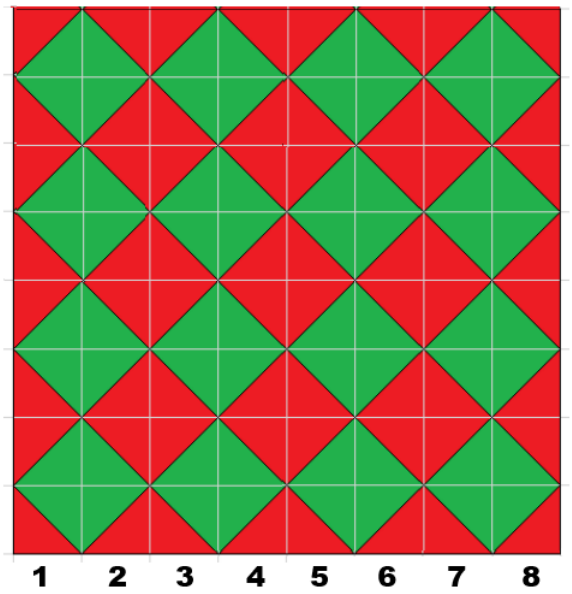
Разбор задач

# Задача 1. Аргайл




Первая подзадача:

Если посмотреть на получившийся узор, то можно заметить, что новый ряд зелёных квадратов появляется при каждом чётном $n$, а новый ряд красных — на каждом нечётном, начиная с 3. Таким образом, можно перебрать все промежуточные значения, увеличивая счётчик рядов на 1 при каждом выполнении условия чётности или нечётности для определённого цвета. Такое решение наберёт 50 баллов.

```
n = int(input())
red = 0
for i in range(2, n + 1):
    if i % 2 == 1:
        red += 1
green = 0
for i in range(1, n + 1):
    if i % 2 == 0:
        green += 1
print(red ** 2)
print(green ** 2)
```

Полное решение.

Воспользуемся операциями целочисленного деления. Один зелёный квадрат располагается на квадрате $2 \times 2$, поэтому ответ для этого цвета будет равен (n // 2) ** 2.

Если убрать полосы шириной 1 слева и снизу ткани, то ответ для красного цвета сведётся к предыдущему: теперь один красный квадрат располагается на квадрате 2 на 2, поэтому ответ для этого цвета будет равен ((n - 1) // 2) ** 2.

```
n = int(input())
red = ((n - 1) // 2) ** 2
green = (n // 2) ** 2
print(red)
print(green)
```

# Задача 2. Оптом — дешевле?

Прежде всего отметим, что всегда выгодно купить коробку вместо $k$ отдельных упаковок сока. Поэтому при необходимости приобрести определённое количество упаковок сока, нужно посчитать количество полных упаковок, взяв частное от деления на $k$, и остаток от деления на $k$ будет равен количеству отдельных упаковок, которое необходимо приобрести.

Рассмотрим сначала случай, когда хотя бы одна из величин $n$ или $m$ делится нацело на $k$. Для определённости будем считать, что $n = \text{boxesN} \cdot k$, а $m = \text{boxesM} \cdot k + \text{packsM}$ (при необходимости следует поменять местами $n$ и $m$ в рассуждениях).

Это значит, что на апельсиновый сок друзья должны потратить $boxesN \cdot p$ рублей, а на оставшиеся деньги они приобретут максимально возможное количество упаковок яблочного сока. Если это количество окажется не меньше $m$, то условия задачи будут выполнены.

Пример решения.

```
n = int(input())  # Требуется первого сока
m = int(input())  # Требуется второго сока
r = int(input())  # Цена упаковки сока
k = int(input())  # Количество упаковок сока в коробке
p = int(input())  # Цена коробки сока
s = int(input())  # Сумма денег

if n % k != 0:
    n, m = m, n
boxesN = n // k   # коробок первого сока

rem = s - boxesN * p  # останется денег после приобретения первого сока
boxes = rem // p  # количество коробок, купленное на оставшиеся деньги
packs = (rem % p) // r  # количество упаковок, купленное на оставшиеся
деньги

packsRem = boxes * k + packs  # суммарное количество упаковок, купленное на
оставшиеся деньги

if packsRem >= m:
    ans = n + packsRem
else:
    ans = -1
print(ans)
```

Для полного решения заметим, что нет разницы, на какой вид сока потратить излишек денег. Давайте считать, что первого сока мы купим столько, сколько необходимо, а излишки потратим на покупку максимального количества второго сока. Но возможны два способа приобрести первый сок: приобрести ровно $n$ упаковок целыми коробками и отдельными упаковками или вместо отдельных упаковок приобрести одну дополнительную коробку сока. На оставшиеся деньги приобретается максимальное количество второго сока, как в предыдущем решении.

Пример решения на языке Python.

```
n = int(input())  # Требуется первого сока
m = int(input())  # Требуется второго сока
r = int(input())  # Цена сока
k = int(input())  # Кол-во сока в коробке
p = int(input())  # Цена упаковки
s = int(input())  # Сумма денег
ans = -1

boxesN = n // k   # коробок первого сока
```

```
packsN = n % k      # упаковок первого сока

rem = s - boxesN * p - packsN * r  # останется денег после приобретения
n упаковок первого сока
if rem >=0:
    boxes = rem // p
    packs = (rem % p) // r
    if packs + boxes * k >= m:
        ans = max(ans, packsN + boxesN * k + packs + boxes * k)
boxesN += 1                        # добавим одну коробку первого сока
packsN = 0                         # а упаковки приобретать не будем
rem = s - boxesN * p  - packsN * r

if rem >= 0:
    boxes = rem // p
    packs = (rem % p) // r
    if packs + boxes * k >= m:
        ans = max(ans, packsN + boxesN * k + packs + boxes * k)

print(ans)
```

# Задача 3. Арт-объект

При $n = 1$ последовательность из одной колонны уже неубывающая, поэтому нужно вывести 0.

При $n = 2$, если $a_0 > a_1$, то ответ будет равен $a_0 - a_1$, иначе ответ 0. Разобрав эти два случая, можно набрать 30 баллов.

Если высоты колонн небольшие, то можно перебрать значения $d$. Прибавив к значениям $a_i$ нужные значения, проверим, что мы получили неубывающую последовательность.

Такие решения будут набирать не менее 30 баллов. Пример такого решения.

```
n = int(input())
a = [int(input()) for i in range(n)]
d = 0
while True:
    i = 1
    while i < n and a[i - 1] <= a[i]:
        i += 1
    if i == n:
        print(d)
        break
    for i in range(n):
        a[i] += i + 1
    d += 1
```

Для того чтобы набрать полный балл, можно заметить, что если разница между двумя соседними значениями $a_i - a_{i+1}$ положительна, то ответ должен быть не меньше этой разницы. Поэтому минимальное значение ответа будет равно наибольшему из значений $a_i - a_{i+1}$. Этот максимум можно вычислить сразу при считывании данных, необязательно сохранять все значения $a_i$ в массиве, достаточно запоминать только значение высоты предыдущей колонны.

Пример такого решения.

```
n = int(input())
ans = 0
prev = 0
for i in range(n):
```

```
    curr = int(input())
    ans = max(ans, prev - curr)
    prev = curr
print(ans)
```

# Задача 4. Длина риса

Отсортируем значения $h_i$ по возрастанию, то есть дальше будем считать, что $h_1 \leqslant h_2 \leqslant \cdots \leqslant h_n$.

Определим, когда мы можем угодить группе поваров. Пусть есть группа поваров с номерами $i_1 < i_2 < \cdots < i_k$, причём $h_{i_1} \leqslant h_{i_2} \leqslant \cdots \leqslant h_{i_k}$. Тогда этой группе мы можем угодить тогда и только тогда, когда $h_{i_j} - h_{i+j-1} \leqslant 1$. Сначала $i_2$-й повар переубедит $i_1$-го повара, затем $i_3$-й повар переубедит $i_1$-го и $i_2$-го поваров, ..., в конце $i_k$-й повар переубедит $i_1$-го, $i_2$-го, ..., $i_{k-1}$-го поваров. В этом случае все повара сойдутся во мнении, что рис, по мнению $i_k$-го повара, самый оптимальный.

Заметим, что если $i_j = i_{j-1} + k$ и при этом $h_{i_j} - h_{i_{j-1}} \leqslant 1$, то $h_{i_{j-1}} \leqslant h_{i_{j-1}+1} \leqslant h_{i_{j-1}+2} \leqslant \leqslant \cdots \leqslant h_{i_{j-1}+2} = i_j$, а также модуль разности двух соседних элементов не превосходит 1. Значит, будем искать группу поваров $i_1 < i_2 < \cdots < i_k$, где $i_2 = i_1 + 1$, $i_3 = i_1 + 2$, $\cdots$, $i_k = i_1 + k - 1$, то есть будем рассматривать группу поваров с подряд идущими номерами.

Для решения первой подгруппы можно перебрать все группы поваров с индексами $i, i+1, i+2, \cdots, j$ и проверить, что $h_j - h_i = j - i$, и для всех индексов, для которых выполняется это условие, выбрать максимальное значение $j - i + 1$. Действительно, так как все $h_i$ различны, для подходящей группы выполняется $h_i < h_{i+1} < \cdots h_j$, причём модуль разности между соседними элементами равен 1. Значит, $h_{i+1} = h_i + 1$, $h_{i+2} = h_{i+1} + 1 = h_i + 2$, $\cdots$, $h_j = h_i + j - i$. Найдём разность между первым и последним элементами $h_j - h_i = h_i + j - i - h_i = j - i$. Значит, если выполняется равенство $h_j - h_i = j - i$, то группа поваров сможет между собой договориться. Данное решение работает за $O(n^2)$, так как мы перебираем левую и правую границы группы независимо друг от друга.

```
n = int(input())
h = [int(input()) for i in range(n)]
h.sort()
ans = 0
for i in range(n):
    for j in range(i, n):
        if h[j] - h[i] == j - i:
            ans = max(ans, j - i + 1)
print(ans)
```

Во второй подгруппе так же проверять не получится. Может быть, к примеру, ситуация $h_i = h_{i+1} = h_{i+2}$, $h_{i+3} = h_i + 3$, и вроде бы $h_j - h_i = h_i + 3 - h_i = 3$, $j - i = 3$, но группа подходящей не является.

Давайте перебирать левую границу $i$, а затем перебирать максимально большую правую границу $j$. Изначально $i = j$. Если $h_{j+1} - h_j \leqslant 1$, то увеличим $j$ на единицу. Если в какой-то момент мы дошли до последнего элемента массива или неравенство не выполняется, это значит, что мы нашли максимально большую правую границу. Данное решение работает за $O(n \cdot k)$, где $k$ — ответ на задачу. В худшем случае асимптотика будет равна $O(n^2)$, если $k = n$.

```
n = int(input())
h = [int(input()) for i in range(n)]
h.sort()
ans = 0
for i in range(n):
    j = i
    while j < n - 1 and h[j+1] - h[j] <= 1:
        j += 1
    ans = max(ans, j - i + 1)
```

```
print(ans)
```

Для решения третьей подгруппы нужно немного оптимизировать идею второй подгруппы. Давайте зафиксируем счётчик count, который показывает количество элементов в группе. Первоначально count = 1 (мы рассматриваем первый элемент). Переберём элементы от первого до последнего. Если разница между очередной длиной и прошлой длиной меньше либо равна 1, увеличим count на единицу. Если же разница больше 1, то это значит, что мы перешли к новой группе, поэтому сбросим значение count = 1. Одновременно с этим, нужно искать максимум из всех значений count.

Данное решение работает за $O(n \log n)$, так как необходимо отсортировать массив и пройти циклом по его элементам.

```
n = int(input())
h = [int(input()) for i in range(n)]
h.sort()
count = 1
ans = 1
for i in range(1, n):
    if h[i] - h[i-1] <= 1:
        count += 1
    else:
        count = 1
    ans = max(ans, count)
print(ans)
```

# Задача 5. Чувство прекрасного

Пусть мы выбрали набор цветов (чисел) для полоски. Тогда яркость будет минимальной, если цвета в полоске упорядочить по неубыванию. Пусть полоска состоит из ячеек с цветами $c_1 \leqslant c_2 \leqslant ... \leqslant c_n$, тогда яркость такой полоски равна $c_n - c_1$. Также заметим, что невыгодно «пропускать» цвета. То есть, если для выбранной полоски есть какой-то цвет $c$, причём $c_1 < c < c_n$, то в наилучшем ответе все ячейки цвета $c$ должны присутствовать (цвет $c$ должен быть использован ровно $a_c$ раз). Иначе можно выкинуть из полоски последнюю ячейку цвета $c_n$ и добавить ячейку цвета $c$; для такой полоски ответ не будет хуже.

Рассмотрим большую полоску, состоящую из цвета 1, повторённого $a_1$ раз, затем из цвета 2, повторённого $a_2$ раз и т.д. В этой полоске нужно выбрать $n$ подряд идущих клеток с минимальной разницей значений в правой и левой клетках.

Решение, реализующее этот процесс, имеет сложность $O(a_1 + a_2 + ... + a_m)$ и набирает 25 баллов. Пример такого решения.

```
n = int(input())
m = int(input())
c = []
for i in range(1, m + 1):
    c += [i] * int(input())
ans = m + 1
for i in range(len(c) - n + 1):
    ans = min(ans, c[i + n - 1] - c[i])
print(ans)
```

Но сумма $a_1 + a_2 + ... + a_m$ может достигать $10^{14}$, что очень много. Для того чтобы получить больше баллов, нужно избавиться от любых значений $a_i$ в оценке сложности.

В предложенном решении в оптимальной полоске мы использовали все цвета полностью, кроме, быть может, цвета, в который покрашены первая и последняя клетки полоски. Но здесь тоже можно применить уже использованную идею: если цвет $c_1$ использован не полностью, то уберём последнюю

клетку цвета $c_n$ и добавим клетку цвета $c_1$. То есть оптимальный ответ можно искать только среди полосок следующего вида. Выберем какой-то начальный цвет полоски $c$. Добавим в полоску все клетки цвета $c$. Если размер полоски меньше $n$, добавим клетки цвета $c+1$. Если по-прежнему не набралось $n$ клеток, добавим все клетки цвета $c+2$, и т.д. То есть мы будем использовать все цвета полностью, пока не наберётся $n$ клеток. Поэтому решение задачи сводится к тому, что в массиве $a$ нужно выбрать два индекса $i \leqslant j$ — цвет первой и последней клеток полоски, при этом краски цветов от $i$ до $j$ должно быть достаточно для покраски $n$ клеток, то есть $a_i + ... + a_j \geqslant n$. Тогда яркость такой полоски равна $j - i$, и необходимо найти минимальное значение $j - i$ для всех подходящих пар $(i, j)$.

Если перебирать пары индексов $(i, j)$ и сумму чисел $a_i + ... + a_j$ считать циклом (или, например, функцией sum в Python), то сложность решения получится $O(m^3)$. Пример такого решения.

```
n = int(input())
m = int(input())
a = [int(input()) for i in range(m)]
ans = m
for i in range(m):
    for j in range(i, m):
        sm = sum(a[i:j+1])
        if sm >= n:
            ans = min(ans, j - i)
print(ans)
```

Это решение можно улучшить, если сумму внутри цикла не пересчитывать заново, а при увеличении $j$ на 1 добавлять к сумме sm значение одного элемента $a_j$. Сложность такого решения $O(m^2)$. Также цикл по $j$ можно заканчивать, когда сумма $sm$ превысила $n$, поскольку добавлять новые цвета к такой полоске бессмысленно. Такое решение набирает 70 баллов.

```
n = int(input())
m = int(input())
a = [int(input()) for i in range(m)]
ans = m
for i in range(m):
    sm = 0
    for j in range(i, m):
        sm += a[j]
        if sm >= n:
            ans = min(ans, j - i)
            break
print(ans)
```

Чтобы набрать 100 баллов, нужно использовать метод двух указателей. Для этого заметим, что если для какого-то значения $i$ нашли подходящее наименьшее значение $j$, то при увеличении $i$ значение $j$ не изменится или увеличится. Поэтому после увеличения $i$ запустим цикл, увеличивающий значение $j$, пока новая сумма не станет больше или равна $n$. При каждом увеличении $j$ к сумме добавляется $a_j$, а при увеличении $i$ сумма уменьшается на $a_i$. Такое решение имеет сложность $O(m)$. Пример такого решения.

```
n = int(input())
m = int(input())
a = [int(input()) for i in range(m)]
j = 0
sm = 0
ans = m
for i in range(m):
    while j < m and sm < n:
```

```
            sm += a[j]
            j += 1
        if sm >= n:
            ans = min(ans, j - i - 1)
        sm -= a[i]
print(ans)
```