Задача A. Симфония будильников

Будильники звонят в моменты, кратные a, b и c соответственно, начиная с $t = 0$. Требуется найти количество моментов $t \in [0, T]$, в которые звонят **хотя бы два** будильника.

Основная идея.

Момент t является «аккордом», если он делится хотя бы на два из чисел a, b, c. Это эквивалентно тому, что t кратен $\mathrm{lcm}(a, b)$, или $\mathrm{lcm}(a, c)$, или $\mathrm{lcm}(b, c)$. Применяя принцип включения-исключения, получаем формулу:

$$\text{ответ} = N_{ab} + N_{ac} + N_{bc} - 2 \cdot N_{abc},$$

где:

- $N_{ab} = \left\lfloor \dfrac{T}{\mathrm{lcm}(a, b)} \right\rfloor + 1$ — число моментов, когда звонят первый и второй будильники;
- N_{ac}, N_{bc} определяются аналогично;
- $N_{abc} = \left\lfloor \dfrac{T}{\mathrm{lcm}(a, b, c)} \right\rfloor + 1$ — число моментов, когда звонят все три будильника.

Множитель 2 возникает потому, что моменты, в которые звонят все три будильника, учитываются в каждом из трёх парных слагаемых (итого трижды), но должны быть учтены ровно один раз.

Подзадачи.

- **Группа A ($T \leqslant 2 \cdot 10^5$).** Можно перебрать все t от 0 до T и проверить, сколько будильников звонят в момент t. Сложность $O(T)$.
- **Группа B.** Требуется применять формулу с lcm и gcd.Сложность $O(log_2(max(a, b, c)))$.

Пример реализации на Python.

```
import math

a = int(input())
b = int(input())
c = int(input())
T = int(input())

def lcm(x, y):
    return x // math.gcd(x, y) * y

lab = lcm(a, b)
lac = lcm(a, c)
lbc = lcm(b, c)
labc = lcm(lab, c)

x_ab = T // lab + 1
x_ac = T // lac + 1
x_bc = T // lbc + 1
x_abc = T // labc + 1

ans = x_ab + x_ac + x_bc - 2 * x_abc
print(ans)
```

Задача B. Кто больше?

В игре Крош (игрок 1) и Ёжик (игрок 2) по очереди выкладывают карточки: Крош — нечётное количество $(1, 3, 5, \dots)$, Ёжик — чётное $(2, 4, 6, \dots)$. Проигрывает тот, кто не может сделать свой ход. Требуется определить победителя и остатки карточек.

Ключевые наблюдения. Пусть k — количество полных ходов, сделанных каждым игроком. Тогда:

- Крош израсходовал $1 + 3 + \cdots + (2k - 1) = k^2$ карточек;
- Ёжик израсходовал $2 + 4 + \cdots + 2k = k(k + 1)$ карточек.

Максимальное возможное k ограничено запасами игроков: $k^2 \leqslant n$ и $k(k + 1) \leqslant m$.

Обозначим:

- $k_1 = \lfloor \sqrt{n} \rfloor$ — наибольшее k, при котором Крош может сделать k ходов;
- k_2 — наибольшее k, при котором Ёжик может сделать k ходов. Его можно найти как наибольшее целое, для которого $k(k + 1) \leqslant m$.

Логика определения победителя.

- Если $k_1 > k_2$, то после k_2 ходов каждого Ёжик больше не может ходить, но Крош может сделать следующий $(k_2 + 1)$-й ход. Значит, Ёжик проигрывает, побеждает Крош (1).
- Иначе ($k_1 \leqslant k_2$), после k_1 ходов Крош не может сделать $(k_1 + 1)$-й ход, поэтому проигрывает Крош, побеждает Ёжик (2).

Остатки карточек.

- Если победил Крош: у него осталось $n - (k_2 + 1)^2$, у Ёжика — $m - k_2(k_2 + 1)$.
- Если победил Ёжик: у Кроша осталось $n - k_1^2$, у Ёжика — $m - k_1(k_1 + 1)$.

Тонкие моменты.

- Для вычисления k_2 можно использовать приближение $k_2 \approx \lfloor \sqrt{m + 1} \rfloor$, а затем скорректировать: если $k_2(k_2 + 1) > m$, уменьшить k_2 на 1.
- Случай $n = 0$: тогда $k_1 = 0$, $k_1 \leqslant k_2$, побеждает Ёжик, что корректно (Крош не может сделать первый ход).
- Все вычисления выполняются за $O(1)$ и корректны даже при $n, m \leqslant 10^{18}$ благодаря использованию целочисленного корня.

Подзадачи.

- **Группа A ($n, m \leqslant 2 \cdot 10^9$).** Простое моделирование игры циклом дает сложность $O(\sqrt{N})$, где $N = \min(n, m)$ и набирает 70 баллов.
- **Группа B ($n, m \leqslant 10^{18}$).** На полный балл необходимо написать решение через формулу (сложность $O(1)$) либо бинарный поиск по ответу (сложность $O(N \log_2 N)$).

Пример реализации на Python.

```
from math import isqrt

n = int(input())
m = int(input())

k1 = isqrt(n)
k2 = isqrt(m + 1)
if k2 * (k2 + 1) > m:
    k2 -= 1

if k1 > k2:
```

```
    print(1)
    print(n - (k2 + 1) ** 2, m - k2 * (k2 + 1))
else:
    print(2)
    print(n - k1 * k1, m - k1 * (k1 + 1))
```

Задача C. Турнир Смешариков

Необходимо вывести все $\frac{N(N-1)}{2}$ пар игроков ровно по одному разу так, чтобы в любых двух соседних матчах не было общих участников.

Ключевое наблюдение.

Такая последовательность существует **для всех N, кроме $N = 3$ и $N = 4$.** Для $N = 2$ решение тривиально: один матч. Для $N = 3$ всего 3 матча, но каждый игрок участвует в двух матчах, и в цепочке из 3 матчей обязательно найдутся соседние с общим игроком. Для $N = 4$ — 6 матчей, но максимальная длина пути без повторяющихся вершин в графе полного графа K_4 ограничена, и построить требуемую последовательность невозможно.

Идея построения для $N \geqslant 5$.

Рассмотрим игроков, расположенных по кругу: $1, 2, \ldots, N$. Будем перебирать все возможные «расстояния» $d = 1, 2, \ldots, \lfloor N/2 \rfloor$. Для каждого фиксированного d пройдём по кругу и добавим все пары вида $(i, i + d \bmod N)$ (с коррекцией на 1-индексацию), пропуская уже использованные пары. При переходе от одного расстояния к другому следим, чтобы первый матч нового блока не содержал игроков из последнего матча предыдущего блока — при $N \geqslant 5$ это всегда можно обеспечить за счёт достаточного количества свободных вершин.

Альтернативный подход — жадное построение: на каждом шаге выбирать любой ещё не сыгранный матч, не содержащий игроков из предыдущего матча. При $N \geqslant 5$ такой жадный алгоритм всегда находит решение.

Сложность.

Построение выполняется за $O(N^2)$, что достаточно при $N \leqslant 100$.

Реализация.

Если $N = 3$ или $N = 4$, выводим -1. Иначе генерируем все пары, соблюдая условие на соседние матчи (например, методом круговых сдвигов или жадно).

Задача D. НОД-свёртка

Даны три отрезка целых неотрицательных чисел:

$$A \in [L_A, R_A], \quad B \in [L_B, R_B], \quad C \in [L_C, R_C].$$

Требуется найти количество троек (A, B, C), для которых выполняется равенство

$$\gcd(A + B,\ B + C) = A + C.$$

Идея: Найдём все тройки (A, B, C), такие что $\gcd(A+B, B+C) = A+C$. Для этого рассмотрим ключевые случаи при натуральных и нулевых значениях.

1. Перебор (1 группа, $N \leqslant 100$):

Здесь можно просто для каждой тройки (A, B, C) перебрать все значения из отрезков и посчитать количество подходящих троек, используя стандартную функцию вычисления НОД. Такой перебор сработает благодаря малому диапазону.

2. Только совпадения ($L_A, L_B, L_C \geqslant 1$):

Заметим, что если все числа положительные, уравнение выполняется только в случае $A = B = C$ (иначе НОД не равен сумме двух других). Тогда количество подходящих троек — это количество чисел, лежащих во всех трёх отрезках:

$$\max(0,\ \min(R_A, R_B, R_C) - \max(L_A, L_B, L_C) + 1)$$

3. Общий случай:

Рассмотрим три случая:

- Если $A = C$, то условие становится $A + B = B + C \Leftrightarrow A = B = C$. Эти тройки уже учтены выше.
- Если $A = 0$, $\gcd(B, B + C) = \gcd(C, B) = C$ — это возможно только если C делит B, и $C \geqslant 1$, либо при $B = 0$.
- Аналогично, если $C = 0$, $\gcd(A + B, B) = \gcd(A, B) = A$ — это возможно только если A делит B, и $A \geqslant 1$, либо при $B = 0$.

Полный ответ состоит из суммы:

1. Количество t, где $A = B = C = t$ внутри всех трёх отрезков;
2. Для каждого $C \geqslant 1$, $L_C \leqslant C \leqslant R_C$, количество B, делящихся на C внутри $[L_B, R_B]$,при $A = 0$;
3. Для каждого $A \geqslant 1$, $L_A \leqslant A \leqslant R_A$, количество B, делящихся на A внутри $[L_B, R_B]$,при $C = 0$;
4. Длина интервала $[max(1, LC); RC]$ при $B = A = 0$;
5. Длина интервала $[max(1, LA); RA]$ при $B = C = 0$;

Задача E. Игра Кроша

Крош последовательно побеждает монстров, применяя Импульсный Луч: при ударе по монстру i он наносит урон $1, 2, \ldots, k$ монстрам $i, i + 1, \ldots, i + k - 1$. Здоровье обновляется как $h \to |h - d|$, и монстр исчезает при $h = 0$.

Ключевая идея.

Влияние прошлых ударов на текущего монстра i можно описать массивом $\texttt{cnt}[d]$, где $\texttt{cnt}[d]$ — количество ударов величины d ($2 \leqslant d \leqslant k$), которые монстр i получит от предыдущих позиций. При переходе к следующему монстру массив сдвигается: $\texttt{cnt}[d] \to \texttt{cnt}[d - 1]$, а $\texttt{cnt}[k]$ отбрасывается.

Для текущего здоровья $x = h_i$ последовательно обрабатываем $d = 2, 3, \ldots, k$:

- Если $x \geqslant d$, то первые $\min(\texttt{cnt}[d], \lfloor x/d \rfloor)$ ударов уменьшают x на d;
- Оставшиеся удары (если $x < d$) меняют x по правилу «пинг-понга»: при нечётном количестве $x \to d - x$, при четном x не меняется.

Если после обработки всех d осталось $x > 0$, нужно нанести x ударов величины 1 по текущему монстру. Эти x ударов добавляются к $\texttt{cnt}[2]$ для следующего монстра.

Сложность.

Алгоритм работает за $O(n \cdot k)$ времени и $O(k)$ памяти. При $n \leqslant 2 \cdot 10^5$, $k \leqslant 200$ это укладывается в ограничения.

Частичные решения.

- При $k = 1$ ответ — сумма всех h_i.
- При малых n можно просто смоделировать игровой процесс.