Разбор задач

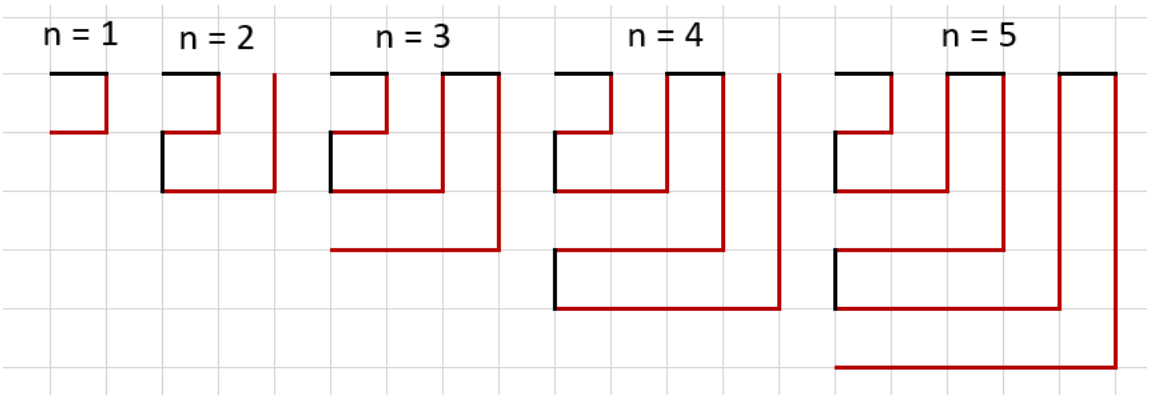
Максимальное количество баллов — 500

Задача 1. Змейка

При переходе от n к $n+1$ добавляется одна линия длины 1 и две линии длины $n+1$. Чтобы набрать 40 баллов, можно написать цикл, суммирующий эти значения.

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

Чтобы набрать 100 баллов необходимо заменить цикл на сумму арифметической прогрессии. Раскрасим линии так, как показано на рисунке. Тогда длина красных линий равна $n \times (n+1)$ (как две суммы членов арифметической прогрессии от 1 до n), а длина чёрных линий равна n. Всего получится $n \times (n+2)$.




```
n = int(input())
ans = n * (n + 2)
print(ans)
```

Задача 2. Две сестры

В первой подзадаче $a = 1$: каждый день сёстры принимают по $b+1$ таблетке. Их хватит на $\lfloor \frac{n}{(b+1)} \rfloor$ дней (целая часть частного).

Во второй подзадаче (40 баллов) достаточно написать решение, моделирующее процесс по дням. Заведём счетчик дней и будем определять, хватит ли нам оставшихся таблеток ещё на один день. Цикл продолжается, пока у нас есть таблетки ($n > 0$). Внутри цикла уменьшаем значение n на b, а также если номер шага цикла делится на a, то уменьшаем ещё раз на 1.

Цикл остановится, когда таблетки закончатся, то есть $n \leqslant 0$. При этом, если оказалось, что $n < 0$, то есть количество таблеток стало отрицательным, то таблеток не хватило при последней итерации цикла, поэтому количество дней нужно уменьшить на 1. Пример такого решения:

```
a = int(input())
b = int(input())
n = int(input())
day = 0
while n > 0:
    n -= b
    if day % a == 0:
        n -= 1
    day += 1
if n < 0:
```

```
    day -= 1
print(day)
```

При большом n эта программа работает долго. Для полного решения заметим, что процесс имеет период из a дней, за которые сёстры выпивают $ab + 1$ таблеток. Посчитаем количество полностью завершённых циклов c, поделив n на $ab + 1$.

```
c = n // (a * b + 1)
```

За эти дни будет принято $c(ab + 1)$ таблеток. Вычтем это значение из n, получим количество оставшихся таблеток. Их не хватит на полный цикл.

В первый день нового цикла сёстры выпьют $b + 1$ таблетку. Проверим, что $n \geqslant b + 1$. Если да, то добавим ещё один день, а также добавим количество последующих дней, в каждый из которых только Белла Прокофьевна выпивает по b таблеток. Количество таких дней найдём целочисленным делением на b.

Пример решения на языке Python.

```
a = int(input())
b = int(input())
n = int(input())
pills_in_a_days = a * b + 1
c = n // pills_in_a_days
days = c * a
n %= pills_in_a_days
if n >= b + 1:
    days += 1
    n -= b + 1
    days += n // b
print(days)
```

Ещё один быстрый способ решения этой задачи – двоичный поиск по ответу. Его можно применить, поскольку количество принимаемых таблеток с каждым днём увеличивается, а само количество выпитых таблеток за интересующее количество дней найти несложно.

```
a = int(input())
b = int(input())
n = int(input())
left = 0
right = n + 1
while right - left > 1:
    middle = (left + right) // 2
    pills = middle * b + 1 + (middle - 1) // a
    if pills > n:
        right = middle
    else:
        left = middle
print(left)
```

Задача 3. Мастерство фотографии

Рассмотрим разные по эффективности решения задачи.

Первое решение набирает 20 баллов. Переберём четырьмя вложенными циклами все возможные варианты размеров рядов. Первый и четвёртый ряд могут содержать от 0 до a мальчиков, второй и третий — от 0 до b девочек. Проверим, что сумма первого и четвёртого равна a, второго и третьего равна b и сумма второго и четвёртого не превосходит c (последнее условие означает, что для такого размещения хватит стульчиков).

Такой подход позволяет решить задачу при $a, b, c \leqslant 50$.

Пример решения на языке Python.

```
a = int(input())
b = int(input())
c = int(input())
ans = 10**18
for i1 in range(a + 1):
    for i4 in range(a + 1):
        for i2 in range(b + 1):
            for i3 in range(b + 1):
                if i1 + i4 == a and i2 + i3 == b and i2 + i4 <= c:
                    ans = min(ans, max(i1, i2, i3, i4))
print(ans)
```

Ускорим это решение. Заметим, что если мы определили число сидящих девочек (это значение i2 в примере выше), то число стоящих девочек перебирать не нужно, оно равно b − i2. Также нужно перебирать только число стоящих на стульчиках мальчиков, получив число сидящих мальчиков вычитанием. То есть мы будем перебирать только два значения: количество стульчиков, которые заняли девочки, и количество стульчиков, занятых мальчиками. Нужно ещё проверить, что сумма этих величин не превосходит c.

Такое решение проходит тесты, в которых $a, b, c \leqslant 1000$ и набирает 30 баллов.

Пример решения на языке Python.

```
a = int(input())
b = int(input())
c = int(input())
ans = 10**18
for i in range(a + 1):
    for j in range(b + 1):
        if i + j <= c:
            ans = min(ans, max(i, a - i, j, b - j))
print(ans)
```

Теперь рассмотрим два решения, содержащих один цикл.

Будем перебирать величину ans — значение самого широкого ряда, то есть мы хотим разместить детей так, чтобы в каждом ряду было не более ans человек. Тогда мы можем посадить ans мальчиков на корточки в первом ряду, а оставшимся мальчикам понадобятся стульчики. Аналогично, мы можем поставить ans девочек в третьем ряду, а оставшимся девочкам понадобятся стульчики. Посчитаем количество нужных стульчиков, и если оно не превосходит c, то мы нашли подходящий ответ (необходимо вывести минимальное значение ans, при котором хватило стульчиков для размещения, также должно выполняться условие, что значения a и b не превышают 2 * ans).

Такое решение пройдёт тесты в которых $a, b, c \leqslant 10^6$, и наберёт от 50 до 70 баллов в зависимости от используемого языка программирования. Пример такого решения:

```
a = int(input())
b = int(input())
c = int(input())
ans = 1
while True:
    na = max(0, a - ans)  # Кол-во стульев для мальчиков
    nb = max(0, b - ans)  # Кол-во стульев для девочек
    if a <= 2 * ans and b <= 2 * ans and na + nb <= c:
        print(ans)
        break
    ans += 1
```

Во втором линейном решении переберём число стульев i, используемых для мальчиков от 0 до c. Тогда девочкам останется $c - i$ стульев.

Посчитаем ширину ряда, необходимую для размещения a мальчиков в двух рядах, если можно использовать i стульев. Хотя бы в одном ряду окажется не менее, чем $\lceil a/2 \rceil$ мальчиков (частное от деления $a/2$, округлённое вверх, что можно вычислить по формуле (a + 1) // 2). Но также не менее чем $a - i$ мальчиков будут сидеть на корточках в первом ряду, т.к. число стульев для мальчиков из четвёртого ряда не превышает i. Поэтому ширина наибольшего из двух рядов мальчиков есть максимум из величин $\lceil a/2 \rceil$ и $a - i$.

Посчитаем ширину максимального ряда у девочек, это максимум из $\lceil b/2 \rceil$ и $b - (c - i)$.

Это решение также пройдёт все тесты, где числа не превосходят 10^6, и наберёт от 50 до 72 баллов. Пример решения на языке Python.

```
a = int(input())
b = int(input())
c = int(input())
ans = 10**18
for i in range(c + 1):
    ma = max((a + 1) // 2, a - i)
    mb = max((b + 1) // 2, b - (c - i))
    ans = min(ans, max(ma, mb))
print(ans)
```

Наконец, рассмотрим решения, набирающие 100 баллов.

Для начала рассмотрим решение при помощи двоичного поиска по ответу. Возьмём первое решение с одним циклом, в котором перебиралось значение ответа ans и заметим, что для небольших значений ans невозможно расставить детей так, что ширина каждого ряда не превосходит ans, а начиная с какого-то момента это становится возможно. Мы искали это минимальное подходящее значение ans линейным поиском, но можно заменить его на двоичный поиск. Возьмём в качестве значения left = 0 такое значение ans, которая заведомо не может быть ответом, а в качестве значения right = max(a, b) — значение ширины ряда, при котором рассадка заведомо возможна. Далее будем сдвигать границы left и right, выбирая середину отрезка от left до right. Проверка того, можно ли рассадить детей при выбранной допустимой ширине ряда, аналогична представленной в решении с одним циклом.

Пример решения на языке Python.

```
a = int(input())
b = int(input())
c = int(input())
left = 0
right = max(a, b)
while right - left > 1:
    m = (left + right) // 2
    chairs = max(0, a - m) + max(0, b - m)
    if chairs <= c and a <= 2 * m and b <= 2 * m:
        right = m
    else:
        left = m
print(right)
```

Наконец обсудим полное решение, которое не содержит ни одного цикла и имеет сложность $O(1)$. Задачу можно решить при помощи нескольких условий и формул. Не ограничивая общности, будем считать, что мальчиков не больше, чем девочек ($a \leqslant b$), иначе поменяем a и b местами, т.к. условие в некотором смысле «симметрично» для мальчиков и девочек.

Рассмотрим сначала случай, когда стульев нет совсем. Тогда ответом является число b — все девочки стоят. Если теперь начать добавлять стулья, то количество стоящих девочек станет уменьшаться на 1 с каждым дополнительным стулом, до тех пор, пока оно не сравняется с количеством мальчиков. То есть если количество стульев c не превосходит разности $b - a$, то каждый дополнительный стул будет уменьшать ответ на 1, то есть при $c \leqslant b - a$ ответом будет $b - c$.

Пусть $c > b - a$. Тогда мы уже использовали $b - a$ стульев, чтобы уравнять количество девочек без стульев (стоящих) с количеством мальчиков. Вычтем из значения c значение $b - a$, получим количество оставшихся стульев. Сейчас есть $b - a$ сидящих девочек во втором ряду, a стоящих девочек в третьем ряду и a сидящих на корточках мальчиков в первом ряду. Самые широкие ряды — это первый и третий. Чтобы уменьшить их ширину на 1, теперь нужно 2 стула (один стул для мальчика, другой — для девочки). Поэтому поделив c на 2 (нацело), мы получим количество мальчиков и девочек, на которое может быть уменьшена ширина первого и третьего рядов, то есть ответом будет $a - \lfloor c/2 \rfloor$.

Но также нужно учесть, что и в первом, и во втором случае ответ не может быть меньше значения половины от числа девочек, округлённого вверх, то есть при вычислении ответа в каждом случае нужно ещё взять максимум найденного значения и значения (b + 1) // 2.

Пример решения на языке Python.

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

if a > b:
    a, b = b, a

if b - c >= a:
    print(max(b - c, (b + 1) // 2))
else:
    c -= b - a
    print(max(a - c // 2, (b + 1) // 2))
```

Задача 4. Места в ряду

В первой подзадаче $k = 1$, то есть нам нужно обработать только одного нового человека. Можно написать цикл, который просто переберёт все места, для каждого свободного места посчитает расстояние до краёв и выберет место с минимальным расстоянием.

```
n = int(input())
k = int(input())
s = input()
ans = 0
min_dist = n + 1
for i in range(n):
    if s[i] == '0':
        dist = min(i + 1, n - i)
        if dist < min_dist:
            min_dist = dist
            ans = i
print(ans + 1)
```

Во второй подзадаче строка s состоит только из символов «0». Можно заметить, что, так как все места изначально свободны, места будут браться поочерёдно с левого и правого краёв, т.е. в последовательности 1, n, 2, $n-1$, 3, $n-2$,... Нужно вывести k первых элементов этой последовательности.

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

```
s = input()
for i in range(k):
    if i % 2 == 0:
        print(1 + i // 2)
    else:
        print(n - i // 2)
```

В третьей подзадаче $n \leqslant 1000$, можно написать решение для общего случая, но неэффективное. Можно взять первое решение и k раз находить ответ, затем изменять в строке символ «0» на «1», тем самым делая найденное место занятым.

```
n = int(input())
k = int(input())
s = input()
for j in range(k):
    ans = 0
    min_dist = n + 1
    for i in range(n):
        if s[i] == '0':
            dist = min(i + 1, n - i)
            if dist < min_dist:
                min_dist = dist
                ans = i
    print(ans + 1)
    s = s[:ans] + "1" + s[ans + 1:]
```

Заметим, что если какой-то вновь пришедший человек занял место «слева», то следующий человек может взять только место с большим номером, причём это окажется следующее свободное место слева. Аналогично, если кто-то занял какое-то место справа, то следующее занятое справа место будет иметь меньший номер. Поэтому не надо каждый раз просматривать все имеющиеся места, а достаточно только найти первое свободное место слева и ближайшее свободное место справа, выбрать наибольшее подходящее из них, а для следующего человека продолжать поиск с тех мест, которые были найдены ранее.

Пусть i и j указывают на два равноудалённых от краёв места, i — от левого края, а j — от правого края. Начнём со значений i=1 и j=n. Если из этих двух место одно — свободно, то нужно занять его, а если оба свободны — то нужно занять левое. При обработке нового пришедшего человека будем увеличивать i и уменьшать j, пока среди этих мест не найдётся свободное. Если окажется свободным место i, то выберем его, иначе выберем место j. При выборе места заменим соответствующий символ «0» в строке на «1», чтобы не выбрать это место повторно.

Пример решения на языке Python.

```
n = int(input())
k = int(input())
s = ["1"] + list(input())
i = 1
j = n
for _ in range(k):
    while s[i] == '1' and s[j] == '1':
        i += 1
        j -= 1
    if s[i] == "0":
        print(i)
        s[i] = "1"
    else:
        print(j)
```

```
        s[j] = "1"
```

В этой реализации мы добавляем в начало строки ещё один фиктивный элемент «1», чтобы элементы строки нумеровались от 1 до n. Также поскольку в Python строки — неизменяемые объекты, то для быстрой замены элемента строки мы преобразуем строку в список из символов «0» и «1», тогда можно будет выполнять присваивания вида s[i] = "1".

Задача 5. Гармония

В первой подзадаче строка состоит только их одних нулей, поэтому любая подстрока чётной длины является гармонической. Нужно подсчитать количество подстрок чётной длины в строке длины n. Заметим, что подстрок длины k в строке будет $n-k+1$, поэтому можно просуммировать в цикле значения $n-k+1$ для $k = 2, 4, 6, ...$ Можно вместо цикла использовать формулу для суммы арифметической прогрессии, но это не требуется в данной подзадаче.

```
n = int(input())
s = input()
ans = 0
for k in range(2, n + 1, 2):
    ans += n - k + 1
print(ans)
```

Дальнейшие подзадачи предполагают общее решение, но разной алгоритмической сложности.

Решение сложности $O(n^3)$ можно получить, если перебрать начало подстроки i и конец подстроки j и для рассматриваемой подстроки проверить выполнение условия подсчётом числа нулей и единиц.

```
n = int(input())
s = input()
ans = 0
for i in range(n):
    for j in range(i + 1, n + 1):
        c0 = 0
        c1 = 0
        for t in range(i, j):
            if s[t] == '0':
                c0 += 1
            else:
                c1 += 1
        if c0 % 2 == 0 and c1 % 2 == 0:
            ans += 1
print(ans)
```

Сложность этого решения можно улучшить, если избавиться от вложенного цикла по t. Для этого заметим, что, когда правая граница j увеличивается на 1, не нужно пересчитывать значения c0 и c1 заново, достаточно только учесть один новый добавленный символ. Такое решение будет иметь сложность $O(n^2)$.

```
n = int(input())
s = input()
ans = 0
for i in range(n):
    c0 = 0
    c1 = 0
    for j in range(i, n):
        if s[j] == '0':
            c0 += 1
```

```
        else:
            c1 += 1
        if c0 % 2 == 0 and c1 % 2 == 0:
            ans += 1
print(ans)
```

Полное решение имеет сложность $O(n)$. Будем рассматривать все префиксы исходной строки, то есть первые j символов, увеличивая значение j. Для данного префикса длины j посчитаем количество нулей и единиц на этом префиксе в переменных c0 и c1. Эти значения на самом деле не требуется пересчитывать заново, а нужно только учесть один новый добавляемый символ. Теперь мы хотим определить, сколько подстрок исходной строки, у которых правая граница совпадает с j, являются гармоничными. Такие строки получаются из рассматриваемого префикса отбрасыванием какого-то другого, меньшего префикса (в том числе, возможно, и пустого префикса). При этом чтобы получилась гармоничная подстрока, мы должны отбросить такой префикс, на котором чётность числа нулей совпадает с чётностью c0, а чётность числа единиц совпадает с чётностью c1. Значит, нам нужно знать, сколько ранее мы рассмотрели префиксов, у которых число нулей и число единиц имеет определённую чётность.

Давайте для каждого префикса определим его *тип*. Типом назовём пару из остатка от деления количества нулей на префиксе на 2 и остатка от деления количества единиц на префиксе на 2. Таким образом, рассмотрев какой-то префикс, нужно добавить к ответу число, равное количеству ранее рассмотренных префиксов такого же типа.

Пример такого решения на языке Python.

```
n = int(input())
s = input()
count = [[0, 0], [0, 0]]
ans = 0
c0 = 0
c1 = 0
count[0][0] = 1
for c in s:
    if c == '0':
        c0 += 1
    else:
        c1 += 1
    ans += count[c0 % 2][c1 % 2]
    count[c0 % 2][c1 % 2] += 1
print(ans)
```

В этом решении в массиве count хранится количество префиксов каждого из четырёх типов. Например, count[0][0] равен количеству префиксов, у которых чётное число нулей и чётное число единиц. count[1][0] равен количеству префиксов, у которых нечётное число нулей и чётное число единиц. count[0][1] равен количеству префиксов, у которых чётное число нулей и нечётное число единиц. count[1][1] равен количеству префиксов, у которых нечётное число нулей и нечётное число единиц.

В самом начале count[0][0] равен 1, что соответствует пустому префиксу (у него чётное число нулей и единиц), остальные значения count равны 0.

Рассматриваем следующий символ, в зависимости от его значения изменяем c0 или c1. Тип получившегося префикса есть [c0 % 2][c1 % 2]. Добавим к ответу count[c0 % 2][c1 % 2] и увеличим это значение на 1, чтобы учесть этот префикс в дальнейшем.