Разбор задач

Задача 1. Флеш и Зум на пробежке

Посчитаем, в каких координатах окажутся Зум и Флеш через t секунд. Затем есть два варианта — либо Флешу нужно будет перебегать через начало стадиона, либо нет. Пусть спустя t секунд Флеш и Зум находятся в точках x_1 и x_2 соответственно. Тогда для первого варианта Флешу надо будет пробежать $d - |x_1 - x_2|$, а для второго — $|x_1 - x_2|$. Осталось только взять минимум из этих величин.

```
d = int(input())
v1 = int(input())
v2 = int(input())
t = int(input())

x1 = v1 * t % d
x2 = v2 * t % d

x1 = (x1 - x2) % d
print(min(x1, d - x1))
```

Задача 2. Миша и сериалы

Для начала заметим, что если $m \geqslant k$, то можно как минимум один раз нажать на кнопку. Далее будем считать, что $m < k$. При еще одном нажатии на кнопку Миша пропустит все интро, а также $k - m$ секунд серии. Осталось только проверить, можно ли пропустить такое количество сериала.

```
n = int(input())
m = int(input())
k = int(input())
t = int(input())
m %= k
skip = k - m
if skip > t:
    skip = 0
print(n - skip)
```

Задача 3. Серверы

Пронумеруем цепочки с серверами сверху вниз — самая длинная имеет номер 1, цепочка длины $n - 1$ имеет номер 2 и так далее. Рассмотрим цепочку i. В момент времени $i \cdot a$ первый сервер в ней получит данные от сервера 1. Затем $2 \cdot b \cdot (n - i)$ времени потребуется для того, чтобы данные от первого сервера в цепочке дошли до последнего и обратно. Затем еще нужно a секунд для того, чтобы данные вернулись на сервер с номером 1. Таким образом, для того, чтобы посчитать время, которое потребуется цепочке i, нужно воспользоваться формулой $a \cdot i + 2 \cdot b \cdot (n - i) + a$.

Тогда можно написать следующее решение за $\mathcal{O}(n)$, получающее 50 баллов:

```
n = int(input())
a = int(input())
b = int(input())
ans = 0
for i in range(1, n + 1):
    ans = max(ans, i * a + 2 * (n - i) * b  + a)
print(ans)
```

Для получения полного балла по задаче нужно рассмотреть формулу $a \cdot i + 2 \cdot b \cdot (n - i) + a$. Если $i = 1$, то значение выражения равно $2a + 2b(n - 1)$. Предположим, что вторая цепочка серверов позже пришлет свои данные. Тогда $3a + 2b(n - 2) > 2a + 2b(n - 1) \Leftrightarrow a > 2b$. Но тогда третья цепочка

серверов пришлет данные обратно еще позже и так далее. Таким образом получаем, что достаточно рассмотреть только $i = 1$ и $i = n$.

```
n = int(input())
a = int(input())
b = int(input())
print(max(2 * a + 2 * b * (n - 1), n * a + a))
```

Задача 4. Скучные квартиры

Для получения 60 баллов можно поступить следующим образом. Сначала сохраним в массив все скучные номера от 1 до n. Для этого можно просто пройтись циклом по всем числам от 1 до n и добавлять в список только скучные. Затем поменяем порядок элементов в списке — сначала поставим все номера из единиц, потом все из двоек и так далее. Для получения ответа достаточно просто найти номер элемента k. Это решение работает за $\mathcal{O}(n)$, а потому не набирает полный балл.

Для получения полного балла по задаче нужно заметить, что скучных чисел от 1 до n не очень много. Действительно, если число n состоит из x цифр, то скучных номеров не более $9x$. Значит можно просто перебрать все скучные номера от 1 до n в правильном порядке. Один из возможных способов — сначала перебрать цифру, из которой будут состоять скучные номера, а затем перебрать и сами эти номера.

```
n = int(input())
k = int(input())
ans = 0
for d in range(1, 10):
    flat = d
    while flat <= n:
        ans += 1
        if flat == k:
            print(ans)
            exit(0)
        flat = flat * 10 + d
```

Задача 5. Университетская команда

Для получения 25 баллов можно было просто перебрать все наборы из k чисел.

Далее заметим, что для фиксированного набора минимальная слабость команды достигается в том случае, если набор отсортирован по возрастанию или по убыванию. Тогда для получения 50 баллов можно любой квадратичной сортировкой отсортировать массив a, после чего перебрать все подотрезки длины k. Такое решение будет работать за $\mathcal{O}(n^2)$.

Теперь научимся быстро считать слабость на подотрезке. Заметим, что $|a_1 - a_2| + \ldots + |a_{k-1} - a_k| = (a_2 - a_1) + \ldots + (a_k - a_{k-1}) = a_k - a_1$. Таким образом можно за $\mathcal{O}(1)$ посчитать слабость на подотрезке.

Если использовать сортировку подсчетом, то решение будет работать за $\mathcal{O}(n + A)$, где A — максимальное число в массиве a. Такое решение набирает не менее 50 баллов.

Для получения решения на 100 баллов достаточно отсортировать массив любой быстрой сортировкой.

```
n = int(input())
k = int(input())
a = [int(input()) for i in range(n)]
a.sort()
ans = 10**9
for i in range(k - 1, n):
    ans = min(ans, a[i] - a[i - k + 1])
print(ans)
```