Разбор задач

Задача 1. Том Сойер

В первой подзадаче ($b = r = 0$) у Тома только жёлтые билетики, за каждый десяток которых он получает одну награду. Найдём частное от деления y на 10, это число и является ответом.

```
y = int(input())
r = int(input())
b = int(input())
ans = y // 10
print(ans)
```

Во второй подзадаче ($b = 0$) у Тома только жёлтые и красные билетики. Сведём эту задачу к предыдущей: обменяем все красные билетики на жёлтые, добавим их к тем, которые уже есть у Тома и потом обменяем их все на награды.

```
ans = (r // 10 + y) // 10
```

Третья подзадача ($y, r, b \leqslant 10^5$) позволяет получить баллы за моделирование обменов билетиков с помощью циклов, например, так:

```
while b >= 10:
    b -= 10
    r += 1
while r >= 10:
    r -= 10
    y += 1
ans = 0
while y >= 10:
    y -= 10
    ans += 1
```

Полное решение: узнаем, сколько дополнительных красных билетиков можно получить из синих, и добавим их к уже имеющимся. Аналогично узнаем, сколько дополнительных жёлтых билетиков можно получить из красных, и добавим их к уже имеющимся. Определим количество наград за жёлтые билетики.

```
y = int(input())
r = int(input())
b = int(input())
ans = ((b // 10 + r) // 10 + y) // 10
print(ans)
```

Задача 2. Прогрессия

Решение заключается в разборе различных случаев. Посчитаем разности между выписанными числами: $d_1 = b - a$, $d_2 = c - b$.

Если $d_1 = d_2$, то записанные числа уже образуют арифметическую прогрессию. Значит, стёртое число было первым или четвёртым. Если оно стояло четвёртым, то принимало значение $c + d_1$. Если оно стояло первым, то равнялось $a - d_1$. Можно вывести любой из этих вариантов, они оба правильные.

В остальных случаях стёртое число находилось либо между a и b, либо между b и c. Если $d_1 = 2d_2$, то было стёрто второе число из четырёх, это число равно среднему арифметическому a и b. В противном случае стёрли третье число из четырёх, оно равно среднему арифметическому b и c.

```
a = int(input())
b = int(input())
```

```
c = int(input())

d1 = b - a
d2 = c - b

if d1 == d2:
    print(c + d1, 4)
elif d1 == 2 * d2:
    print((a + b) // 2, 2)
else:
    print((b + c) // 2, 3)
```

При этом нельзя использовать для анализа случаев условие $d_1 > d_2$ вместо $d_1 = 2d_2$, так как арифметическая прогрессия может быть убывающей, тогда значения d_1 и d_2 будут отрицательными. Но возможно использовать абсолютные значения: $|d_1| > |d_2|$.

Задача 3. 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))

```
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 4. Жребий Крижановского

В этой задаче нужно найти наименьшее число, которое встречается в данном массиве ровно один раз.

Первую группу ($n \leqslant 10^3$) можно решить перебором всех элементов массива, посчитав количество появлений каждого элемента в массиве. Среди тех элементов, которые встречаются ровно один раз, требуется выбрать наименьший. Сложность такого решения $O(n^2)$.

```
n = int(input())
a = [int(input()) for i in range(n)]

ans = -1
for elem in a:
    if a.count(elem) == 1:
        if ans == -1 or elem < ans:
            ans = elem
print(ans)
```

Во второй группе чисел может быть много, но все они не превосходят 10^5. Для решения этой задачи воспользуемся идеей «подсчёта»: для каждого допустимого значения элемента массива подсчитаем, сколько раз оно встречается в исходном массиве. Для этого заведём массив `cnt`, в котором индекс i принимает значение не превышающее 10^5, и если мы считали число num, то увеличим на 1 значение `cnt[`num`]`.

После этого в массиве `cnt` найдём такое минимальное значение индекса i, что `cnt[`i`]`$= 1$. Если же подходящее значение не найдётся, то выведем -1.

```
n = int(input())
cnt = [0] * (10 ** 5 + 1)
for i in range(n):
    num = int(input())
    cnt[num] += 1

i = 0
while i < len(cnt) and cnt[i] != 1:
    i += 1

if i < len(cnt):
    print(i)
else:
    print(-1)
```

Однако, если числа большие, то массив `cnt` не поместится в памяти. Чтобы такое решение проходило все тесты, нужно заменить массив `cnt` на структуру данных типа «словарь» (например, `dict` в Python или `map` в C++). Затем переберём ключи словаря в порядке возрастания и найти минимальный ключ, для которого значение в словаре будет равно 1.

```
n = int(input())
cnt = dict()
```

```
for i in range(n):
    num = int(input())
    if num not in cnt:
        cnt[num] = 0
    cnt[num] += 1
for num in sorted(cnt.keys()):
    if cnt[num] == 1:
        print(num)
        break
else:
    print(-1)
```

Есть и другое полное решение. Заметим, что если в массиве есть другой элемент, равный рассматриваемому, то после сортировки массива эти числа будут стоять рядом. А если число встречалось в массиве только один раз, то после сортировки соседями этого числа окажутся неравные ему числа. Отсортируем массив, пройдём по нему в порядке возрастания. Если текущий элемент не имеет равного соседа, то он и является ответом, так как если бы было меньшее число, удовлетворяющее условию, то мы бы уже нашли его. Если же мы прошли по всему массиву и не нашли ответ, то нужно вывести «-1»

```
n = int(input())
a = [int(input()) for i in range(n)]
a.sort()
a = [0] + a + [a[-1] + 1]
for i in range(1, n + 1):
    if a[i - 1] < a[i] < a[i + 1]:
        print(a[i])
        break
else:
    print(-1)
```

В этом варианте решения для удобства обработки мы добавили в массив два фиктивных элемента: 0 в начало и число, большее максимального элемента в массиве на 1 в конец для того, чтобы у первого и последнего элемента в массиве также были два соседа.

Задача 5. Долгое вычитание

В первой подзадаче достаточно просто реализовать предложенную последовательность действий.

```
n = int(input())
ans = 0
while n:
    s = str(n)
    if '1' in s or '3' in s or '5' in s or '7' in s or '9' in s:
        n -= 1
    else:
        n //= 2
    ans += 1
print(ans)
```

Для решения задачи на полный балл заметим, что количество операций деления не превосходит $\log_2 n \leqslant 60$, а вот количество выполненных подряд вычитаний единицы может быть очень большим. Например, при $n = 777$ вычитать придётся до числа 688 (почти сто раз).

Попробуем определить, до какого числа нам придётся вычитать единицу, пока не произойдёт следующая операция деления. Очевидно, что первая встретившаяся в числе нечётная цифра d должна стать чётной: ближайшей её четной цифрой, меньшей d, является $d - 1$. На последующих позициях должны стоять наибольшие чётные цифры, то есть восмёрки.

Было: <любые чётные цифры> <нечётная цифра d> <любые цифры>

Стало: <те же чётные цифры> <чётная цифра $d-1$> <8...8>

Количество пропущенных операций вычитания единицы равно разности между старым и новым числом. В предложенном ниже решении функция `first_odd` возвращает индекс первой нечётной цифры, и тогда следующее число строится по описанному алгоритму. Если же нечётных цифр нет, то функция возвращает -1 и тогда выполняется операция деления на 2.

```python
n = input()

def first_odd(n):
    for i in range(len(n)):
        if int(n[i]) % 2 == 1:
            return i
    return -1

ans = 0
while int(n) > 0:
    i = first_odd(n)
    if i == -1:
        n = str(int(n) // 2)
        ans += 1
    else:
        new_n = n[:i] + str(int(n[i]) - 1) + "8" * (len(n) - i - 1)
        ans += int(n) - int(new_n)
        n = new_n
print(ans)
```