

## 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. Сонный Крош

### Что требуется сделать?

Нам даны записи о том, во сколько Крош лёг спать и во сколько проснулся за несколько дней. Нужно посчитать, сколько **всего** часов и минут он проспал.

Важная деталь: иногда Крош ложится спать вечером, а просыпается утром **следующего дня** (например, лёг в 23:30, проснулся в 06:15). Наша программа должна уметь обрабатывать такие случаи.

### Основная идея

Лучше всего работать не с часами и минутами отдельно, а перевести всё время в **минуты с начала суток**. Например:

- $02:30 \rightarrow 2 \cdot 60 + 30 = 150$  минут,
- $23:10 \rightarrow 23 \cdot 60 + 10 = 1390$  минут.

Если время пробуждения (в минутах) **не больше**, чем время засыпания, значит, Крош проснулся на следующий день. Тогда к времени пробуждения нужно прибавить  $24 \cdot 60 = 1440$  минут (ровно одни сутки).

После этого длительность сна в этом эпизоде — это просто разность:

$$\text{сон} = \text{время\_пробуждения} - \text{время\_засыпания}.$$

Сложив такие значения за все дни, получим общее время сна в **минутах**.

В конце переводим минуты обратно в часы и минуты:

- `часы = время // 60`,
- `минуты = время % 60`.

#### Код

```
n = int(input())
total_minutes = 0

for _ in range(n):
    t1, t2 = input().split()

    # Преобразуем первое время в минуты
    h1, m1 = map(int, t1.split(':'))
    start = h1 * 60 + m1

    # Преобразуем второе время в минуты
    h2, m2 = map(int, t2.split(':'))
    end = h2 * 60 + m2

    # Если время пробуждения не позже засыпания --- значит, на следующий день
    if end <= start:
        end += 24 * 60

    total_minutes += end - start

hours = total_minutes // 60
minutes = total_minutes % 60
print(hours, minutes)
```

## Задача D. Глиняная табличка

На вход даётся таблица размером  $N \times M$ , состоящая из символов '\*' (закрашенная клетка) и '.' (пустая клетка). Требуется рассмотреть все возможные прямоугольники размером  $5 \times 3$ , 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  $5 \times 3$  (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  $5 \times 3$  (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  $5 \times 3$ . Оно получает полный балл.

## Задача Е. Модница

#### Что нужно сделать?

Дан массив чисел  $a_1, a_2, \dots, a_n$ . Для **каждого** подотрезка (то есть группы подряд идущих элементов) нужно посчитать XOR всех чисел в нём, а потом взять XOR всех этих результатов. Нужно вывести итоговое число.

#### Частичное решение для $n \leq 100$ (30 баллов)

Можно просто перебрать все возможные подотрезки. Для этого:

- Перебираем начало подотрезка  $l$  от 1 до  $n$ ,
- Перебираем конец подотрезка  $r$  от  $l$  до  $n$ ,
- Для каждого такого подотрезка считаем XOR от  $a_l$  до  $a_r$  «в лоб», то есть заново проходим по всем элементам от  $l$  до  $r$  и XOR-им их.

Это решение работает за  $O(n^3)$ , но при  $n \leq 100$  это укладывается в ограничения по времени.

**Улучшенное частичное решение для  $n \leq 5000$  (ещё 30 баллов)**

Можно не пересчитывать XOR каждый раз заново. Если мы уже знаем XOR подотрезка  $[l, r]$ , то XOR подотрезка  $[l, r + 1]$  получается просто добавлением  $a_{r+1}$ :

$$\text{XOR}(l, r + 1) = \text{XOR}(l, r) \oplus a_{r+1}.$$

Тогда для каждого начала  $l$  мы проходим вправо и накапливаем XOR, добавляя его к общему ответу. Это работает за  $O(n^2)$  и проходит при  $n \leq 5000$ .

**Полное решение для  $n \leq 200\,000$  (40 баллов)**

Заметим важное свойство XOR: если какое-то число участвует в итоговом XOR чётное число раз, оно не влияет на результат (потому что  $x \oplus x = 0$ ). Значит, нас интересует только **чётность** количества подотрезков, в которых встречается каждый элемент  $a_i$ .

Элемент  $a_i$  (на позиции  $i$ , считая с 0) входит в  $(i + 1) \cdot (n - i)$  подотрезков. Это произведение нечётно **только если оба множителя нечётны**, то есть:

- $i$  — чётное (тогда  $i + 1$  нечётно),
- $n - i$  — нечётно, что возможно только если  $n$  нечётно.

Получается:

- Если  $n$  чётное — каждый элемент встречается чётное число раз, ответ = 0.
- Если  $n$  нечётное — только элементы на чётных позициях ( $a_0, a_2, a_4, \dots$ ) вносят вклад. Нужно просто XOR-ить их все.

**Итоговый алгоритм:**

- Считываем  $n$  и массив  $a$ .
- Если  $n$  чётное — выводим 0.
- Иначе — XOR-им элементы  $a_0, a_2, a_4, \dots$  и выводим результат.

Это решение работает за  $O(n)$  и легко проходит максимальные ограничения.

```
n = int(input().strip())
a = list(map(int, input().split()))
if n % 2 == 0:
    print(0)
else:
    x = 0
    for i in range(0, n, 2):
        x ^= a[i]
    print(x)
```