

Задача А. Совунья и соления

Совунья хочет, чтобы общее количество грибов делилось нацело на число банок k . Пусть изначально у неё n грибов. Обозначим остаток от деления n на k как $r = n \bmod k$.

Если $r = 0$, то грибы уже можно разложить поровну — дополнительные грибы не нужны, ответ равен 0.

Если же $r > 0$, то ближайшее число, не меньшее n и кратное k , равно $n + (k - r)$. Следовательно, Совунье нужно попросить ровно $k - r$ дополнительных грибов.

Таким образом, ответ вычисляется по формуле:

$$\text{ответ} = \begin{cases} 0, & \text{если } n \bmod k = 0, \\ k - (n \bmod k), & \text{иначе.} \end{cases}$$

Эквивалентно можно записать без условия:

$$\text{ответ} = (k - (n \bmod k)) \bmod k,$$

или через округление вверх:

$$\text{ответ} = \left\lceil \frac{n}{k} \right\rceil \cdot k - n.$$

Решение на частичные баллы. Можно последовательно увеличивать n на 1, пока текущее значение не станет кратным k , подсчитывая количество шагов. Такой перебор корректен, но выполняет до $k - 1$ итераций. При $k \leq 10^5$ это укладывается во время, поэтому решение получает не менее 50 баллов.

Пример реализации на Python.

```
n = int(input())
k = int(input())
```

```
r = n % k
if r == 0:
    print(0)
else:
    print(k - r)
```

Альтернативная однострочная реализация:

```
print((k - n \% k) \% k)
```

Задача В. Числовые шестерёнки

У нас есть четыре числа: a, b, c, d . Нужно разбить их на две упорядоченные пары (x_1, y_1) и (x_2, y_2) так, чтобы:

- $y_1 \neq 0$ и $y_2 \neq 0$ (деление на ноль запрещено);
- $x_1 \bmod y_1 = x_2 \bmod y_2$.

Поскольку всего четыре числа, общее количество способов разбить их на две упорядоченные пары невелико. Действительно, сначала выбираем первую пару: из 4 чисел можно выбрать упорядоченную пару (x_1, y_1) , где $x_1 \neq y_1$, 12 способами (4 варианта для x_1 , 3 — для y_1). Оставшиеся два числа образуют вторую пару, и их тоже можно упорядочить двумя способами. Однако проще и надёжнее просто перебрать все возможные разбиения.

Всего существует ровно 12 корректных способов разбить четыре различных позиции на две упорядоченные пары (с учётом порядка внутри пар и порядка самих пар не важен, так как нам достаточно найти любое решение). На практике удобно перебрать все возможные упорядоченные пары для первой шестерёнки, а затем — все возможные упорядоченные пары из оставшихся двух чисел для второй.

На самом деле, достаточно перебрать все 12 возможных упорядоченных разбиений. Например, можно явно перечислить все варианты распределения четырёх чисел по позициям:

- первая пара: (a, b) , вторая: (c, d) или (d, c) ;
- первая пара: (a, c) , вторая: (b, d) или (d, b) ;
- и так далее для всех $\binom{4}{2} = 6$ способов выбрать два числа для первой пары, и по 2 порядка внутри каждой пары.

Общее число проверяемых комбинаций — $6 \cdot 2 \cdot 2 = 24$. Однако даже перебор всех 24 вариантов работает мгновенно.

Важно помнить: операция взятия остатка $x \bmod y$ определена только при $y \neq 0$. Поэтому любую пару с нулём в правой позиции следует игнорировать.

Если хотя бы одна подходящая комбинация найдена — выводим её. Если таких нет — выводим -1 .

Пример реализации на Python с использованием функции `permutations` - генерация всех перестановок.

```
from itertools import permutations
```

```
a, b, c, d = map(int, input().split())
for x1, y1, x2, y2 in permutations([a, b, c, d]):
    if y1 == 0 or y2 == 0:
        continue
    if x1 % y1 == x2 % y2:
        print(x1, y1)
        print(x2, y2)
        break
else:
    print(-1)
```

Задача С. Глиняная табличка

На вход даётся таблица размером $N \times M$, состоящая из символов `*` (закрашенная клетка) и `.` (пустая клетка). Требуется рассмотреть все возможные прямоугольники размером 5×3 , которые полностью помещаются в таблицу, и для каждого определить, какую цифру от 0 до 9 он кодирует.

Правила распознавания цифры.

- У каждой цифры есть фиксированный шаблон 5×3 (дан в условии).
- Прямоугольник считается содержащим цифру d , если **во всех позициях**, где в шаблоне цифры d стоит `*`, в прямоугольнике тоже стоит `*`.
- Наличие дополнительных `*` в прямоугольнике (там, где в шаблоне стоит `.`) **допускается**.
- Если прямоугольник подходит под несколько цифр, выбирается **наибольшая** из них.
- Если не подходит ни под одну — он не вносит вклада в сумму.

Алгоритм решения.

1. Заранее запишем шаблоны всех цифр 0 – 9 в виде списка из 5 строк по 3 символа.
2. Для каждой возможной позиции левого верхнего угла прямоугольника 5×3 (то есть для всех i от 0 до $N - 5$ и j от 0 до $M - 3$):
 - проверим цифры от 9 до 0 (в порядке убывания!);
 - для каждой цифры сравним её шаблон с фрагментом таблицы: если во всех позициях, где в шаблоне `*`, в таблице тоже `*`, то цифра подходит;
 - как только нашли первую (а значит, наибольшую) подходящую цифру — добавляем её к общей сумме и переходим к следующему прямоугольнику.

3. Выводим итоговую сумму.

Проверка одного прямоугольника на соответствие шаблону занимает $5 \cdot 3 = 15$ операций. Общее число прямоугольников — не более $(100 - 4) \cdot (100 - 2) = 96 \cdot 98 < 10\,000$. Таким образом, полный перебор легко укладывается во время.

Частичные решения.

Если проверять только точное совпадение с шаблоном (запрещать лишние '*'), или обрабатывать только один прямоугольник (когда $N = 5$, $M = 3$), то решение будет работать на части тестов и получит до 40 баллов.

Пример реализации на Python.

```
digits = {
    0: ["***", "*.*", "*.*", "*.*", "***"],
    1: [".*.", ".*.", ".*.", ".*.", ".*."],
    2: ["***", ".*.", "***", "*..", "***"],
    3: ["***", ".*.", "***", ".*.", "***"],
    4: ["*.*", "*.*", "***", ".*.", ".*."],
    5: ["***", "*..", "***", ".*.", "***"],
    6: ["***", "*..", "***", "*.*", "***"],
    7: ["***", ".*.", ".*.", ".*.", ".*."],
    8: ["***", "*.*", "***", "*.*", "***"],
    9: ["***", "*.*", "***", ".*.", "***"]
}

n, m = map(int, input().split())
grid = [input().strip() for _ in range(n)]

total = 0
for i in range(n - 4):
    for j in range(m - 2):
        for d in range(9, -1, -1):
            match = True
            for di in range(5):
                for dj in range(3):
                    if digits[d][di][dj] == '*' and \
                       grid[i + di][j + dj] != '*':
                        match = False
                        break
                if not match:
                    break
            if match:
                total += d
                break

print(total)
```

Это решение корректно обрабатывает все особенности задачи: допускает лишние звёздочки, выбирает наибольшую подходящую цифру и перебирает все возможные позиции окна 5×3 . Оно получает полный балл.

Задача D. Великое тождество

Пусть на ленте записана строка s , состоящая только из цифр и не начинающаяся с нуля. Требуется определить, можно ли вставить знак равенства так, чтобы получилось корректное тождество вида $a = a$, где:

- слева от знака записано число a без ведущих нулей (то есть как обычное натуральное число);

- справа от знака записано то же число a , но, возможно, с ведущими нулями (например, 0012 считается допустимой записью числа 12).

Обозначим длину строки s через n . Если искомое число a существует, то его десятичная запись (без ведущих нулей) имеет некоторую длину k . Тогда вся строка s состоит из двух частей: первая — запись a длины k , вторая — запись того же числа, но длины $n - k$. Поскольку вторая запись может содержать ведущие нули, её длина не меньше k , то есть $n - k \geq k$, откуда $k \leq n/2$.

Следовательно, длина числа a не превосходит половины длины всей строки. Это ключевое наблюдение позволяет ограничиться рассмотрением только одной кандидатной позиции: разделим строку s пополам и возьмём вторую половину как возможную запись числа a с ведущими нулями. Удалим из неё все ведущие нули — получим строку t , которая должна совпадать с записью числа a без нулей.

Теперь проверим, может ли t быть левой частью тождества. Для этого:

- строка t должна быть префиксом первой половины исходной строки s ;
- оставшаяся часть первой половины (после префикса t) должна состоять только из нулей (иначе левая часть не будет равна числу, задаваемому t).

Если оба условия выполнены, то длина строки t и есть искомая позиция (после неё ставится знак «равно»). В противном случае тождество восстановить невозможно.

Частичные решения.

- **Группа А.** При малой длине строки (до 18 символов) можно перебрать все возможные позиции знака равенства, выделить левую и правую части, преобразовать их в целые числа и сравнить. Такой подход корректен, так как числа помещаются в стандартные целочисленные типы.
- **Группа В.** При длине до 1000 символов числа уже не помещаются в 64-битные типы, но можно сравнивать строки: для каждой позиции удалять ведущие нули из правой части и проверять, совпадает ли она с левой частью после удаления концевых нулей. Однако перебор всех позиций может быть медленным при больших длинах.

Полное решение.

Используя ключевое наблюдение о том, что длина числа a не превосходит половины длины строки, достаточно рассмотреть только одну кандидатную позицию — конец первой половины. После удаления ведущих нулей из второй половины проверяется, является ли полученная строка префиксом первой половины, за которым следуют только нули. Эта проверка выполняется за линейное время и работает даже при длине строки до $4 \cdot 10^5$.

```
for _ in range(int(input())):
    s = input()
    k = len(s) // 2
    s1 = s[:k]
    s2 = s[k:]
    s2 = s2.lstrip('0') # Обрезаем ведущие нули у второй половинки
    size = len(s2)
    if s1[:size] == s2 and (not s1[size:] or set(s1[size:]) == {'0'}):
        print(size)
    else:
        print(-1)
```

Задача Е. Модница

Требуется вычислить сумму наибольших общих делителей (НОД) всех подотрезков заданного массива. Ответ выводится по модулю $10^9 + 7$.

Ключевое наблюдение.

Для фиксированного левого конца подотрезка i рассмотрим последовательность значений $\gcd(a_i, a_{i+1}, \dots, a_j)$ при увеличении правого конца j . Эта последовательность:

- не возрастает;
- принимает не более $O(\log A)$ различных значений, где $A = \max a_i$.

Это связано с тем, что каждый раз, когда НОД уменьшается, он делится хотя бы на одно простое число, и за $O(\log A)$ шагов достигает 1.

Идея решения.

Будем обрабатывать массив слева направо. Для текущей позиции будем хранить сжатый список (или словарь) пар (g, cnt) , где:

- g — значение НОД для подотрезков, заканчивающихся в текущей позиции;
- cnt — количество таких подотрезков.

При переходе к новому элементу x :

- создаём новое состояние, начинающееся с подотрезка $[x]$ (т.е. добавляем пару $(x, 1)$);
- для каждого предыдущего значения g вычисляем $\text{gcd}(g, x)$ и добавляем соответствующее количество подотрезков к новому состоянию;
- одинаковые значения НОД автоматически объединяются (в словаре — через суммирование счётчиков).

Такой список всегда содержит не более $O(\log A)$ элементов.

Вычисление ответа.

На каждом шаге для каждого значения g в текущем состоянии добавляем к ответу величину $g \cdot \text{cnt}$, взятую по модулю $10^9 + 7$.

Сложность.

Алгоритм выполняет $O(n \log A)$ операций вычисления НОД, что укладывается в ограничения при $n \leq 200\,000$.

Подходы для разных подзадач.

- **Подзадача 1** ($n \leq 50$). Применяется прямой перебор всех подотрезков с тройным вложенным циклом: для каждой пары границ (l, r) вычисляется НОД всех элементов на отрезке. Такое решение имеет сложность $O(n^3 \log A)$ и достаточно для самых малых тестов.
- **Подзадача 2** ($n \leq 1000$). Улучшаем предыдущий метод: для фиксированного левого конца l последовательно расширяем правый конец r и обновляем текущий НОД по формуле $\text{НОД} = \text{gcd}(\text{текущий НОД}, a_r)$. Это даёт сложность $O(n^2 \log A)$, что укладывается в ограничения при $n \leq 1000$.
- **Подзадача 3 и 4** ($n \leq 200\,000$). Используется ключевое наблюдение: при фиксированном правом конце количество различных значений НОД среди всех подотрезков, заканчивающихся в этой позиции, не превышает $O(\log A)$. Поддерживаем список (или словарь) пар (g, cnt) , где g — значение НОД, а cnt — количество подотрезков с таким НОД. При переходе к следующему элементу обновляем все значения НОД и объединяем одинаковые. Сложность — $O(n \log A)$. При неаккуратной реализации решение может не пройти все тесты последней подзадачи и получит 70 баллов.

Итог. Основная идея полного решения — эффективное сжатие множества возможных НОДов для подотрезков, заканчивающихся в текущей позиции. Благодаря логарифмическому числу различных значений НОД алгоритм работает быстро даже при больших n .

Пример реализации на Python со словарем:

```
from math import gcd

MOD = 10**9 + 7

n = int(input())
a = [int(x) for x in input().split()]

ans = 0
cur = {}
for x in a:
    new = {x: 1}
    for g, cnt in cur.items():
        ng = gcd(g, x)
        new[ng] = new.get(ng, 0) + cnt
    cur = new
    for g, cnt in cur.items():
        ans = (ans + g * cnt) % MOD

print(ans)
```