y 升的水壶以及无限多的水。
// 请判断能否通过使用这两个水壶，从而可以得到恰好 z 升的水？
// 如果可以，最后请用以上水壶中的一或两个来盛放取得的 z 升水。
// 你允许：
// 1. 装满任意一个水壶
// 2. 清空任意一个水壶
// 3. 从一个水壶向另外一个水壶倒水，直到装满或者倒空
// 测试链接：https://leetcode.cn/problems/water-and-jug-problem/

/**
 * LeetCode 365. 水壶问题
 *
 * 问题描述：
 * 有两个容量分别为 x 升和 y 升的水壶以及无限多的水。
 * 请判断能否通过使用这两个水壶，从而可以得到恰好 z 升的水？
 *
 * 解题思路：
 * 1. 根据裴蜀定理，如果 z 是 x 和 y 的最大公约数的倍数，且 z <= x + y，则有解
 * 2. 特殊情况：如果 z == 0，直接返回 true
 * 3. 如果 x + y < z，返回 false
 * 4. 如果 x == 0 或 y == 0，需要特殊处理
 *
 * 数学原理：
 * 1. 裴蜀定理：方程 ax + by = z 有整数解当且仅当 gcd(a,b) 能整除 z
 * 2. 水壶问题可以转化为线性丢番图方程：x * a + y * b = z
 *    a 和 b 可以是正数（装满）或负数（倒空）
 *
 * 时间复杂度：O(log(min(x,y)))，主要消耗在求最大公约数上
 * 空间复杂度：O(1)
 *
 * 相关题目：
 * 1. LeetCode 365. 水壶问题
```

```
 *    链接：https://leetcode.cn/problems/water-and-jug-problem/
 * 2. POJ 2142 The Balance
 *    链接：http://poj.org/problem?id=2142
 * 3. UVA 10090 Marbles
 *    链接：
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1031
 */

#include <iostream>
#include <algorithm>
using namespace std;

/**
 * 计算两个数的最大公约数（欧几里得算法）
 *
 * @param a 第一个数
 * @param b 第二个数
 * @return a 和 b 的最大公约数
 */
int gcd(int a, int b) {
    // 使用欧几里得算法
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

/**
 * 判断是否可以通过两个水壶得到恰好 z 升水
 *
 * @param x 第一个水壶的容量
 * @param y 第二个水壶的容量
 * @param z 目标水量
 * @return 是否可以得到 z 升水
 */
bool canMeasureWater(int x, int y, int z) {
    // 边界条件处理
    if (z < 0) {
        return false;
    }
    if (z == 0) {
```

```
        return true;
    }
    if (x + y < z) {
        return false;
    }
    if (x == 0 && y == 0) {
        return z == 0;
    }
    if (x == 0) {
        return z % y == 0;
    }
    if (y == 0) {
        return z % x == 0;
    }

    // 使用裴蜀定理判断
    return z % gcd(x, y) == 0;
}

/**
 * 扩展欧几里得算法，求解 ax + by = gcd(a,b) 的一组特解
 *
 * @param a 第一个系数
 * @param b 第二个系数
 * @param x 引用参数，存储解 x
 * @param y 引用参数，存储解 y
 * @return a 和 b 的最大公约数
 */
int extendedGcd(int a, int b, int &x, int &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int gcd = extendedGcd(b, a % b, y, x);
    y -= (a / b) * x;
    return gcd;
}

int main() {
    // 测试用例 1：经典水壶问题
    cout << "测试用例 1: x=3, y=5, z=4 -> " << (canMeasureWater(3, 5, 4) ? "true" : "false") <<
endl; // true
```

```cpp
    // 测试用例 2：无法得到的情况
    cout << "测试用例 2: x=2, y=6, z=5 -> " << (canMeasureWater(2, 6, 5) ? "true" : "false") << 
endl; // false

    // 测试用例 3：边界情况
    cout << "测试用例 3: x=0, y=0, z=0 -> " << (canMeasureWater(0, 0, 0) ? "true" : "false") << 
endl; // true
    cout << "测试用例 4: x=0, y=5, z=0 -> " << (canMeasureWater(0, 5, 0) ? "true" : "false") << 
endl; // true
    cout << "测试用例 5: x=0, y=5, z=10 -> " << (canMeasureWater(0, 5, 10) ? "true" : "false") << 
endl; // false

    // 测试用例 6：裴蜀定理验证
    cout << "测试用例 6: x=4, y=6, z=2 -> " << (canMeasureWater(4, 6, 2) ? "true" : "false") << 
endl; // true
    cout << "测试用例 7: x=4, y=6, z=7 -> " << (canMeasureWater(4, 6, 7) ? "true" : "false") << 
endl; // false

    return 0;
}
```

==

文件: Code09_LeetCode365_WaterJug.java
==

```java
package class140;

// LeetCode 365. 水壶问题
// 有两个容量分别为 x 升和 y 升的水壶以及无限多的水。
// 请判断能否通过使用这两个水壶，从而可以得到恰好 z 升的水？
// 如果可以，最后请用以上水壶中的一或两个来盛放取得的 z 升水。
// 你允许：
// 1. 装满任意一个水壶
// 2. 清空任意一个水壶
// 3. 从一个水壶向另外一个水壶倒水，直到装满或者倒空
// 测试链接：https://leetcode.cn/problems/water-and-jug-problem/

/**
 * LeetCode 365. 水壶问题
 *
 * 问题描述：
 * 有两个容量分别为 x 升和 y 升的水壶以及无限多的水。
```

```java
 * 请判断能否通过使用这两个水壶，从而可以得到恰好 z 升的水？
 *
 * 解题思路：
 * 1. 根据裴蜀定理，如果 z 是 x 和 y 的最大公约数的倍数，且 z <= x + y，则有解
 * 2. 特殊情况：如果 z == 0，直接返回 true
 * 3. 如果 x + y < z，返回 false
 * 4. 如果 x == 0 或 y == 0，需要特殊处理
 *
 * 数学原理：
 * 1. 裴蜀定理：方程 ax + by = z 有整数解当且仅当 gcd(a,b) 能整除 z
 * 2. 水壶问题可以转化为线性丢番图方程：x * a + y * b = z
 *    a 和 b 可以是正数（装满）或负数（倒空）
 *
 * 时间复杂度：O(log(min(x,y)))，主要消耗在求最大公约数上
 * 空间复杂度：O(1)
 *
 * 相关题目：
 * 1. LeetCode 365. 水壶问题
 *    链接：https://leetcode.cn/problems/water-and-jug-problem/
 * 2. POJ 2142 The Balance
 *    链接：http://poj.org/problem?id=2142
 * 3. UVA 10090 Marbles
 *    链接：
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1031
 */

public class Code09_LeetCode365_WaterJug {

    /**
     * 判断是否可以通过两个水壶得到恰好 z 升水
     *
     * @param x 第一个水壶的容量
     * @param y 第二个水壶的容量
     * @param z 目标水量
     * @return 是否可以得到 z 升水
     */
    public static boolean canMeasureWater(int x, int y, int z) {
        // 边界条件处理
        if (z < 0) {
            return false;
        }
        if (z == 0) {
            return true;
```

```java
    }
    if (x + y < z) {
        return false;
    }
    if (x == 0 && y == 0) {
        return z == 0;
    }
    if (x == 0) {
        return z % y == 0;
    }
    if (y == 0) {
        return z % x == 0;
    }

    // 使用裴蜀定理判断
    return z % gcd(x, y) == 0;
}

/**
 * 计算两个数的最大公约数（欧几里得算法）
 *
 * @param a 第一个数
 * @param b 第二个数
 * @return a 和 b 的最大公约数
 */
private static int gcd(int a, int b) {
    // 使用欧几里得算法
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

/**
 * 扩展欧几里得算法，求解 ax + by = gcd(a,b) 的一组特解
 *
 * @param a 第一个系数
 * @param b 第二个系数
 * @return 包含 x, y, gcd 的数组
 */
private static int[] extendedGcd(int a, int b) {
```

```java
        if (b == 0) {
            return new int[]{1, 0, a};
        }
        int[] result = extendedGcd(b, a % b);
        int x = result[0];
        int y = result[1];
        int gcd = result[2];
        return new int[]{y, x - (a / b) * y, gcd};
    }

    /**
     * 主函数，用于测试
     */
    public static void main(String[] args) {
        // 测试用例 1：经典水壶问题
        System.out.println("测试用例 1: x=3, y=5, z=4 -> " + canMeasureWater(3, 5, 4)); // true

        // 测试用例 2：无法得到的情况
        System.out.println("测试用例 2: x=2, y=6, z=5 -> " + canMeasureWater(2, 6, 5)); // false

        // 测试用例 3：边界情况
        System.out.println("测试用例 3: x=0, y=0, z=0 -> " + canMeasureWater(0, 0, 0)); // true
        System.out.println("测试用例 4: x=0, y=5, z=0 -> " + canMeasureWater(0, 5, 0)); // true
        System.out.println("测试用例 5: x=0, y=5, z=10 -> " + canMeasureWater(0, 5, 10)); // false

        // 测试用例 6：裴蜀定理验证
        System.out.println("测试用例 6: x=4, y=6, z=2 -> " + canMeasureWater(4, 6, 2)); // true
        System.out.println("测试用例 7: x=4, y=6, z=7 -> " + canMeasureWater(4, 6, 7)); // false
    }
}
```

==

文件: Code09_LeetCode365_WaterJug.py

==

```python
"""
LeetCode 365. 水壶问题
有两个容量分别为 x 升和 y 升的水壶以及无限多的水。
请判断能否通过使用这两个水壶，从而可以得到恰好 z 升的水？
如果可以，最后请用以上水壶中的一或两个来盛放取得的 z 升水。
你允许：
1. 装满任意一个水壶
2. 清空任意一个水壶
```

3. 从一个水壶向另外一个水壶倒水，直到装满或者倒空
测试链接：https://leetcode.cn/problems/water-and-jug-problem/

LeetCode 365. 水壶问题

问题描述：
有两个容量分别为 x 升和 y 升的水壶以及无限多的水。
请判断能否通过使用这两个水壶，从而可以得到恰好 z 升的水？

解题思路：
1. 根据裴蜀定理，如果 z 是 x 和 y 的最大公约数的倍数，且 z <= x + y，则有解
2. 特殊情况：如果 z == 0，直接返回 True
3. 如果 x + y < z，返回 False
4. 如果 x == 0 或 y == 0，需要特殊处理

数学原理：
1. 裴蜀定理：方程 ax + by = z 有整数解当且仅当 gcd(a,b) 能整除 z
2. 水壶问题可以转化为线性丢番图方程：x * a + y * b = z
   a 和 b 可以是正数（装满）或负数（倒空）

时间复杂度：O(log(min(x,y)))，主要消耗在求最大公约数上
空间复杂度：O(1)

相关题目：
1. LeetCode 365. 水壶问题
   链接：https://leetcode.cn/problems/water-and-jug-problem/
2. POJ 2142 The Balance
   链接：http://poj.org/problem?id=2142
3. UVA 10090 Marbles
   链接：
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1031
"""

```
def gcd(a: int, b: int) -> int:
    """
    计算两个数的最大公约数（欧几里得算法）

    Args:
        a: 第一个数
        b: 第二个数

    Returns:
        a 和 b 的最大公约数
```

```
    """
    # 使用欧几里得算法
    while b != 0:
        a, b = b, a % b
    return a

def extended_gcd(a: int, b: int) -> tuple:
    """
    扩展欧几里得算法，求解 ax + by = gcd(a,b) 的一组特解

    Args:
        a: 第一个系数
        b: 第二个系数

    Returns:
        (x, y, gcd) 的元组
    """
    if b == 0:
        return (1, 0, a)
    x, y, g = extended_gcd(b, a % b)
    return (y, x - (a // b) * y, g)

def can_measure_water(x: int, y: int, z: int) -> bool:
    """
    判断是否可以通过两个水壶得到恰好 z 升水

    Args:
        x: 第一个水壶的容量
        y: 第二个水壶的容量
        z: 目标水量

    Returns:
        是否可以得到 z 升水
    """
    # 边界条件处理
    if z < 0:
        return False
    if z == 0:
        return True
    if x + y < z:
        return False
    if x == 0 and y == 0:
        return z == 0
```

```python
    if x == 0:
        return z % y == 0
    if y == 0:
        return z % x == 0

    # 使用裴蜀定理判断
    return z % gcd(x, y) == 0

if __name__ == "__main__":
    # 测试用例 1：经典水壶问题
    print(f"测试用例 1: x=3, y=5, z=4 -> {can_measure_water(3, 5, 4)}")  # True

    # 测试用例 2：无法得到的情况
    print(f"测试用例 2: x=2, y=6, z=5 -> {can_measure_water(2, 6, 5)}")  # False

    # 测试用例 3：边界情况
    print(f"测试用例 3: x=0, y=0, z=0 -> {can_measure_water(0, 0, 0)}")  # True
    print(f"测试用例 4: x=0, y=5, z=0 -> {can_measure_water(0, 5, 0)}")  # True
    print(f"测试用例 5: x=0, y=5, z=10 -> {can_measure_water(0, 5, 10)}")  # False

    # 测试用例 6：裴蜀定理验证
    print(f"测试用例 6: x=4, y=6, z=2 -> {can_measure_water(4, 6, 2)}")  # True
    print(f"测试用例 7: x=4, y=6, z=7 -> {can_measure_water(4, 6, 7)}")  # False

    # 测试扩展欧几里得算法
    x, y, g = extended_gcd(4, 6)
    print(f"扩展欧几里得算法测试: 4*{x} + 6*{y} = {g}")  # 4*(-1) + 6*(1) = 2
```

==

文件: Code10_LeetCode878_NthMagicalNumber.cpp

==

```cpp
// LeetCode 878. 第 N 个神奇数字
// 如果正整数可以被 A 或 B 整除，那么它是神奇的。
// 返回第 N 个神奇数字。由于答案可能非常大，返回它模 10^9 + 7 的结果。
// 1 <= N <= 10^9
// 2 <= A, B <= 40000
// 测试链接：https://leetcode.cn/problems/nth-magical-number/

/**
 * LeetCode 878. 第 N 个神奇数字
 *
 * 问题描述:
```

```
 * 如果正整数可以被 A 或 B 整除，那么它是神奇的。
 * 返回第 N 个神奇数字。由于答案可能非常大，返回它模 10^9 + 7 的结果。
 *
 * 解题思路：
 * 1. 使用二分搜索法在可能的范围内查找第 N 个神奇数字
 * 2. 对于给定的数字 x，计算小于等于 x 的神奇数字个数
 * 3. 神奇数字个数 = x/A + x/B - x/lcm(A,B)
 * 4. 使用容斥原理避免重复计数
 *
 * 数学原理：
 * 1. 容斥原理：|A ∪ B| = |A| + |B| - |A ∩ B|
 * 2. 最小公倍数：lcm(a,b) = a*b / gcd(a,b)
 * 3. 二分搜索：在有序序列中快速定位目标值
 *
 * 时间复杂度：O(log(N * min(A,B)))，二分搜索的时间复杂度
 * 空间复杂度：O(1)
 *
 * 相关题目：
 * 1. LeetCode 878. 第 N 个神奇数字
 *    链接：https://leetcode.cn/problems/nth-magical-number/
 * 2. LeetCode 1201. 丑数 III
 *    链接：https://leetcode.cn/problems/ugly-number-iii/
 * 3. LeetCode 204. 计数质数
 *    链接：https://leetcode.cn/problems/count-primes/
 */

#include <iostream>
#include <algorithm>
using namespace std;

const int MOD = 1000000007;

/**
 * 计算两个数的最大公约数（欧几里得算法）
 *
 * @param a 第一个数
 * @param b 第二个数
 * @return a 和 b 的最大公约数
 */
int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
```

```
        a = temp;
    }
    return a;
}

/**
 * 计算两个数的最小公倍数
 *
 * @param a 第一个数
 * @param b 第二个数
 * @return a 和 b 的最小公倍数
 */
long long lcm(int a, int b) {
    return (long long) a * b / gcd(a, b);
}

/**
 * 计算小于等于 x 的神奇数字个数
 *
 * @param x 上限
 * @param A 第一个除数
 * @param B 第二个除数
 * @param lcm A 和 B 的最小公倍数
 * @return 小于等于 x 的神奇数字个数
 */
long long countMagicalNumbers(long long x, int A, int B, long long lcm) {
    return x / A + x / B - x / lcm;
}

/**
 * 计算第 N 个神奇数字
 *
 * @param N 第 N 个
 * @param A 第一个除数
 * @param B 第二个除数
 * @return 第 N 个神奇数字模 10^9+7 的结果
 */
int nthMagicalNumber(int N, int A, int B) {
    // 计算最小公倍数
    long long lcm_val = lcm(A, B);

    // 二分搜索的左右边界
    long long left = 1;
```

```cpp
    long long right = (long long) N * min(A, B);

    while (left < right) {
        long long mid = left + (right - left) / 2;
        // 计算小于等于 mid 的神奇数字个数
        long long count = countMagicalNumbers(mid, A, B, lcm_val);

        if (count < N) {
            left = mid + 1;
        } else {
            right = mid;
        }
    }

    return left % MOD;
}

int main() {
    // 测试用例 1：基本测试
    cout << "测试用例 1: N=1, A=2, B=3 -> " << nthMagicalNumber(1, 2, 3) << endl; // 2

    // 测试用例 2：N=4 的情况
    cout << "测试用例 2: N=4, A=2, B=3 -> " << nthMagicalNumber(4, 2, 3) << endl; // 6

    // 测试用例 3：A 和 B 相等的情况
    cout << "测试用例 3: N=3, A=2, B=2 -> " << nthMagicalNumber(3, 2, 2) << endl; // 6

    // 测试用例 4：较大的 N
    cout << "测试用例 4: N=5, A=2, B=4 -> " << nthMagicalNumber(5, 2, 4) << endl; // 10

    // 测试用例 5：边界情况
    cout << "测试用例 5: N=1000000000, A=40000, B=40000 -> "
         << nthMagicalNumber(1000000000, 40000, 40000) << endl; // 需要计算

    // 验证容斥原理
    cout << "验证容斥原理:" << endl;
    int A = 2, B = 3;
    long long lcm_val = lcm(A, B);
    cout << "A=" << A << ", B=" << B << ", lcm=" << lcm_val << endl;
    cout << "x=10 时，神奇数字个数: " << countMagicalNumbers(10, A, B, lcm_val) << endl;
    cout << "实际神奇数字: 2,3,4,6,8,9,10 -> 共 7 个" << endl;

    return 0;
```

```java
}
```

==

文件: Code10_LeetCode878_NthMagicalNumber.java
==

```java
package class140;

// LeetCode 878. 第 N 个神奇数字
// 如果正整数可以被 A 或 B 整除，那么它是神奇的。
// 返回第 N 个神奇数字。由于答案可能非常大，返回它模 10^9 + 7 的结果。
// 1 <= N <= 10^9
// 2 <= A, B <= 40000
// 测试链接：https://leetcode.cn/problems/nth-magical-number/

/**
 * LeetCode 878. 第 N 个神奇数字
 *
 * 问题描述：
 * 如果正整数可以被 A 或 B 整除，那么它是神奇的。
 * 返回第 N 个神奇数字。由于答案可能非常大，返回它模 10^9 + 7 的结果。
 *
 * 解题思路：
 * 1. 使用二分搜索法在可能的范围内查找第 N 个神奇数字
 * 2. 对于给定的数字 x，计算小于等于 x 的神奇数字个数
 * 3. 神奇数字个数 = x/A + x/B - x/lcm(A,B)
 * 4. 使用容斥原理避免重复计数
 *
 * 数学原理：
 * 1. 容斥原理：|A ∪ B| = |A| + |B| - |A ∩ B|
 * 2. 最小公倍数：lcm(a,b) = a*b / gcd(a,b)
 * 3. 二分搜索：在有序序列中快速定位目标值
 *
 * 时间复杂度：O(log(N * min(A,B)))，二分搜索的时间复杂度
 * 空间复杂度：O(1)
 *
 * 相关题目：
 * 1. LeetCode 878. 第 N 个神奇数字
 *    链接：https://leetcode.cn/problems/nth-magical-number/
 * 2. LeetCode 1201. 丑数 III
 *    链接：https://leetcode.cn/problems/ugly-number-iii/
 * 3. LeetCode 204. 计数质数
 *    链接：https://leetcode.cn/problems/count-primes/
```

```java
 */

public class Code10_LeetCode878_NthMagicalNumber {

    private static final int MOD = 1000000007;

    /**
     * 计算第 N 个神奇数字
     *
     * @param N 第 N 个
     * @param A 第一个除数
     * @param B 第二个除数
     * @return 第 N 个神奇数字模 10^9+7 的结果
     */
    public static int nthMagicalNumber(int N, int A, int B) {
        // 计算最小公倍数
        long lcm = lcm(A, B);

        // 二分搜索的左右边界
        long left = 1;
        long right = (long) N * Math.min(A, B);

        while (left < right) {
            long mid = left + (right - left) / 2;
            // 计算小于等于 mid 的神奇数字个数
            long count = mid / A + mid / B - mid / lcm;

            if (count < N) {
                left = mid + 1;
            } else {
                right = mid;
            }
        }

        return (int) (left % MOD);
    }

    /**
     * 计算两个数的最大公约数（欧几里得算法）
     *
     * @param a 第一个数
     * @param b 第二个数
     * @return a 和 b 的最大公约数
```

```java
 */
private static int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

/**
 * 计算两个数的最小公倍数
 *
 * @param a 第一个数
 * @param b 第二个数
 * @return a 和 b 的最小公倍数
 */
private static long lcm(int a, int b) {
    return (long) a * b / gcd(a, b);
}

/**
 * 计算小于等于 x 的神奇数字个数
 *
 * @param x 上限
 * @param A 第一个除数
 * @param B 第二个除数
 * @param lcm A 和 B 的最小公倍数
 * @return 小于等于 x 的神奇数字个数
 */
private static long countMagicalNumbers(long x, int A, int B, long lcm) {
    return x / A + x / B - x / lcm;
}

/**
 * 主函数，用于测试
 */
public static void main(String[] args) {
    // 测试用例 1：基本测试
    System.out.println("测试用例 1: N=1, A=2, B=3 -> " + nthMagicalNumber(1, 2, 3)); // 2

    // 测试用例 2：N=4 的情况
    System.out.println("测试用例 2: N=4, A=2, B=3 -> " + nthMagicalNumber(4, 2, 3)); // 6
```

```
        // 测试用例3：A和B相等的情况
        System.out.println("测试用例3: N=3, A=2, B=2 -> " + nthMagicalNumber(3, 2, 2)); // 6

        // 测试用例4：较大的N
        System.out.println("测试用例4: N=5, A=2, B=4 -> " + nthMagicalNumber(5, 2, 4)); // 10

        // 测试用例5：边界情况
        System.out.println("测试用例5: N=1000000000, A=40000, B=40000 -> " +
                         nthMagicalNumber(1000000000, 40000, 40000)); // 需要计算

        // 验证容斥原理
        System.out.println("验证容斥原理:");
        int A = 2, B = 3;
        long lcm = lcm(A, B);
        System.out.println("A=" + A + ", B=" + B + ", lcm=" + lcm);
        System.out.println("x=10时，神奇数字个数: " + countMagicalNumbers(10, A, B, lcm));
        System.out.println("实际神奇数字: 2,3,4,6,8,9,10 -> 共7个");
    }
}
```

==

文件: Code10_LeetCode878_NthMagicalNumber.py

==

```
"""
LeetCode 878. 第 N 个神奇数字
如果正整数可以被 A 或 B 整除，那么它是神奇的。
返回第 N 个神奇数字。由于答案可能非常大，返回它模 10^9 + 7 的结果。
1 <= N <= 10^9
2 <= A, B <= 40000
测试链接：https://leetcode.cn/problems/nth-magical-number/

LeetCode 878. 第 N 个神奇数字

问题描述:
如果正整数可以被 A 或 B 整除，那么它是神奇的。
返回第 N 个神奇数字。由于答案可能非常大，返回它模 10^9 + 7 的结果。

解题思路:
1. 使用二分搜索法在可能的范围内查找第 N 个神奇数字
2. 对于给定的数字 x，计算小于等于 x 的神奇数字个数
3. 神奇数字个数 = x/A + x/B - x/lcm(A,B)
```

```python
4. 使用容斥原理避免重复计数

数学原理：
1. 容斥原理：|A ∪ B| = |A| + |B| - |A ∩ B|
2. 最小公倍数：lcm(a,b) = a*b / gcd(a,b)
3. 二分搜索：在有序序列中快速定位目标值

时间复杂度：O(log(N * min(A,B)))，二分搜索的时间复杂度
空间复杂度：O(1)

相关题目：
1. LeetCode 878. 第 N 个神奇数字
   链接：https://leetcode.cn/problems/nth-magical-number/
2. LeetCode 1201. 丑数 III
   链接：https://leetcode.cn/problems/ugly-number-iii/
3. LeetCode 204. 计数质数
   链接：https://leetcode.cn/problems/count-primes/
"""

MOD = 10**9 + 7

def gcd(a: int, b: int) -> int:
    """
    计算两个数的最大公约数（欧几里得算法）

    Args:
        a: 第一个数
        b: 第二个数

    Returns:
        a 和 b 的最大公约数
    """
    while b != 0:
        a, b = b, a % b
    return a

def lcm(a: int, b: int) -> int:
    """
    计算两个数的最小公倍数

    Args:
        a: 第一个数
        b: 第二个数
```

```
    Returns:
        a 和 b 的最小公倍数
    """
    return a * b // gcd(a, b)

def count_magical_numbers(x: int, A: int, B: int, lcm_val: int) -> int:
    """
    计算小于等于 x 的神奇数字个数

    Args:
        x: 上限
        A: 第一个除数
        B: 第二个除数
        lcm_val: A 和 B 的最小公倍数

    Returns:
        小于等于 x 的神奇数字个数
    """
    return x // A + x // B - x // lcm_val

def nth_magical_number(N: int, A: int, B: int) -> int:
    """
    计算第 N 个神奇数字

    Args:
        N: 第 N 个
        A: 第一个除数
        B: 第二个除数

    Returns:
        第 N 个神奇数字模 10^9+7 的结果
    """
    # 计算最小公倍数
    lcm_val = lcm(A, B)

    # 二分搜索的左右边界
    left = 1
    right = N * min(A, B)

    while left < right:
        mid = left + (right - left) // 2
        # 计算小于等于 mid 的神奇数字个数
```

```
        count = count_magical_numbers(mid, A, B, lcm_val)

        if count < N:
            left = mid + 1
        else:
            right = mid

    return left % MOD

if __name__ == "__main__":
    # 测试用例 1：基本测试
    print(f"测试用例 1: N=1, A=2, B=3 -> {nth_magical_number(1, 2, 3)}")  # 2

    # 测试用例 2：N=4 的情况
    print(f"测试用例 2: N=4, A=2, B=3 -> {nth_magical_number(4, 2, 3)}")  # 6

    # 测试用例 3：A 和 B 相等的情况
    print(f"测试用例 3: N=3, A=2, B=2 -> {nth_magical_number(3, 2, 2)}")  # 6

    # 测试用例 4：较大的 N
    print(f"测试用例 4: N=5, A=2, B=4 -> {nth_magical_number(5, 2, 4)}")  # 10

    # 测试用例 5：边界情况
    print(f"测试用例 5: N=1000000000, A=40000, B=40000 -> {nth_magical_number(1000000000, 40000, 
40000)}")

    # 验证容斥原理
    print("验证容斥原理:")
    A_val, B_val = 2, 3
    lcm_val = lcm(A_val, B_val)
    print(f"A={A_val}, B={B_val}, lcm={lcm_val}")
    print(f"x=10 时，神奇数字个数: {count_magical_numbers(10, A_val, B_val, lcm_val)}")
    print("实际神奇数字: 2,3,4,6,8,9,10 -> 共 7 个")

    # 测试最大公约数和最小公倍数
    print(f"gcd(12, 18) = {gcd(12, 18)}")  # 6
    print(f"lcm(12, 18) = {lcm(12, 18)}")  # 36
```

==

文件: Code11_Poj2142_TheBalance.cpp

==

```
// POJ 2142 The Balance
```

```cpp
// 给定a、b、c，求解方程ax + by = c
// 要求找到一组解(x,y)，使得|x| + |y|最小
// 如果有多个解，选择x最小的解
// 测试链接：http://poj.org/problem?id=2142

/**
 * POJ 2142 The Balance
 *
 * 问题描述：
 * 给定a、b、c，求解方程ax + by = c
 * 要求找到一组解(x,y)，使得|x| + |y|最小
 * 如果有多个解，选择x最小的解
 *
 * 解题思路：
 * 1. 使用扩展欧几里得算法求解ax + by = gcd(a,b)的一组特解
 * 2. 判断方程是否有解：当c能被gcd(a,b)整除时有解
 * 3. 如果有解，将特解乘以c/gcd(a,b)得到原方程的一组特解
 * 4. 根据通解公式求出满足条件的解
 * 5. 在所有解中寻找|x| + |y|最小的解
 *
 * 数学原理：
 * 1. 裴蜀定理：方程ax + by = c有整数解当且仅当gcd(a,b)能整除c
 * 2. 扩展欧几里得算法：求解ax + by = gcd(a,b)的一组特解
 * 3. 通解公式：如果(x0,y0)是ax + by = c的一组特解，那么通解为：
 *    x = x0 + (b/gcd(a,b)) * t
 *    y = y0 - (a/gcd(a,b)) * t
 *    t为任意整数
 *
 * 时间复杂度：O(log(min(a,b)))，主要消耗在扩展欧几里得算法上
 * 空间复杂度：O(1)
 *
 * 相关题目：
 * 1. POJ 2142 The Balance
 *    链接：http://poj.org/problem?id=2142
 * 2. UVA 10090 Marbles
 *    链接：
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1031
 * 3. Codeforces 7C. Line
 *    链接：https://codeforces.com/problemset/problem/7/C
 */

#include <iostream>
#include <cmath>
```

```
#include <climits>
using namespace std;

/**
 * 扩展欧几里得算法，求解 ax + by = gcd(a,b)的一组特解
 *
 * @param a 第一个系数
 * @param b 第二个系数
 * @param x 引用参数，存储解 x
 * @param y 引用参数，存储解 y
 * @return a 和 b 的最大公约数
 */
long long extendedGcd(long long a, long long b, long long &x, long long &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    long long gcd = extendedGcd(b, a % b, y, x);
    y -= (a / b) * x;
    return gcd;
}

/**
 * 求解方程 ax + by = c，并找到|x| + |y|最小的解
 *
 * @param a 第一个系数
 * @param b 第二个系数
 * @param c 常数项
 * @param x 引用参数，存储解 x
 * @param y 引用参数，存储解 y
 * @return 是否有解
 */
bool solveEquation(long long a, long long b, long long c, long long &x, long long &y) {
    // 特殊情况处理
    if (a == 0 && b == 0) {
        if (c == 0) {
            x = 0;
            y = 0;
            return true;
        } else {
            return false;
        }
```

```
}

// 使用扩展欧几里得算法
long long x0, y0;
long long gcd = extendedGcd(a, b, x0, y0);

// 判断是否有解
if (c % gcd != 0) {
    return false;
}

// 计算原方程的特解
long long factor = c / gcd;
x0 *= factor;
y0 *= factor;

// 通解公式参数
long long k1 = b / gcd;
long long k2 = a / gcd;

// 寻找|x| + |y|最小的解
// 通解：x = x0 + k1 * t, y = y0 - k2 * t
// 我们需要最小化 |x0 + k1*t| + |y0 - k2*t|

// 使用数学方法找到最优的 t 值
// 最优 t 应该在 x0/k1 和 y0/k2 附近
long long t1 = (long long)floor((double)(-x0) / k1);
long long t2 = (long long)ceil((double)y0 / k2);

// 检查几个候选 t 值
long long bestX = 0, bestY = 0;
long long minSum = LLONG_MAX;

// 检查 t1-1, t1, t1+1, t2-1, t2, t2+1
for (long long t = t1 - 1; t <= t1 + 1; t++) {
    long long x_val = x0 + k1 * t;
    long long y_val = y0 - k2 * t;
    long long sum = abs(x_val) + abs(y_val);
    if (sum < minSum || (sum == minSum && x_val < bestX)) {
        minSum = sum;
        bestX = x_val;
        bestY = y_val;
    }
```

```
    }

    for (long long t = t2 - 1; t <= t2 + 1; t++) {
        long long x_val = x0 + k1 * t;
        long long y_val = y0 - k2 * t;
        long long sum = abs(x_val) + abs(y_val);
        if (sum < minSum || (sum == minSum && x_val < bestX)) {
            minSum = sum;
            bestX = x_val;
            bestY = y_val;
        }
    }

    x = bestX;
    y = bestY;
    return true;
}

int main() {
    long long a, b, c;

    while (cin >> a >> b >> c) {
        if (a == 0 && b == 0 && c == 0) {
            break;
        }

        long long x, y;
        if (solveEquation(a, b, c, x, y)) {
            cout << x << " " << y << endl;
        } else {
            cout << "No solution" << endl;
        }
    }

    return 0;
}

/**
 * 测试函数
 */
void test() {
    // 测试用例 1：POJ 2142 示例
    cout << "测试用例 1: a=700, b=300, c=200" << endl;
```

```
    long long x1, y1;
    if (solveEquation(700, 300, 200, x1, y1)) {
        cout << "x=" << x1 << ", y=" << y1 << endl;
        cout << "|x| + |y| = " << (abs(x1) + abs(y1)) << endl;
    }

    // 测试用例 2：简单情况
    cout << "\\n 测试用例 2: a=2, b=3, c=5" << endl;
    long long x2, y2;
    if (solveEquation(2, 3, 5, x2, y2)) {
        cout << "x=" << x2 << ", y=" << y2 << endl;
        cout << "|x| + |y| = " << (abs(x2) + abs(y2)) << endl;
    }

    // 测试用例 3：无解情况
    cout << "\\n 测试用例 3: a=2, b=4, c=1" << endl;
    long long x3, y3;
    if (!solveEquation(2, 4, 1, x3, y3)) {
        cout << "No solution (expected)" << endl;
    }

    // 测试扩展欧几里得算法
    cout << "\\n 测试扩展欧几里得算法:" << endl;
    long long x_gcd, y_gcd;
    long long gcd = extendedGcd(12, 18, x_gcd, y_gcd);
    cout << "12*" << x_gcd << " + 18*" << y_gcd << " = " << gcd << endl;
}
```

===

文件: Code11_Poj2142_TheBalance.java

===

```
package class140;

// POJ 2142 The Balance
// 给定 a、b、c，求解方程 ax + by = c
// 要求找到一组解(x,y)，使得|x| + |y|最小
// 如果有多个解，选择 x 最小的解
// 测试链接：http://poj.org/problem?id=2142

/**
 * POJ 2142 The Balance
 *
```

```
 * 问题描述：
 * 给定 a、b、c，求解方程 ax + by = c
 * 要求找到一组解(x,y)，使得|x| + |y|最小
 * 如果有多个解，选择 x 最小的解
 *
 * 解题思路：
 * 1. 使用扩展欧几里得算法求解 ax + by = gcd(a,b)的一组特解
 * 2. 判断方程是否有解：当 c 能被 gcd(a,b)整除时有解
 * 3. 如果有解，将特解乘以 c/gcd(a,b)得到原方程的一组特解
 * 4. 根据通解公式求出满足条件的解
 * 5. 在所有解中寻找|x| + |y|最小的解
 *
 * 数学原理：
 * 1. 裴蜀定理：方程 ax + by = c 有整数解当且仅当 gcd(a,b)能整除 c
 * 2. 扩展欧几里得算法：求解 ax + by = gcd(a,b)的一组特解
 * 3. 通解公式：如果(x0,y0)是 ax + by = c 的一组特解，那么通解为：
 *    x = x0 + (b/gcd(a,b)) * t
 *    y = y0 - (a/gcd(a,b)) * t
 *    t 为任意整数
 *
 * 时间复杂度：O(log(min(a,b)))，主要消耗在扩展欧几里得算法上
 * 空间复杂度：O(1)
 *
 * 相关题目：
 * 1. POJ 2142 The Balance
 *    链接：http://poj.org/problem?id=2142
 * 2. UVA 10090 Marbles
 *    链接：
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1031
 * 3. Codeforces 7C. Line
 *    链接：https://codeforces.com/problemset/problem/7/C
 */

import java.util.Scanner;

public class Code11_Poj2142_TheBalance {

    /**
     * 扩展欧几里得算法，求解 ax + by = gcd(a,b)的一组特解
     *
     * @param a 第一个系数
     * @param b 第二个系数
     * @return 包含 x, y, gcd 的数组
```

```java
 */
private static long[] extendedGcd(long a, long b) {
    if (b == 0) {
        return new long[]{1, 0, a};
    }
    long[] result = extendedGcd(b, a % b);
    long x = result[0];
    long y = result[1];
    long gcd = result[2];
    return new long[]{y, x - (a / b) * y, gcd};
}

/**
 * 求解方程 ax + by = c，并找到|x| + |y|最小的解
 *
 * @param a 第一个系数
 * @param b 第二个系数
 * @param c 常数项
 * @return 包含 x 和 y 的数组，如果没有解返回 null
 */
public static long[] solveEquation(long a, long b, long c) {
    // 特殊情况处理
    if (a == 0 && b == 0) {
        return c == 0 ? new long[]{0, 0} : null;
    }

    // 使用扩展欧几里得算法
    long[] result = extendedGcd(a, b);
    long x0 = result[0];
    long y0 = result[1];
    long gcd = result[2];

    // 判断是否有解
    if (c % gcd != 0) {
        return null;
    }

    // 计算原方程的特解
    long factor = c / gcd;
    x0 *= factor;
    y0 *= factor;

    // 通解公式参数
```

```
    long k1 = b / gcd;
    long k2 = a / gcd;

    // 寻找|x| + |y|最小的解
    // 通解：x = x0 + k1 * t, y = y0 - k2 * t
    // 我们需要最小化 |x0 + k1*t| + |y0 - k2*t|

    // 使用数学方法找到最优的 t 值
    // 最优 t 应该在 x0/k1 和 y0/k2 附近
    long t1 = (long) Math.floor((double) -x0 / k1);
    long t2 = (long) Math.ceil((double) y0 / k2);

    // 检查几个候选 t 值
    long bestX = 0, bestY = 0;
    long minSum = Long.MAX_VALUE;

    // 检查 t1-1, t1, t1+1, t2-1, t2, t2+1
    for (long t = t1 - 1; t <= t1 + 1; t++) {
        long x = x0 + k1 * t;
        long y = y0 - k2 * t;
        long sum = Math.abs(x) + Math.abs(y);
        if (sum < minSum || (sum == minSum && x < bestX)) {
            minSum = sum;
            bestX = x;
            bestY = y;
        }
    }

    for (long t = t2 - 1; t <= t2 + 1; t++) {
        long x = x0 + k1 * t;
        long y = y0 - k2 * t;
        long sum = Math.abs(x) + Math.abs(y);
        if (sum < minSum || (sum == minSum && x < bestX)) {
            minSum = sum;
            bestX = x;
            bestY = y;
        }
    }

    return new long[]{bestX, bestY};
}

/**
```

```
     * 主函数，用于测试
     */
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        while (true) {
            long a = scanner.nextLong();
            long b = scanner.nextLong();
            long c = scanner.nextLong();

            if (a == 0 && b == 0 && c == 0) {
                break;
            }

            long[] result = solveEquation(a, b, c);
            if (result == null) {
                System.out.println("No solution");
            } else {
                System.out.println(result[0] + " " + result[1]);
            }
        }

        scanner.close();
    }

    /**
     * 测试函数
     */
    public static void test() {
        // 测试用例 1：POJ 2142 示例
        System.out.println("测试用例 1: a=700, b=300, c=200");
        long[] result1 = solveEquation(700, 300, 200);
        if (result1 != null) {
            System.out.println("x=" + result1[0] + ", y=" + result1[1]);
            System.out.println("|x| + |y| = " + (Math.abs(result1[0]) + Math.abs(result1[1])));
        }

        // 测试用例 2：简单情况
        System.out.println("\\n 测试用例 2: a=2, b=3, c=5");
        long[] result2 = solveEquation(2, 3, 5);
        if (result2 != null) {
            System.out.println("x=" + result2[0] + ", y=" + result2[1]);
            System.out.println("|x| + |y| = " + (Math.abs(result2[0]) + Math.abs(result2[1])));
```

```
        }

        // 测试用例3：无解情况
        System.out.println("\\n测试用例3: a=2, b=4, c=1");
        long[] result3 = solveEquation(2, 4, 1);
        if (result3 == null) {
            System.out.println("No solution (expected)");
        }

        // 测试扩展欧几里得算法
        System.out.println("\\n测试扩展欧几里得算法:");
        long[] gcdResult = extendedGcd(12, 18);
        System.out.println("12*" + gcdResult[0] + " + 18*" + gcdResult[1] + " = " +
gcdResult[2]);
    }
}
```

==

文件: Code11_Poj2142_TheBalance.py
==
"""
POJ 2142 The Balance
给定a、b、c，求解方程ax + by = c
要求找到一组解(x,y)，使得|x| + |y|最小
如果有多个解，选择x最小的解
测试链接：http://poj.org/problem?id=2142

POJ 2142 The Balance

问题描述：
给定a、b、c，求解方程ax + by = c
要求找到一组解(x,y)，使得|x| + |y|最小
如果有多个解，选择x最小的解

解题思路：
1. 使用扩展欧几里得算法求解ax + by = gcd(a,b)的一组特解
2. 判断方程是否有解：当c能被gcd(a,b)整除时有解
3. 如果有解，将特解乘以c/gcd(a,b)得到原方程的一组特解
4. 根据通解公式求出满足条件的解
5. 在所有解中寻找|x| + |y|最小的解

数学原理：

```
1. 裴蜀定理：方程 ax + by = c 有整数解当且仅当 gcd(a,b)能整除 c
2. 扩展欧几里得算法：求解 ax + by = gcd(a,b)的一组特解
3. 通解公式：如果(x0,y0)是 ax + by = c 的一组特解，那么通解为：
   x = x0 + (b/gcd(a,b)) * t
   y = y0 - (a/gcd(a,b)) * t
   t 为任意整数

时间复杂度：O(log(min(a,b)))，主要消耗在扩展欧几里得算法上
空间复杂度：O(1)

相关题目：
1. POJ 2142 The Balance
   链接：http://poj.org/problem?id=2142
2. UVA 10090 Marbles
   链接：
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1031
3. Codeforces 7C. Line
   链接：https://codeforces.com/problemset/problem/7/C
"""


def extended_gcd(a: int, b: int) -> tuple:
    """
    扩展欧几里得算法，求解 ax + by = gcd(a,b)的一组特解

    Args:
        a: 第一个系数
        b: 第二个系数

    Returns:
        (x, y, gcd) 的元组
    """
    if b == 0:
        return (1, 0, a)
    x, y, g = extended_gcd(b, a % b)
    return (y, x - (a // b) * y, g)

def solve_equation(a: int, b: int, c: int) -> tuple:
    """
    求解方程 ax + by = c，并找到|x| + |y|最小的解

    Args:
        a: 第一个系数
        b: 第二个系数
```

```
    c: 常数项

Returns:
    (x, y) 的元组，如果没有解返回 None
"""
# 特殊情况处理
if a == 0 and b == 0:
    return (0, 0) if c == 0 else None

# 使用扩展欧几里得算法
x0, y0, gcd_val = extended_gcd(a, b)

# 判断是否有解
if c % gcd_val != 0:
    return None

# 计算原方程的特解
factor = c // gcd_val
x0 *= factor
y0 *= factor

# 通解公式参数
k1 = b // gcd_val
k2 = a // gcd_val

# 寻找|x| + |y|最小的解
# 通解：x = x0 + k1 * t, y = y0 - k2 * t
# 我们需要最小化 |x0 + k1*t| + |y0 - k2*t|

# 使用数学方法找到最优的 t 值
# 最优 t 应该在 x0/k1 和 y0/k2 附近
t1 = int((-x0) / k1)
t2 = int((y0 + k2 - 1) / k2)  # 向上取整

# 检查几个候选 t 值
best_x, best_y = 0, 0
min_sum = float('inf')

# 检查 t1-1, t1, t1+1, t2-1, t2, t2+1
for t in range(t1 - 1, t1 + 2):
    x_val = x0 + k1 * t
    y_val = y0 - k2 * t
    current_sum = abs(x_val) + abs(y_val)
```

```
        if current_sum < min_sum or (current_sum == min_sum and x_val < best_x):
            min_sum = current_sum
            best_x = x_val
            best_y = y_val

    for t in range(t2 - 1, t2 + 2):
        x_val = x0 + k1 * t
        y_val = y0 - k2 * t
        current_sum = abs(x_val) + abs(y_val)
        if current_sum < min_sum or (current_sum == min_sum and x_val < best_x):
            min_sum = current_sum
            best_x = x_val
            best_y = y_val

    return (best_x, best_y)

def main():
    """主函数，用于测试"""
    while True:
        try:
            a, b, c = map(int, input().split())
            if a == 0 and b == 0 and c == 0:
                break

            result = solve_equation(a, b, c)
            if result is None:
                print("No solution")
            else:
                print(f"{result[0]} {result[1]}")
        except EOFError:
            break

if __name__ == "__main__":
    # 测试用例 1：POJ 2142 示例
    print("测试用例 1: a=700, b=300, c=200")
    result1 = solve_equation(700, 300, 200)
    if result1 is not None:
        print(f"x={result1[0]}, y={result1[1]}")
        print(f"|x| + |y| = {abs(result1[0]) + abs(result1[1])}")

    # 测试用例 2：简单情况
    print("\\n 测试用例 2: a=2, b=3, c=5")
    result2 = solve_equation(2, 3, 5)
```

```
    if result2 is not None:
        print(f"x={result2[0]}, y={result2[1]}")
        print(f"|x| + |y| = {abs(result2[0]) + abs(result2[1])}")

    # 测试用例 3：无解情况
    print("\\n测试用例 3: a=2, b=4, c=1")
    result3 = solve_equation(2, 4, 1)
    if result3 is None:
        print("No solution (expected)")

    # 测试扩展欧几里得算法
    print("\\n测试扩展欧几里得算法:")
    x, y, g = extended_gcd(12, 18)
    print(f"12*{x} + 18*{y} = {g}")

    # 运行主函数进行交互式测试
    # main()
```

==

文件: Code12_Codeforces7C_Line.cpp

==

```
// Codeforces 7C. Line
// 给定直线方程 Ax + By + C = 0，求直线上任意一个整数点(x,y)
// 如果不存在整数点，输出-1
// 测试链接：https://codeforces.com/problemset/problem/7C

/**
 * Codeforces 7C. Line
 *
 * 问题描述:
 * 给定直线方程 Ax + By + C = 0，求直线上任意一个整数点(x,y)
 * 如果不存在整数点，输出-1
 *
 * 解题思路:
 * 1. 将直线方程转换为标准形式：Ax + By = -C
 * 2. 使用扩展欧几里得算法求解方程 Ax + By = gcd(A,B)的一组特解
 * 3. 判断方程是否有整数解: 当-C 能被 gcd(A,B)整除时有解
 * 4. 如果有解，将特解乘以(-C)/gcd(A,B)得到原方程的一组特解
 *
 * 数学原理:
 * 1. 裴蜀定理：方程 Ax + By = -C 有整数解当且仅当 gcd(A,B)能整除-C
 * 2. 扩展欧几里得算法：求解 Ax + By = gcd(A,B)的一组特解
```

```
 * 3. 通解公式：如果(x0,y0)是 Ax + By = -C 的一组特解，那么通解为：
 *    x = x0 + (B/gcd(A,B)) * t
 *    y = y0 - (A/gcd(A,B)) * t
 *    t 为任意整数
 *
 * 时间复杂度：O(log(min(A,B)))，主要消耗在扩展欧几里得算法上
 * 空间复杂度：O(1)
 *
 * 相关题目：
 * 1. Codeforces 7C. Line
 *    链接：https://codeforces.com/problemset/problem/7C
 * 2. POJ 2142 The Balance
 *    链接：http://poj.org/problem?id=2142
 * 3. UVA 10090 Marbles
 *    链接：
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1031
 */

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

/**
 * 扩展欧几里得算法，求解 ax + by = gcd(a,b)的一组特解
 *
 * @param a 第一个系数
 * @param b 第二个系数
 * @param x 引用参数，存储解 x
 * @param y 引用参数，存储解 y
 * @return a 和 b 的最大公约数
 */
long long extendedGcd(long long a, long long b, long long &x, long long &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    long long gcd = extendedGcd(b, a % b, y, x);
    y -= (a / b) * x;
    return gcd;
}

/**
```

```
 * 求解直线方程 Ax + By + C = 0 的整数解
 *
 * @param A 系数 A
 * @param B 系数 B
 * @param C 常数 C
 * @param x 引用参数，存储解 x
 * @param y 引用参数，存储解 y
 * @return 是否有整数解
 */
bool solveLineEquation(long long A, long long B, long long C, long long &x, long long &y) {
    // 特殊情况处理
    if (A == 0 && B == 0) {
        if (C == 0) {
            x = 0;
            y = 0;
            return true;
        } else {
            return false;
        }
    }

    // 将方程转换为标准形式：Ax + By = -C
    long long target = -C;

    // 处理 A 或 B 为 0 的情况
    if (A == 0) {
        if (target % B == 0) {
            // 任意 x 都可以，y = -C/B
            x = 0;
            y = target / B;
            return true;
        } else {
            return false;
        }
    }

    if (B == 0) {
        if (target % A == 0) {
            // 任意 y 都可以，x = -C/A
            x = target / A;
            y = 0;
            return true;
        } else {
```

```
        return false;
    }
}

// 使用扩展欧几里得算法
long long x0, y0;
long long gcd = extendedGcd(abs(A), abs(B), x0, y0);

// 判断是否有解
if (target % gcd != 0) {
    return false;
}

// 计算原方程的特解
long long factor = target / gcd;
x0 *= factor;
y0 *= factor;

// 处理 A 或 B 为负数的情况
if (A < 0) {
    x0 = -x0;
}
if (B < 0) {
    y0 = -y0;
}

x = x0;
y = y0;
return true;
}

int main() {
    long long A, B, C;
    cin >> A >> B >> C;

    long long x, y;
    if (solveLineEquation(A, B, C, x, y)) {
        cout << x << " " << y << endl;
    } else {
        cout << -1 << endl;
    }

    return 0;
```

```
}

/**
 * 测试函数
 */
void test() {
    // 测试用例 1：Codeforces 7C 示例
    cout << "测试用例 1: A=2, B=5, C=3" << endl;
    long long x1, y1;
    if (solveLineEquation(2, 5, 3, x1, y1)) {
        cout << "x=" << x1 << ", y=" << y1 << endl;
        // 验证：2*x + 5*y + 3 = 0
        cout << "验证: 2*" << x1 << " + 5*" << y1 << " + 3 = " << (2*x1 + 5*y1 + 3) << endl;
    }

    // 测试用例 2：简单情况
    cout << "\\n 测试用例 2: A=1, B=1, C=1" << endl;
    long long x2, y2;
    if (solveLineEquation(1, 1, 1, x2, y2)) {
        cout << "x=" << x2 << ", y=" << y2 << endl;
        cout << "验证: 1*" << x2 << " + 1*" << y2 << " + 1 = " << (x2 + y2 + 1) << endl;
    }

    // 测试用例 3：无解情况
    cout << "\\n 测试用例 3: A=2, B=4, C=1" << endl;
    long long x3, y3;
    if (!solveLineEquation(2, 4, 1, x3, y3)) {
        cout << "No solution (expected)" << endl;
    }

    // 测试用例 4：A 为 0 的情况
    cout << "\\n 测试用例 4: A=0, B=3, C=6" << endl;
    long long x4, y4;
    if (solveLineEquation(0, 3, 6, x4, y4)) {
        cout << "x=" << x4 << ", y=" << y4 << endl;
        cout << "验证: 0*" << x4 << " + 3*" << y4 << " + 6 = " << (3*y4 + 6) << endl;
    }

    // 测试用例 5：B 为 0 的情况
    cout << "\\n 测试用例 5: A=4, B=0, C=8" << endl;
    long long x5, y5;
    if (solveLineEquation(4, 0, 8, x5, y5)) {
        cout << "x=" << x5 << ", y=" << y5 << endl;
```

```
        cout << "验证: 4*" << x5 << " + 0*" << y5 << " + 8 = " << (4*x5 + 8) << endl;
    }

    // 测试扩展欧几里得算法
    cout << "\\n测试扩展欧几里得算法:" << endl;
    long long x_gcd, y_gcd;
    long long gcd = extendedGcd(12, 18, x_gcd, y_gcd);
    cout << "12*" << x_gcd << " + 18*" << y_gcd << " = " << gcd << endl;
}
```

==

文件: Code12_Codeforces7C_Line.java
==

```
package class140;

// Codeforces 7C. Line
// 给定直线方程Ax + By + C = 0，求直线上任意一个整数点(x,y)
// 如果不存在整数点，输出-1
// 测试链接：https://codeforces.com/problemset/problem/7/C

/**
 * Codeforces 7C. Line
 *
 * 问题描述:
 * 给定直线方程Ax + By + C = 0，求直线上任意一个整数点(x,y)
 * 如果不存在整数点，输出-1
 *
 * 解题思路:
 * 1. 将直线方程转换为标准形式：Ax + By = -C
 * 2. 使用扩展欧几里得算法求解方程Ax + By = gcd(A,B)的一组特解
 * 3. 判断方程是否有整数解：当-C能被gcd(A,B)整除时有解
 * 4. 如果有解，将特解乘以(-C)/gcd(A,B)得到原方程的一组特解
 *
 * 数学原理:
 * 1. 裴蜀定理：方程Ax + By = -C有整数解当且仅当gcd(A,B)能整除-C
 * 2. 扩展欧几里得算法：求解Ax + By = gcd(A,B)的一组特解
 * 3. 通解公式：如果(x0,y0)是Ax + By = -C的一组特解，那么通解为：
 *    x = x0 + (B/gcd(A,B)) * t
 *    y = y0 - (A/gcd(A,B)) * t
 *    t为任意整数
 *
 * 时间复杂度：O(log(min(A,B)))，主要消耗在扩展欧几里得算法上
```

```java
 * 空间复杂度：O(1)
 *
 * 相关题目：
 * 1. Codeforces 7C. Line
 *    链接：https://codeforces.com/problemset/problem/7/C
 * 2. POJ 2142 The Balance
 *    链接：http://poj.org/problem?id=2142
 * 3. UVA 10090 Marbles
 *    链接：
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1031
 */

import java.util.Scanner;

public class Code12_Codeforces7C_Line {

    /**
     * 扩展欧几里得算法，求解ax + by = gcd(a,b)的一组特解
     *
     * @param a 第一个系数
     * @param b 第二个系数
     * @return 包含x, y, gcd的数组
     */
    private static long[] extendedGcd(long a, long b) {
        if (b == 0) {
            return new long[]{1, 0, a};
        }
        long[] result = extendedGcd(b, a % b);
        long x = result[0];
        long y = result[1];
        long gcd = result[2];
        return new long[]{y, x - (a / b) * y, gcd};
    }

    /**
     * 求解直线方程Ax + By + C = 0的整数解
     *
     * @param A 系数A
     * @param B 系数B
     * @param C 常数C
     * @return 包含x和y的数组，如果没有整数解返回null
     */
    public static long[] solveLineEquation(long A, long B, long C) {
```

```
// 特殊情况处理
if (A == 0 && B == 0) {
    return C == 0 ? new long[]{0, 0} : null;
}

// 将方程转换为标准形式：Ax + By = -C
long target = -C;

// 处理 A 或 B 为 0 的情况
if (A == 0) {
    if (target % B == 0) {
        // 任意 x 都可以，y = -C/B
        return new long[]{0, target / B};
    } else {
        return null;
    }
}

if (B == 0) {
    if (target % A == 0) {
        // 任意 y 都可以，x = -C/A
        return new long[]{target / A, 0};
    } else {
        return null;
    }
}

// 使用扩展欧几里得算法
long[] result = extendedGcd(Math.abs(A), Math.abs(B));
long x0 = result[0];
long y0 = result[1];
long gcd = result[2];

// 判断是否有解
if (target % gcd != 0) {
    return null;
}

// 计算原方程的特解
long factor = target / gcd;
x0 *= factor;
y0 *= factor;
```

```java
        // 处理A或B为负数的情况
        if (A < 0) {
            x0 = -x0;
        }
        if (B < 0) {
            y0 = -y0;
        }

        return new long[]{x0, y0};
    }

    /**
     * 主函数，用于测试
     */
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        long A = scanner.nextLong();
        long B = scanner.nextLong();
        long C = scanner.nextLong();

        long[] result = solveLineEquation(A, B, C);
        if (result == null) {
            System.out.println(-1);
        } else {
            System.out.println(result[0] + " " + result[1]);
        }

        scanner.close();
    }

    /**
     * 测试函数
     */
    public static void test() {
        // 测试用例1：Codeforces 7C示例
        System.out.println("测试用例1: A=2, B=5, C=3");
        long[] result1 = solveLineEquation(2, 5, 3);
        if (result1 != null) {
            System.out.println("x=" + result1[0] + ", y=" + result1[1]);
            // 验证：2*x + 5*y + 3 = 0
            System.out.println("验证: 2*" + result1[0] + " + 5*" + result1[1] + " + 3 = " +
                             (2*result1[0] + 5*result1[1] + 3));
```

```java
        }

        // 测试用例 2：简单情况
        System.out.println("\\n 测试用例 2: A=1, B=1, C=1");
        long[] result2 = solveLineEquation(1, 1, 1);
        if (result2 != null) {
            System.out.println("x=" + result2[0] + ", y=" + result2[1]);
            System.out.println("验证: 1*" + result2[0] + " + 1*" + result2[1] + " + 1 = " +
                             (result2[0] + result2[1] + 1));
        }

        // 测试用例 3：无解情况
        System.out.println("\\n 测试用例 3: A=2, B=4, C=1");
        long[] result3 = solveLineEquation(2, 4, 1);
        if (result3 == null) {
            System.out.println("No solution (expected)");
        }

        // 测试用例 4：A 为 0 的情况
        System.out.println("\\n 测试用例 4: A=0, B=3, C=6");
        long[] result4 = solveLineEquation(0, 3, 6);
        if (result4 != null) {
            System.out.println("x=" + result4[0] + ", y=" + result4[1]);
            System.out.println("验证: 0*" + result4[0] + " + 3*" + result4[1] + " + 6 = " +
                             (3*result4[1] + 6));
        }

        // 测试用例 5：B 为 0 的情况
        System.out.println("\\n 测试用例 5: A=4, B=0, C=8");
        long[] result5 = solveLineEquation(4, 0, 8);
        if (result5 != null) {
            System.out.println("x=" + result5[0] + ", y=" + result5[1]);
            System.out.println("验证: 4*" + result5[0] + " + 0*" + result5[1] + " + 8 = " +
                             (4*result5[0] + 8));
        }

        // 测试扩展欧几里得算法
        System.out.println("\\n 测试扩展欧几里得算法:");
        long[] gcdResult = extendedGcd(12, 18);
        System.out.println("12*" + gcdResult[0] + " + 18*" + gcdResult[1] + " = " +
gcdResult[2]);
    }
}
```

==

文件: Code12_Codeforces7C_Line.py
==
"""
Codeforces 7C. Line
给定直线方程 Ax + By + C = 0，求直线上任意一个整数点(x,y)
如果不存在整数点，输出-1
测试链接：https://codeforces.com/problemset/problem/7/C

Codeforces 7C. Line

问题描述：
给定直线方程 Ax + By + C = 0，求直线上任意一个整数点(x,y)
如果不存在整数点，输出-1

解题思路：
1. 将直线方程转换为标准形式：Ax + By = -C
2. 使用扩展欧几里得算法求解方程 Ax + By = gcd(A,B)的一组特解
3. 判断方程是否有整数解：当-C 能被 gcd(A,B)整除时有解
4. 如果有解，将特解乘以(-C)/gcd(A,B)得到原方程的一组特解

数学原理：
1. 裴蜀定理：方程 Ax + By = -C 有整数解当且仅当 gcd(A,B)能整除-C
2. 扩展欧几里得算法：求解 Ax + By = gcd(A,B)的一组特解
3. 通解公式：如果(x0,y0)是 Ax + By = -C 的一组特解，那么通解为：
   x = x0 + (B/gcd(A,B)) * t
   y = y0 - (A/gcd(A,B)) * t
   t 为任意整数

时间复杂度：O(log(min(A,B)))，主要消耗在扩展欧几里得算法上
空间复杂度：O(1)

相关题目：
1. Codeforces 7C. Line
   链接：https://codeforces.com/problemset/problem/7C
2. POJ 2142 The Balance
   链接：http://poj.org/problem?id=2142
3. UVA 10090 Marbles
   链接：
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1031
"""

```python
def extended_gcd(a: int, b: int) -> tuple:
    """
    扩展欧几里得算法，求解ax + by = gcd(a,b)的一组特解

    Args:
        a: 第一个系数
        b: 第二个系数

    Returns:
        (x, y, gcd) 的元组
    """
    if b == 0:
        return (1, 0, a)
    x, y, g = extended_gcd(b, a % b)
    return (y, x - (a // b) * y, g)

def solve_line_equation(A: int, B: int, C: int) -> tuple:
    """
    求解直线方程Ax + By + C = 0的整数解

    Args:
        A: 系数A
        B: 系数B
        C: 常数C

    Returns:
        (x, y) 的元组，如果没有整数解返回None
    """
    # 特殊情况处理
    if A == 0 and B == 0:
        return (0, 0) if C == 0 else None

    # 将方程转换为标准形式：Ax + By = -C
    target = -C

    # 处理A或B为0的情况
    if A == 0:
        if target % B == 0:
            # 任意x都可以，y = -C/B
            return (0, target // B)
        else:
            return None
```

```
    if B == 0:
        if target % A == 0:
            # 任意 y 都可以，x = -C/A
            return (target // A, 0)
        else:
            return None

    # 使用扩展欧几里得算法
    x0, y0, gcd_val = extended_gcd(abs(A), abs(B))

    # 判断是否有解
    if target % gcd_val != 0:
        return None

    # 计算原方程的特解
    factor = target // gcd_val
    x0 *= factor
    y0 *= factor

    # 处理 A 或 B 为负数的情况
    if A < 0:
        x0 = -x0
    if B < 0:
        y0 = -y0

    return (x0, y0)

def main():
    """主函数，用于测试"""
    A, B, C = map(int, input().split())

    result = solve_line_equation(A, B, C)
    if result is None:
        print(-1)
    else:
        print(f"{result[0]} {result[1]}")

if __name__ == "__main__":
    # 测试用例 1：Codeforces 7C 示例
    print("测试用例 1: A=2, B=5, C=3")
    result1 = solve_line_equation(2, 5, 3)
    if result1 is not None:
```

```
        print(f"x={result1[0]}, y={result1[1]}")
        # 验证：2*x + 5*y + 3 = 0
        print(f"验证: 2*{result1[0]} + 5*{result1[1]} + 3 = {2*result1[0] + 5*result1[1] + 3}")

    # 测试用例 2：简单情况
    print("\\n 测试用例 2: A=1, B=1, C=1")
    result2 = solve_line_equation(1, 1, 1)
    if result2 is not None:
        print(f"x={result2[0]}, y={result2[1]}")
        print(f"验证: 1*{result2[0]} + 1*{result2[1]} + 1 = {result2[0] + result2[1] + 1}")

    # 测试用例 3：无解情况
    print("\\n 测试用例 3: A=2, B=4, C=1")
    result3 = solve_line_equation(2, 4, 1)
    if result3 is None:
        print("No solution (expected)")

    # 测试用例 4：A 为 0 的情况
    print("\\n 测试用例 4: A=0, B=3, C=6")
    result4 = solve_line_equation(0, 3, 6)
    if result4 is not None:
        print(f"x={result4[0]}, y={result4[1]}")
        print(f"验证: 0*{result4[0]} + 3*{result4[1]} + 6 = {3*result4[1] + 6}")

    # 测试用例 5：B 为 0 的情况
    print("\\n 测试用例 5: A=4, B=0, C=8")
    result5 = solve_line_equation(4, 0, 8)
    if result5 is not None:
        print(f"x={result5[0]}, y={result5[1]}")
        print(f"验证: 4*{result5[0]} + 0*{result5[1]} + 8 = {4*result5[0] + 8}")

    # 测试扩展欧几里得算法
    print("\\n 测试扩展欧几里得算法:")
    x, y, g = extended_gcd(12, 18)
    print(f"12*{x} + 18*{y} = {g}")

    # 运行主函数进行交互式测试
    # main()
```

==

文件: Code13_Uva10090_Marbles.java

==

```
package class140;

// UVA 10090 Marbles
// 有两种盒子：第一种盒子可以装 n1 个弹珠，价格为 c1；第二种盒子可以装 n2 个弹珠，价格为 c2
// 需要装恰好 n 个弹珠，求最小总价格
// 如果无法恰好装 n 个弹珠，输出"failed"
// 测试链接：
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1031

/**
 * UVA 10090 Marbles
 *
 * 问题描述：
 * 有两种盒子：第一种盒子可以装 n1 个弹珠，价格为 c1；第二种盒子可以装 n2 个弹珠，价格为 c2
 * 需要装恰好 n 个弹珠，求最小总价格
 * 如果无法恰好装 n 个弹珠，输出"failed"
 *
 * 解题思路：
 * 1. 设第一种盒子用 x 个，第二种盒子用 y 个，则方程为：n1*x + n2*y = n
 * 2. 使用扩展欧几里得算法求解方程
 * 3. 判断方程是否有解：当 n 能被 gcd(n1,n2)整除时有解
 * 4. 如果有解，根据通解公式求出所有可能的解
 * 5. 在所有解中寻找 c1*x + c2*y 最小的解
 *
 * 数学原理：
 * 1. 裴蜀定理：方程 n1*x + n2*y = n 有整数解当且仅当 gcd(n1,n2)能整除 n
 * 2. 扩展欧几里得算法：求解 n1*x + n2*y = gcd(n1,n2)的一组特解
 * 3. 通解公式：如果(x0,y0)是 n1*x + n2*y = n 的一组特解，那么通解为：
 *    x = x0 + (n2/gcd(n1,n2)) * t
 *    y = y0 - (n1/gcd(n1,n2)) * t
 *    t 为任意整数
 *
 * 时间复杂度：O(log(min(n1,n2)))，主要消耗在扩展欧几里得算法上
 * 空间复杂度：O(1)
 *
 * 相关题目：
 * 1. UVA 10090 Marbles
 *    链接：
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1031
 * 2. POJ 2142 The Balance
 *    链接：http://poj.org/problem?id=2142
 * 3. Codeforces 7C. Line
 *    链接：https://codeforces.com/problemset/problem/7/C
```

```java
 */

import java.util.Scanner;

public class Code13_Uva10090_Marbles {

    /**
     * 扩展欧几里得算法，求解ax + by = gcd(a,b)的一组特解
     *
     * @param a 第一个系数
     * @param b 第二个系数
     * @return 包含x, y, gcd的数组
     */
    private static long[] extendedGcd(long a, long b) {
        if (b == 0) {
            return new long[]{1, 0, a};
        }
        long[] result = extendedGcd(b, a % b);
        long x = result[0];
        long y = result[1];
        long gcd = result[2];
        return new long[]{y, x - (a / b) * y, gcd};
    }

    /**
     * 求解弹珠问题
     *
     * @param n 总弹珠数
     * @param n1 第一种盒子的容量
     * @param c1 第一种盒子的价格
     * @param n2 第二种盒子的容量
     * @param c2 第二种盒子的价格
     * @return 包含x和y的数组，如果没有解返回null
     */
    public static long[] solveMarbles(long n, long n1, long c1, long n2, long c2) {
        // 特殊情况处理
        if (n1 == 0 && n2 == 0) {
            return n == 0 ? new long[]{0, 0} : null;
        }

        // 使用扩展欧几里得算法
        long[] result = extendedGcd(n1, n2);
        long x0 = result[0];
```

```
long y0 = result[1];
long gcd = result[2];

// 判断是否有解
if (n % gcd != 0) {
    return null;
}

// 计算原方程的特解
long factor = n / gcd;
x0 *= factor;
y0 *= factor;

// 通解公式参数
long k1 = n2 / gcd;
long k2 = n1 / gcd;

// 通解：x = x0 + k1 * t, y = y0 - k2 * t
// 我们需要 x >= 0, y >= 0，且最小化 c1*x + c2*y

// 计算 t 的范围
// x >= 0 => x0 + k1*t >= 0 => t >= -x0/k1
// y >= 0 => y0 - k2*t >= 0 => t <= y0/k2
long tMin = (long) Math.ceil((double) -x0 / k1);
long tMax = (long) Math.floor((double) y0 / k2);

if (tMin > tMax) {
    return null; // 没有非负整数解
}

// 目标函数：cost = c1*x + c2*y = c1*(x0 + k1*t) + c2*(y0 - k2*t)
// = (c1*x0 + c2*y0) + (c1*k1 - c2*k2)*t

// 如果 c1*k1 - c2*k2 > 0，则 cost 随 t 增加而增加，最小值在 tMin 处
// 如果 c1*k1 - c2*k2 < 0，则 cost 随 t 增加而减少，最小值在 tMax 处
// 如果 c1*k1 - c2*k2 == 0，则 cost 为常数

long bestT;
long coefficient = c1 * k1 - c2 * k2;

if (coefficient > 0) {
    bestT = tMin;
} else if (coefficient < 0) {
```

```
        bestT = tMax;
    } else {
        // 系数为 0，任意 t 都可以，我们选择 tMin
        bestT = tMin;
    }

    long x = x0 + k1 * bestT;
    long y = y0 - k2 * bestT;

    // 验证解是否非负
    if (x < 0 || y < 0) {
        return null;
    }

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

/**
 * 主函数，用于测试
 */
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    while (true) {
        long n = scanner.nextLong();
        if (n == 0) {
            break;
        }

        long c1 = scanner.nextLong();
        long n1 = scanner.nextLong();
        long c2 = scanner.nextLong();
        long n2 = scanner.nextLong();

        long[] result = solveMarbles(n, n1, c1, n2, c2);
        if (result == null) {
            System.out.println("failed");
        } else {
            System.out.println(result[0] + " " + result[1]);
        }
    }

    scanner.close();
```

```java
    }

    /**
     * 测试函数
     */
    public static void test() {
        // 测试用例 1：UVA 10090 示例
        System.out.println("测试用例 1: n=100, n1=3, c1=5, n2=7, c2=8");
        long[] result1 = solveMarbles(100, 3, 5, 7, 8);
        if (result1 != null) {
            System.out.println("x=" + result1[0] + ", y=" + result1[1]);
            System.out.println("总价格: " + (5*result1[0] + 8*result1[1]));
            System.out.println("验证: 3*" + result1[0] + " + 7*" + result1[1] + " = " +
                             (3*result1[0] + 7*result1[1]));
        }

        // 测试用例 2：简单情况
        System.out.println("\\n 测试用例 2: n=10, n1=2, c1=3, n2=3, c2=4");
        long[] result2 = solveMarbles(10, 2, 3, 3, 4);
        if (result2 != null) {
            System.out.println("x=" + result2[0] + ", y=" + result2[1]);
            System.out.println("总价格: " + (3*result2[0] + 4*result2[1]));
        }

        // 测试用例 3：无解情况
        System.out.println("\\n 测试用例 3: n=1, n1=2, c1=3, n2=4, c2=5");
        long[] result3 = solveMarbles(1, 2, 3, 4, 5);
        if (result3 == null) {
            System.out.println("No solution (expected)");
        }

        // 测试扩展欧几里得算法
        System.out.println("\\n 测试扩展欧几里得算法:");
        long[] gcdResult = extendedGcd(12, 18);
        System.out.println("12*" + gcdResult[0] + " + 18*" + gcdResult[1] + " = " +
gcdResult[2]);
    }
}
```

===