Задача A. Проверка на простоту

Задача была разобрана на лекции. Код для решения.

```
for (int i = 2; i * i <= n; ++i) {
    if (n % i == 0) {
        cout << "composite";
        return 0;
    }
}
cout << "prime";
```

Задача B. Количество делителей

Задача была разобрана на лекции. Код для решения.

```
for (int i = 1; i * i <= x; ++i) {
    if (x % i == 0) {
        cnt++;
        if (i != x / i) {
            cnt++;
        }
    }
}
```

Задача C. Разложение на простые

Задача была разобрана на лекции. Код для решения.

```
for (int i = 2; i * i <= n; i++) {
    while (n % i == 0) {
        n /= i;
        cout << i << " ";
    }
}
if (n > 1) {
    cout << n;
}
cout << "prime";
```

Задача D. Обработка заявлений

Для $n \leqslant 1000$ или $n \leqslant 10^5$ задача решается простым моделированием сложности $O(n^2)$ и $O(n)$ соответственно. Нужно создать список из всех заявлений, затем в цикле с первым заявлением из списка выполнять одну из трёх операций. Отличие решений по сложности заключается в том, как реализовать удаление первого элемента из списка. Если в языке Python для этого использовать метод `pop(0)`, то одно такое удаление будет выполняться за $O(n)$, а общая сложность будет $O(n^2)$. Для уменьшения сложности до $O(n)$ нужно использовать структуру данных «очередь» или «дек» или реализовать «ленивое удаление», то есть не удалять элемент, а просто увеличивать на 1 индекс элемента, который является первым в списке.

Пример решения сложности $O(n)$ с «ленивым удалением».

```
n = int(input())
k = int(input())
a = [i + 1 for i in range(n)]
i = 0
while i < len(a):
    if a[i] == k and i % 3 == 0:
```

```
        print("Yes")
        print(i + 1)
        break
    elif a[i] == k and i % 3 == 1:
        print("No")
        print(i + 1)
        break
    elif i % 3 == 2:
        a.append(a[i])
    i += 1
```

Идея полного решения заключается в том, что примерно для $n/3$ заявлений мы сразу же можем дать ответ, что данное заявление будет подписано (и это случится на k-м шаге), ещё примерно $n/3$ заявлений будет отброшено, и примерно $n/3$ заявлений будет переложено, в этом случае нужно решить задачу для нового n, уменьшенного в три раза. Причём «моделирование» обработки всей стопки заявлений можно делать за $O(1)$ операций. Нужно только аккуратно разобраться, как происходит сведение задачи от n к $n/3$.

Рассмотрим задачу в более общем виде — дана тройка (n, k, op), где:

- n — количество заявлений,
- k — порядковый номер искомого заявления,
- op — какую *последнюю* операцию мы выполнили над заявлением (0 — переложить вниз стопки, 1 — подписать, 2 — отбросить). Эти операции повторяются по циклу: 1, 2, 0, 1, 2, 0 и т.д. Поскольку мы начинаем обрабатывать заявления с операции 1, то можно считать, что последней выполненной операцией до этого была операция 0.

Определим, какая операция будет производиться над k-м по порядку заявлением. Это $(op + k) \bmod 3$ (если последней была выполнена операция op, то с первым заявлением в стопке будет выполнена операция $op + 1$, затем — $op + 2$ и т.д. с учётом зацикливания). Если значение $(op+k) \bmod 3$ равно 1 или 2, то добавляем к ответу k, выводим ответ и завершаем программу. Если же $(op+k) \bmod 3 = 0$, то добавим n к количеству шагов и перейдём к такой же точно задаче, только меньшего размера. Для этого нужно определить новые значения n, k и op.

Вычислим новое n — то есть сколько заявлений переместится вниз стопки.

Если $op = 0$, то переместятся $\lfloor n/3 \rfloor$ заявлений — мы перекладываем каждое третье заявление, то есть нам нужно найти количество нулей в последовательности 1, 2, 0, 1, 2, 0, ..., из n элементов.

Если $op = 1$, то нам также достаточно посчитать количество нулей в последовательности 2, 0, 1, 2, 0, 1, ..., это такая же последовательность, но сдвинутая на 1, поэтому ответ будет равен $\lfloor (n+1)/3 \rfloor$.

Если $op = 2$, то тогда нужно посчитать количество нулей в последовательности 0, 1, 2, 0, 1, 2, ..., это такая же последовательность, но сдвинутая на 2, поэтому ответ будет равен $\lfloor (n+2)/3 \rfloor$.

Заметим, что все три варианта можно записать одной формулой: $n' = (n + op)//3$.

Вычислим новое k — то есть каким по порядку будет искомое заявление после рассмотрения всех заявлений и перекладывания части из них вниз стопки. Другими словами, нужно найти количество заявлений с порядковыми номерами до k (включительно), которые переместятся вниз. Это значение также зависит от op (с какого заявления началась обработка). По аналогии с предыдущими рассуждениями получаем: $k' = (k + op)//3$.

Наконец, вычислим новое op — то есть какая операция была выполнена над последним обработанным заявлением: $op' = (op + n) \bmod 3$.

В итоге мы пришли к аналогичной задаче с параметрами (n', k', op'), для решения которой снова повторяем вышеописанные действия.

Поскольку каждый раз остаётся примерно треть заявлений от предыдущего количества, то сложность решения $O(\log n)$.

Пример решения сложности $O(\log n)$.

```
BOTTOM, SIGN, DROP = 0, 1, 2
n = int(input())
k = int(input())
op = BOTTOM
steps = 0
while (op + k) % 3 == BOTTOM:
    steps += n
    n, k, op = (n + op) // 3, (k + op) // 3, (n + op) % 3
print("Yes" if (op + k) % 3 == SIGN else "No")
steps += k
print(steps)
```

Задача E. Осторожно, злые числа!

Для решения подзадачи $n \leqslant 10^5$ переберём все числа от 1 до n и проверим выполнение двух условий (чётность и количество единиц в двоичной записи). Если оба условия выполнены – увеличим ответ на 1.

```
n = int(input())
ans = 0
for i in range(1, n + 1):
    if i % 2 == 0 and bin(i).count('1') % 2 == 0:
        ans += 1
print(ans)
```

Для решения второй подзадачи, когда n – степень двойки, можно заметить (например, проверив маленькие степени двойки), что между 2^p и $2^{p+1} - 1$ у нас получается ровно 2^{p-2} очень злых чисел.

Этот факт доказуем. Представим все числа из интервала $[2^p, 2^{p+1} - 1]$ в двоичной системе счисления. Все они имеют одинаковую длину $p + 1$, у всех чисел первая цифра 1, у всех очень злых чисел последняя цифра будет 0. Тогда мы должны найти количество способов расставить нечетное количество единиц (одна уже стоит в самом начале числа) среди $p - 1$ позиций. Возникает комбинаторная формула суммы числа сочетаний $\sum_{i=1}^{p-1} C_{p-1}^{i}$, где i принимает все нечётные значения, не превышающие $p - 1$.

По свойствам сумм числа сочетаний биномиальных коэффициентов и их знакопеременной сумме можно доказать, что это число равно 2^{p-2}.

Заметим, что само число $n = 2^p$ злым никогда не будет (в его двоичном представлении одна единица на первой позиции). Для ответа на вопрос второй подзадачи задачи для $n = 2^p$ нам останется сложить все степени двойки до $p-3$ включительно. Эта сумма равна $2^0+2^1+2^2+...+2^{p-3} = 2^{p-2}-1$.

Пример решения в этом случае.

```
n = int(input())
ans = 0
p = 0
while n % 2 == 0:
    n //= 2
    p += 1
if n == 1:
    ans = 2 ** (p - 2) - 1
print(ans)
```

Полное решение: рассмотрим четвёрки последовательных целых чисел: (0, 1, 2, 3), (4, 5, 6, 7), (8, 9, 10, 11) и т.д. Числа в этой четвёрке в двоичной системе счисления оканчиваются цифрами 00, 01, 10 и 11. Из них два нечётных числа не подходят, а два оставшихся чётных числа в двоичной записи

отличаются ровно одной предпоследней цифрой, поэтому среди этих чисел ровно одно будет очень злым.

Исключением является только первая четвёрка, в которой чётными числом, содержащим чётное число единиц в двоичной записи, является число 0, однако, оно не подходит, потому что очень злое число должно быть положительным.

Таким образом, для получения ответа нужно посчитать полное число четвёрок в числе n, кроме первой четвёрки, а оставшиеся числа, которые не попали в полную четвёрку, перебрать и для каждого из них непосредственно проверить нужные условия.

Пример решения на языке Python.

```
n = int(input())
ans = (n - 4) // 4
for i in range(n // 4 * 4, n + 1):
    if i % 2 == 0 and bin(i).count('1') % 2 == 0:
        ans += 1
print(ans)
```

Задача F. Расклейка афиш

Первую подзадачу ($n \leqslant 10^5$) можно решить с использованием перебора, например, так:

```
n = int(input())
a = int(input())
b = int(input())
ans = 0
for i in range(1, n + 1):
    if i % a == 0 or i % b == 0:
        ans += 1
print(ans)
```

Вторую подзадачу ($a = 2$) можно решить с использованием разбора двух случаев.

Если b тоже чётное, то при втором проходе Воробьянинов не расклеит новых афиш.

Если b — нечётное, то каждый второй дом, на который нужно наклеить афишу на втором проходе, уже будет иметь афишу (так как сумма двух нечётных чисел всегда чётна). Чтобы найти число подходящих номеров поделим n на b, а потом полученное число поделим на 2 с округлением вверх.

```
ans = n // 2
if b % 2:
    ans += (n // b + 1) // 2
print(ans)
```

Для решения задачи на полный балл нужно сложить количество афиш, наклеенных Воробьяниновым при первом проходе, и количество афиш, которые он наклеит при втором проходе так, если бы первого прохода не было. Мы дважды посчитали дома, номера которых делятся нацело и на a, и на b.

Посчитаем количество номеров домов, которые делятся и на a, и на b. Первое такое числоа должно делиться нацело и на a и на b и быть наименьшим возможным. Согласно определению, это наименьшее общее кратное a и b. Его можно найти через каноническое разложение обоих чисел на простые множители или через наибольший общий делитель (который в свою очередь находится с помощью алгоритма Евклида):

$$\text{НОК}(a, b) = \frac{a \times b}{\text{НОД}(a, b)}$$

Итоговое количество вычитаемых чисел составит n // (a * b // НОД(a, b))

```python
def gcd(n, m):
    if m == 0:
        return n
    return gcd(m, n % m)

n = int(input())
a = int(input())
b = int(input())
ans = n // a + n // b - n // (a * b // gcd(a, b))
print(ans)
```

Задача G. НОД и НОК

Ответа нет, если $\% \neq 0$.

Как ответ можно взять сами числа x, y. Тогда $(x, y) = x$ и $(x, y) = y$

Задача H. Решение Задач

Для решения первой подзадачи достаточно перебрать всевозможные пары чисел (X, Y), где $1 \leqslant X, Y \leqslant N$ и посчитать количество подходящих пар (тех, для которых выполнено $\frac{X}{Y} = \frac{A}{B}$). Проверять данный критерий можно либо воспользовавшись дробными числами, либо преобразовав выражение и сравнив числа $X \cdot B$ и $A \cdot Y$.

Асимптотика: $O(N^2)$.

Для решения второй подзадачи заметим, что можно перебрать число X, после чего число Y определяется однозначно (в случае, если существует подходящая пара с таким значением X). Пусть мы зафиксировали некоторое число X, такое что $1 \leqslant X \leqslant N$. Должно быть выполнено равенство $X \cdot B = A \cdot Y$. Отсюда $Y = \frac{X \cdot B}{A}$. Если $X \cdot B$ не делится на A, данное значение X не подходит. В противном случае вычислим Y и проверим, что он лежит в диапазоне $[1, N]$. Следует обратить внимание, что нужно использовать 64-битный тип данных, чтобы избежать переполнения при умножении.

Асимптотика: $O(N)$.

Для решения третьей и четвертой подзадач нужно заметить несколько дополнительных фактов. В задаче требуется найти количество дробей, равных исходной дроби, числитель и знаменатель которых не превосходят N. Заметим, что дроби $\frac{A}{B}$ и $\frac{A \cdot K}{B \cdot K}$ равны для любого $K \geqslant 1$. Также, для всех K таких, что A и B делятся на K, можно получить равные дроби, разделив числитель и знаменатель на K. Для того, чтобы не рассматривать отдельно эти два случая, поделим изначально A и B на их наибольший общий делитель: пусть $D = gcd(A, B)$ — наибольший общий делитель A и B. Тогда скажем, что $A := \frac{A}{D}$, и $B := \frac{B}{D}$. Теперь задача сведена к тому, что нужно посчитать количество целых чисел $K \geqslant 1$, таких что $A \cdot K$ и $B \cdot K$ лежат в диапазоне $[1, N]$. Нетрудно заметить, что это количество равно $\min\left(\left\lfloor \frac{N}{A} \right\rfloor, \left\lfloor \frac{N}{B} \right\rfloor\right)$.

Таким образом, самая сложная часть задачи — вычисление наибольшего общего делителя. Для решения третьей подзадачи достаточно вычислить наибольший общий делитель при помощи разложения числа на простые множители.

Асимптотика: $O(\sqrt{N})$.

Для полного решения задачи необходимо вычислить наибольший общий делитель при помощи алгоритма Евклида. Также можно воспользоваться встроенными в некоторые языки функциями, вычисляющими НОД чисел. Например, `math.gcd` в языке Python, `std::gcd` в языке C++.

Асимптотика: $O(\log N)$.

Пример решения на языке C++:

```cpp
#include <cstdio>
#include <algorithm>

using namespace std;

long long gcd(long long a, long long b) {
    if (b == 0) {
        return a;
    }
    return gcd(b, a % b);
}

int main() {
    long long n, a, b;
    scanf("%lld%lld%lld", &n, &a, &b);

    long long d = gcd(a, b);
    a /= d;
    b /= d;

    printf("%lld\n", min(n / a, n / b));

    return 0;
}
```

Задача I. Лампочки

Автор задачи: Захар Яковлев, разработчик: Мария Жогова

Для решения задачи на частичный балл достаточно запрограммировать симуляцию процесса, описанного в условии задачи. Состояния всех лампочек будем хранить в массиве. Однако, так как количество лампочек может достигать 10^{18}, данное решение превысит лимит как по времени, так и по используемой памяти.

Для полного решения задачи заметим, что нас интересуют состояния лишь некоторых заданных q лампочек. Научимся для каждой лампочки независимо выяснять ее состояние. Для этого нужно вычислить, сколько раз Захар менял ее состояние. Переберем все степени двойки от 1 до 2^{60}, и для каждой степени 2^k поменяем состояние данной лампочки, если ее номер делится на 2^k. Достаточно перебрать степени двойки до 2^{60}, так как $2^{60} > 10^{18}$, а значит номер никакой лампочки не может делиться на 2^{60}.

Асимптотика: $\mathcal{O}(q \log n)$.

Задача J. Уравнение с НОК

$= a \cdot x/(a, x)$

Давайте переберем (a, x). Это какой-то делитель a, назовем его g. Теперь осталось проверить, что $x = b/a \cdot g$ подходит.