# Задача A. Максимальное произведение

Если мы вычислили произведение $S(i,j) = (j-i) \cdot min(a_i, a_j)$ при $i = L$ и $j = R$ для $L < R$, то для остальных значений индексов $i$ и $j$, таких что $L \leqslant i < j \leqslant R$, произведение $S(i,j)$ может иметь большую величину, чем $S(L,R)$, только если $min(a_L, a_R) < min(a_i, a_j)$. Это означает, что если в процессе поиска мы достигли индексов $i = L$ и $j = R$, и если $a_L < a_R$, нет смысла проверять индексы $i = L$ и $j = R-1$, потому что это не увеличит значение $min(a_i, a_j)$. Таким образом, если $a_L < a_R$, на следующем шаге мы увеличиваем $L$ на единицу и оставляем R без изменений; аналогично, если $a_L \geqslant a_R$, на следующем шаге мы оставляем $L$ без изменений и уменьшаем $R$ на единицу.

# Задача B. Квадратами, квадратами

Первоначальное понимание, которое помогает решить эту задачу, заключается в том, что если мы можем использовать матрицу размером $k$, то мы также можем использовать матрицу размером $k-1$. Воспользуемся двоичным поиском, будем проверять квадраты со стороной $k$.

Вместо того чтобы разбираться со всеми сложными случаями, возникающими при наложении областей друг на друга, мы решаем задачу в обратном направлении. Что должно быть правдой о поле после последнего применения квадрата? Оно должно содержать квадрат KxK одного типа где-то. А что насчет предпоследнего применения квадрата? Оно должно содержать квадрат KxK одного типа, *если* он не покрыт нашим первым квадратом KxK. Исходя из этого, мы наклеиваем квадраты "в обратном порядке": исходя из желаемой конфигурации, мы находим квадрат KxK и отмечаем его как использованный. Затем мы рассматриваем поле с удаленным квадратом: область под ним теперь может считаться либо единицами, либо двойками по мере необходимости. Мы повторяем эту процедуру, пока либо не покроем все поле, либо не сможем найти больше квадратов KxK для использования. Мы не сможем наклеить матрицу на поле ровно тогда, когда исчерпаем все квадраты BxB для наклеивания с помощью этого метода.

Оказалось, что этот метод довольно неэффективен; он требует $O((mn)^2 logn)$ времени (с некоторым динамическим программированием для ускорения проверки, является ли каждый квадрат KxK допустимым местом для использования). Мы можем немного улучшить это, заметив, что каждый раз, когда мы пытаемся найти квадрат KxK для использования, мы можем просто найти и использовать самый большой оставшийся квадрат в сетке. Это устраняет логарифмический множитель, потому что мы получаем наш ответ, взяв минимальную длину стороны всех выбранных квадратов.

Оба вышеупомянутых решения на самом деле проходят все тесты и укладываются в ограничения. Однако мы можем сделать решение $O((mn)^2 logn)$ еще быстрее, используя структуры для запросов на прямоугольниках. Так можно получить как решения $O(mn^2 logn)$, так и $O(mnlog^3 n)$ с использованием соответствующих структур данных.

# Задача C. Покраска матрицы

Очевидно, что порядок строк не важен, поэтому отсортируем их по значению $k_i$. Теперь можно считать, что у каждой строки есть изначальный цвет — черный или белый, и конечный — противоположный, и $i$-я строка меняет свой цвет при переходе от $k_i$-ого столбца к следующему (бывает $k_i = 0$, тогда изначальным цветом называется не тот, в который покрашена вся строка, а противоположный, при $k_i = m$, аналогично, конечный цвет это тот, в который покрашена вся строка, а изначальный — противоположный ему). Таким образом, есть $n$ событий смены цвета строки, происходящие в моменты времени $k_i$. Тогда обозначим за $f(0)$ количество строк с изначальным белым цветом, за $f(1)$ — количество строк с текущим белым цветом после первого события, и так далее, за $f(n)$ — количество строк с конечным белым цветом. Искомая минимизируемая величина это $A = \sum_{i=0}^{n} f(i)(n - f(i)) \cdot l_i$, где $l_i$ — количество столбцов после события $i$ и до события $(i+1)$, то есть столбцов с $f(i)$ белыми клетками (то есть, $l_i = k_i - k_{i-1}$, где дополнительно определим $k_0 = 0$).

У $f(0), f(1) ... f(n-1), f(n)$ есть свойства:

- $f(i)$ целое, $0 \leqslant f(i) \leqslant n$
- $f(0) + f(n) = n$ (так как все строки меняют цвет)
- $|f(i) - f(i-1)| = 1, 1 \leqslant i \leqslant n$, так как за одно событие одна строка меняет цвет

Утверждается, что по любой функции $f$, удовлетворяющей этим условиям, можно построить покраску с соответствующей $f$. Это нетрудно видеть, так как по $f(i) - f(i-1)$ (это 1 или $-1$) однозначно восстанавливается для $i$-й строки то, как она должна быть покрашена — изначально в белый, или изначально в черный. А далее по этим цветам однозначно определяются $f(0)$ и $f(n)$, которые по второму свойству совпадают с изначальным и конечным количеством строк белого цвета. Более подробное доказательство остается в качестве упражнения читателю.

Для удобства в дальнейшем доказательстве рассмотрим не просто раскраску, минимизирующую искомую величину, но еще и в случае равенства минимизирующую $B = \sum_{i=0}^{n} f(i)(n - f(i))$. Также, будем считать, что $n$ четно, случай нечетного $n$ аналогичен, но несколько более технически сложен. Без ограничения общности можно считать, что $f(0) \leqslant \frac{n}{2}$, поскольку при замене $f$ на $n - f$ ни оптимизируемые величины, ни свойства $f$ не меняются. Выполняются следующие свойства:

- В оптимальной $f$ не бывает такого $0 < i < n$, что $f(i-1) < f(i) > f(i+1)$ и $f(i) \leqslant \frac{n}{2}$. (так как тогда f(i-1)=f(i+1)=f(i)-1, и при замене $f(i)$ на $f(i) - 2$ мы не увеличим $A$ и уменьшим $B$)
- В оптимальной $f$ не бывает такого $0 < i < n$, что $f(i-1) > f(i) < f(i+1)$ и $f(i) \geqslant \frac{n}{2}$. (так как тогда f(i-1)=f(i+1)=f(i)+1, и при замене $f(i)$ на $f(i) + 2$ мы не увеличим $A$ и уменьшим $B$)
- В оптимальной $f$ не может быть таких $0 < i < n$, что $f(i) = \frac{n}{2}$, и $j \neq i$, что $f(j) = \frac{n}{2}$. (Рассмотрим ближайший к $i$ такой $j$. Поскольку не выполнены первые и второе условия для $i$, то одно из $f(i-1)$ и $f(i+1)$ меньше $f(i)$, а второе больше. Тогда, если мы заменим $f$ на $n - f$ на отрезке между $i$ и $j$, то все условия на $f$ все еще будут выполнены, и значения $A$ и $B$ не изменятся, но теперь либо оба $f(i-1)$ и $f(i+1)$ окажутся меньше $f(i)$, либо оба больше, что противоречит первым двум свойствам.)

Из этих свойств следует, что у функции не может быть двух точек $i$ и $j$, что $f(i-1) < f(i) > f(i+1)$ и $f(j-1) < f(j) > f(j+1)$, поскольку по доказанному выше тогда $f(i) > \frac{n}{2}$, и $f(j) > \frac{n}{2}$, но тогда между ними есть точка $l$, что $f(l-1) > f(l) < f(l+1)$, для которой по второму свойству $f(l) < \frac{n}{2}$ и, следовательно, между $i$ и $l$ есть $\frac{n}{2}$ и между $j$ и $l$ есть $\frac{n}{2}$, что противоречит третьему свойству.

Аналогично, у функции не может быть двух точек $i$ и $j$, что $f(i-1) < f(i) > f(i+1)$ и $f(j-1) < f(j) > f(j+1)$. Значит, функция либо один, либо два раза меняет направление роста (убывание на возрастание и наоборот).

Покажем, что она не может два раза менять направление роста. Поскольку $f(0) \leqslant \frac{n}{2}$, то $f(1) > f(0)$ (так как иначе можно заменить $f(0)$ на $f(0) - 2$ и не увеличить $A$ и уменьшить $B$). Значит, функция должна возрастать, потом убывать и потом возрастать. Но тогда в первой точке смены направления роста значение больше $\frac{n}{2}$, а во второй меньше. Значит, между ними есть $\frac{n}{2}$, и есть еще $\frac{n}{2}$ на отрезке между началом и пиком (первой точкой смены направления роста). Противоречит третьему свойству.

Доказали, что у $f$ только одна точка смены направления роста, и она сначала возрастает, а потом убывает (так как $f(0) \leqslant \frac{n}{2}$, и иначе можно заменить $f(0)$ на $f(0) - 2$). В случае нечетного $n$ доказывается такое же утверждение.

Поэтому, для оптимального ответа есть всего $n+1$ вариантов: какой-то префикс строк покрашен изначально в черный, а остальные изначально в белый. Остается только перебрать все эти варианты и посчитать искомую величину в них. Это можно сделать, например, при помощи дерева отрезков на прибавление на отрезке и на сумму на отрезке, если аккуратно посмотреть на изменение искомой величины при смене цвета одной строки. Итоговое время $O(n \log n)$.

# Задача D. Подстрока зеркального отражения

Пусть $a$ — это лексикографически наименьшая буква, которая встречается в строке $S$. Мы хотим максимизировать количество последовательных вхождений буквы $a$ в начале результирующей строки (обозначим это количество как $M$). Пусть $L$ — максимальная длина последовательных вхождений буквы $a$ в строке $SS^R$ (в начале). Тогда мы можем доказать, что $M = 2^{K-1}L$ (но если $2^{K-1}L > n$, то этом случае $M = n$). Это связано с тем, что на каждом шаге мы всегда можем удвоить длину

последовательных вхождений буквы $a$ в строке $SS^R$ (если длина не превышает $n$), и это является оптимальным решением.

Таким образом, в случае, если $2^{K-1}L \geqslant n$, ответом будет конкатенация $n$ раз буквы $a$. В противном случае, чтобы достичь $M = 2^{K-1}L$, у нас есть следующие варианты на каждом шаге. На $i$-м шаге:

- Если $i < K$, мы должны выбрать $S'$ из $SS^R$ таким образом, чтобы $2^{i-1}L$ последовательных вхождений буквы $a$ находились в конце $S'$.
- Если $i = K$, мы должны выбрать $S'$ из $SS^R$ таким образом, чтобы $2^{i-1}L$ последовательных вхождений буквы $a$ находились в начале $S'$.

Обратите внимание, что после того как мы определим выбор $S'$ на первом шаге ($i = 1$), выбор $S'$ в будущем может быть однозначно определен. Существует $O(n)$ вариантов для случая $i = 1$, и мы можем получить результирующую строку для фиксированного выбора за $O(n)$, поэтому это решение работает за $O(n^2)$ времени.

# Задача E. Строка с наименьшим хешем

Для каждой позиции в $s$ у нас было три выбора — взять букву из $a$, из $b$ или из $c$. К сожалению, $3^{28}$ вариантов слишком много, но не очень много — всего $22,876,792,454,961$ возможных строк, что в общем масштабе вещей не так уж и огромно. В подобных задачах мы часто применяем подход Meet-in-the-Middle — делим задачу на две относительно равные части, решаем каждую из них независимо и комбинируем результаты. В данном случае длина половин строки может составлять до 14, таким образом, будет $3^{14} = 4,782,969$ возможных строк, которые можно проверить за гораздо меньшее время, чем за секунду. Если мы разделим данную задачу на две с $n \leqslant 14$, мы сможем решить их довольно быстро.

К сожалению, хеш строки зависит от обеих частей, и минимальный хеш нельзя найти просто найдя минимальный хеш в обеих частях. Необходимо объединить результаты подзадач более умным способом, чтобы получить настоящий ответ.

К счастью для нас, заданная хеш-функция довольно проста, и мы можем это использовать. Представьте, что мы разделили входную строку (длиной $n$) на две строки длиной $L$ и $R$ соответственно ($R$ может быть либо $L$, либо $L - 1$, в зависимости от того, было ли $n$ четным или нечетным). Если у нас есть строка в левой части с хешем $HL$ и строка в правой части с хешем $HR$, то конкатенация этих двух строк будет иметь хеш $HL \cdot 127^R + HR$.

Таким образом, мы можем найти хеши всех возможных строк в левой части и отсортировать их; затем найти хеши всех возможных строк в правой части и отсортировать их, а затем выяснить, какая комбинация хеша из левой части, умноженного на $127^R$, плюс хеш из правой части дает наименьшее значение. Как только мы отсортируем хеши, поиск хеша в правой части для фиксированного хеша в левой части можно легко выполнить с помощью двоичного поиска. Более того, поскольку мы также отсортировали хеши в левой части, мы можем использовать решение с двумя указателями (перемещая указатель в левых или правых массивах хешей, чтобы сохранить результирующий хеш как можно ниже при итерации по всем хешам слева).

Описанное решение имеет сложность $O(3^{\frac{n}{2}} \cdot \frac{n}{2} + 3^{\frac{n}{2}} \cdot log(3^{\frac{n}{2}}))$, то есть $O(3^{\frac{n}{2}} \cdot \frac{n}{2})$.

# Задача F. Бегаем между лекциями

Будем решать задачу динамическим программированием. Будем считать $dp[t][l]$ — минимальное время, в которое можно освободиться в $t$-м корпусе, если посетить $l$ лекций (то есть, $1 \leqslant t \leqslant 2$, $0 \leqslant l \leqslant n$). Есть два варианта для ее пересчетов:

- Предыдущая лекция была в том же корпусе. Обозначим минимальный момент начала лекции в $t$-м корпусе, больший $dp[t][l-1]$, за $\tau$. Тогда, точно пройти $l$-ю лекцию можно только если она начинается не раньше, чем в $\tau$, и эту лекцию можно посетить. Значит, минимальное время освобождения это $\tau + 1$.

- Предыдущая лекция была в другом корпусе. Обозначим минимальный момент начала лекции в $t$-м корпусе, больший $dp[3-t][l-1]+d+k(l-1)$, за $\tau$. Тогда, точно пройти $l$-ю лекцию можно только если она начинается не раньше, чем в $\tau$, и эту лекцию можно посетить. Значит, минимальное время освобождения это $\tau+1$.

Таким образом, беря начальные значения $dp[1][0]=0, dp[2][0]=0$, можно посчитать всю динамику в порядке возрастания $l$. Для пересчетов нужно делать бинарные поиски в массивах времен начал лекций. Итоговое время $O(n\log n)$.

# Задача G. Лабиринт с потайными дверьми

Будем исследовать неизвестное пространство с помощью обхода в глубину. Все текущие знания будем хранить в списке фрагментов карт, то есть списке матриц. Как только мы попадаем в неизвестную дверь, будем создавать новый фрагмент карты. Наше решение будет определять, какие два фрагмента на самом деле являются одним и тем же фрагментом, и объединять их, если это возможно. Объединить два фрагмента — идейно несложная задача, однако она требует аккуратной реализации и продумывания кода. Мы же сосредоточимся на определении логики, когда же мы можем объединить два фрагмента.

Если после каждого очередного провала в дверь мы будем в том же порядке обходить имеющуюся компоненту, то провалившись три раза мы будем ходить по одному и тому же шаблону ABABABAB... AAAAAAA... или BBBBBB... Это значит последний и третий с конца фрагменты можно объединять. Также мы можем объединить две компоненты, если обе имеют две потайные двери, или если потайные двери ведут в одно и то же место (на один и тот же участок карты (очевидно, это возможно, если участки карты, куда ведут эти двери, объединились) и в одно и то же место на этой карте).

Получается, у нас есть правила для объединения компонент. Если их много, то объединяем последнюю и предпредпоследнюю, а еще проверяем условия из предыдущего абзаца. Если мы объединим два фрагмента, любые двери, которые у них находятся в одном и том же положении, должны приводить к одному и тому же фрагменту. В результате обхода всей карты в каждой из этих компонент (и постоянных попыток сливания мы получим не более двух компонент. Хотелось бы, и по логике, должны получить одну, но есть неприятные случаи. Общий шаблон кода этой части:

```
while (true)
    get_response();
    if (fragments.size() >= 4)
        merge_fragments(fragments[fragments.size()-3],
                        fragments[fragments.size()-1],
                        pos());
    while (find_and_apply_merge_between_all());

    if (!explore_fragment()) break;
```

У нас остались две компоненты в двух случаях. Если дверь на самом деле одна, то это исходно неотличимо от ситуации, что у нас две одинаковые компоненты связности. По условию, у нас одна компонента, поэтому мы можем оставить только один фрагмент и одну дверь. Второй случай заключается в том, что невозможно достигнуть обеих дверей из одной точки. По гарантии связности двери должны быть рядом друг с другом в узком коридоре, используйте это, чтобы объединить два фрагмента.

```
######     ######    ##########
#a..B? and ?A...# => #a..BA...#
#S.###     ##..b#    #S.###..b#
######     ######    ##########
```

# Задача H. Кузнечик: туда и обратно

Прежде всего, мы перевернем вторую часть пути. Вместо того чтобы человек поднимался и спускался (используя только набор ступеней, используемых при подъеме), у нас будет два человека,

поднимающихся по лестнице, один за другим. У второго человека все еще есть те же ограничения: он может подниматься на четыре ступени за раз, и он должен использовать только ступени, используемые первым человеком.

Теперь второй трюк: мы можем построить оба пути одновременно. В нашем случае лучше сосредоточиться на человеке с большими размерами ступеней. Всякий раз, когда он поднимается на 1 ступень, другой человек также должен подняться на 1 ступень. Всякий раз, когда он поднимается на 2 ступени, у другого человека есть два варианта: либо подняться на 2 ступени, либо подняться на 1 ступень дважды. Аналогично мы получаем 3 варианта для делающего меньшие шаги, если делающий большие шаги поднимается на 3 ступени ($1 + 1 + 1$, $1 + 2$, $2 + 1$) и 5 вариантов, если делающий большие шаги поднимается на 4 ступени.

Таким образом, мы вывели следующее рекуррентное соотношение: $dp_n = dp_{n-1} + 2dp_{n-2} + 3dp_{n-3} + 5dp_{n-4}$.

# Задача I. Криптография и панграммы

Если вы знаете 26 простых чисел, которые используются, то как дешифровать сообщение? Можно попробовать разделить каждое значение на каждое известное простое число, чтобы найти множители значения. (Обратите внимание, что значение в шифре может быть квадратом простого числа, если в открытом тексте дважды подряд встречается одна и та же буква.)

Дальше мы могли бы сказать, что вторая буква открытого текста — это та, чье простое число появляется как множители как первого, так и второго значений зашифрованного текста, третья буква — это та, чье простое число появляется как множители как второго, так и третьего значений зашифрованного текста, и так далее. Затем оставшиеся множители в первом и последнем значениях зашифрованного текста дают нам первую и последнюю буквы открытого текста.

Это почти правильно, но нам придется иметь дело с одной значительной неприятностью. Если открытый текст начинается с чего-то вроде ABABA..., например, то первое, второе, третье и четвертое значения зашифрованного текста будут одинаковыми, являясь произведением тех же двух множителей (простых чисел, соответствующих A и B). В частности, начало BABAB... выглядит точно так же, как начало ABABA...! Хорошей новостью является то, что мы знаем, что этот тип шаблона должен заканчиваться где-то в сообщении; в конечном итоге мы получим либо одну и ту же букву дважды подряд, либо (поскольку открытый текст использует более двух разных букв) три последовательные разные буквы. Как только происходит что-либо из этого, у нас есть «точка взлома», и мы знаем, какие факторы конкретного значения зашифрованного текста соответствуют какой из двух букв открытого текста, которые он кодирует. Затем мы можем получить оттуда остальную часть открытого текста, работая вперед и назад.

Однако мы должны помнить, что мы не знаем 26 секретных простых чисел! Нам нужно найти способ их получить. Ключевое понимание заключается в том, что любые два последовательных значения в шифре имеют по крайней мере один общий множитель. Разложение на множители очень больших чисел может быть неразрешимым, но мы знаем, как эффективно найти наибольший общий делитель двух очень больших чисел! Метод, подобный алгоритму Евклида, будет достаточно быстрым.

Обратите внимание, что если у нас есть открытый текст, такой как ABABC..., возможно, что простое число, соответствующее A, не появится ни в одном из значений парного НОД. Итак, мы должны вычислять НОД последовательных значений шифра, пока не получим значение, которое не равно 1; по крайней мере одно такое значение должно существовать, как описано во введении к задаче. В этот момент мы можем распаковать оставшуюся часть шифра, как описано ранее, находя все простые числа по мере продвижения. Затем мы можем действовать, как описано выше.

# Задача J. Доставка треугольниками

Задачу решаем динамическим программированием. Сначала решим задачу в случае, если все координаты $y_i$ положительные. Сначала сортируем точки по углу относительно позиции склада. Пусть $dp[l][r]$ — это минимальная длина пути, который курьер должен пройти, чтобы обойти все точки в интервале $[l, r]$, если это возможно, и $\infty$ в противном случае. Пусть мы объединили $l$ в треугольник с некоторым $m$, таким что $l < m < r$, тогда $dp[l][r] = \min_{l<m<r}(dp[l][m] + dp[m + 1][r])$.

Второй вариант — это сделать треугольник с $l$ и $r$. Достаточно проверить, пересекает ли прямая, соединяющая их, отрезок между складом и какой-либо другой точкой. Это можно сделать за $O(n)$. Если не пересекает, у нас есть еще один кандидат для решения, тогда $dp[l][r] = min(dp[l][r], dist(l, r) + dp[l + 1][r - 1])$. Сложность решения $O(n^3)$.

Теперь мы можем обобщить решение на случай, если координаты $y_i$ не положительные. Обратите внимание, что как бы мы ни связывали точки в треугольники, мы всегда можем провести луч из $(0, 0)$, которая не пересечет ни один из треугольников. Это верно, потому что треугольники не пересекаются между собой. "Дуга которую образует один треугольник, может содержаться в другой дуге или содержать в себе другую дугу, но они не могут пересекаться. Теперь сортируем точки по углу. Если предположить, что мы провели этот луч где-то между точками $i$ и $i + 1$, мы можем рассматривать последовательность, которая циклически сдвинута влево на $i$ мест и повторить динамическое программирование, как в предыдущей части. Это решение работает со сложностью $O(n^4)$.

Однако мы можем заметить, что большую часть этого динамического программирования мы вычисляем несколько раз. Мы можем продублировать отсортированный массив, то есть циклически повторить его второй раз и снова вычислить $dp[l][r]$. Обратите внимание, что теперь в $dp[i][i + n]$ находится именно решение, которое получается, когда бы луч находился между планетами $i - 1$ и $i$. Таким образом, получим окончательное решение за $O(n^3)$.

# Задача K. Качалка в двенашке

Поделим веса на 25, и теперь можно считать, что вес это целое неотрицательное число, и оно за один день либо увеличивается на 1, либо теряет 10% (с округлением вниз). Обозначим $\phi(n) = \lfloor \frac{9n}{10} \rfloor$. Обозначим изначальный вес за $N$.

Докажем, что все достижимые веса из веса $N$ после $A + B$ дней, среди которых было $A$ прибавлений единицы и $B$ уменьшений на 10%, это все возможные целые числа из диапазона от $\phi^B(N + A)$ до $\phi^B(N) + A$. Доказывать будем по индукции по $A + B$, база для $A + B = 0$ очевидна.

Докажем переход: если $A = 0$ или $B = 0$, то никакого выбора нет, и утверждение очевидно. Если же $A > 0$ и $B > 0$, то число после $A + B$ операций может быть получено двумя способами:

- Последняя операция — прибавление единицы. Тогда до нее по предположению индукции было $K \in [\phi^B(N+A-1), \phi^B(N)+A-1]$, а после операции будет $(K+1) \in [\phi^B(N+A-1)+1, \phi^B(N)+A]$
- Последняя операция — уменьшение на 10%. Тогда до нее по предположению индукции было $K \in [\phi^{B-1}(N + A), \phi^{B-1}(N) + A]$, а после операции будет $\phi(K) \in [\phi^B(N + A), \phi(\phi^{B-1}(N) + A)]$

При этом, в обоих отрезках выше видно, что все числа достижимы. Остается только заметить, что $\phi(X + 1) \leqslant \phi(X) + 1$, так как $\phi(X + 1) = \lfloor \frac{9(X+1)}{10} \rfloor = \lfloor \frac{9X}{10} + \frac{9}{10} \rfloor \leqslant \lfloor \frac{9X}{10} + 1 \rfloor = \lfloor \frac{9X}{10} \rfloor + 1 = \phi(X) + 1$. Из этого сразу по индукции по $C$ следует, что $\phi^C(X + 1) \leqslant \phi^C(X) + 1$. Затем по индукции по $D$ получаем, что $\phi^C(X + D) \leqslant \phi^C(X) + D$ для произвольных натуральных C, D.

Тогда, $\phi^B(N + A) \leqslant \phi^B(N + A - 1) + 1 = \phi(\phi^{B-1}(N + A - 1)) + 1 \leqslant \phi(\phi^{B-1}(N)+A-1)+1 \leqslant \phi(\phi^{B-1}(N)+A)+1$, а также $\phi(\phi^{B-1}(N)+A) \leqslant \phi(\phi^{B-1}(N))+A = \phi^B(N)+A$. Поэтому, объединение отрезков целых чисел $[\phi^B(N + A - 1) + 1, \phi^B(N) + A]$ и $[\phi^B(N + A), \phi(\phi^{B-1}(N) + A)]$ это отрезок $[\phi^B(N + A), \phi^B(N) + A]$. Это завершает доказательство утверждения.

Остается только понять, что при больших $B$ (таких, что $(\frac{9}{10})^B < \frac{1}{N+K}$) описанный отрезок просто имеет вид $[0, A]$, то есть каждый следующий еще и вложен в предыдущий (так как $A + B = K$ — суммарное количество дней). Поэтому, достаточно перебрать $B$ по возрастанию вплоть до момента, когда впервые $(\frac{9}{10})^B < \frac{1}{N+K}$, и посчитать для них отрезки и проверить, лежит ли в них конечный вес. Итого решение работает $O(log(a + k))$ в терминах изначальных констант.

# Задача L. Подстрока с разным количеством

Пусть $[i, j]$ — искомый отрезок. Покажем, что границы этого отрезка довольно близки к границам всей последовательности. Формально, что $i \leqslant 3$ и $j \geqslant n - 2$.

Из всех оптимальных отрезков выберем тот, у которого $i$ минимально. Пусть утверждение не верно, и в ответе $i > 3$ и $j < n - 2$. Это означает, что с каждой стороны отрезка находятся по крайней мере три блока, которые нельзя добавить к этому отрезку. Рассмотрим несколько случаев.

В отрезке $[i, j]$ находятся элементы только одного типа. Итак, у нас есть $S_i = S_{i+1} = \cdots = S_{j-1} = S_j$. Если $i \neq j$, то добавив элемент $S_{i-1}$, мы также получим корректный интервал Это означает, что отрезок $[i, j]$ не является самым длинным корректным — мы получаем противоречие. Если же $i = j$, то отрезок имеет длину 1. Любой отрезок длиной 1 является корректным, следовательно, корректен также отрезок $[1, 1]$. Мы получаем противоречие, поскольку отрезок $[i, j]$ должен был быть оптимальным решением с наименьшим $i$.

В отрезке $[i, j]$ находятся элементы двух типов. В этом случае мы можем предположить, что в отрезке $[i, j]$ находятся элементы трех типов, при этом элементов одного типа 0, что сводится к следующему случаю, описанному ниже.

В отрезке $[i, j]$ находятся элементы трех типов. Пусть $|X|$ обозначает количество вхождений элементов $X$ в отрезок $[i, j]$. Без потери общности мы можем предположить, что $|A| < |B| < |C|$.

**Пусть элемент $S_{i-1}$ равен $B$**. Тогда плохой случай только если $|B| + 1 = |C|$. Тогда посмотрим на $S_{j+1}$. Либо все стало хорошо, и мы получили противоречие, либо $S_{j+1} = A$. При этом, если $|A| < |B| - 1$, то отрезок $[i, j + 1]$ хороший и мы получили противоречие. Значит, $|A| = x$, $|B| = x + 1$ и $|C| = x + 2$ для некоторого $x$.

Посмотрим теперь на $S_{j+2}$, если это $B$ или $C$, то отрезок $[i, j + 2]$ хороший (дающий противоречие). Значит, $S_{j+2} = A$. Теперь смотрим на $S_{j+3}$, если это $A$ или $C$, то отрезок $[i, j + 3]$ хороший. Значит, $S_{j+3} = B$, и сейчас у нас строка вида

```
??B...AAB
```

Смотрим на $S_{i-2}$. Если там $B$ или $C$, то отрезок $[i - 2, j]$ хороший, значит там $A$. Открываем $S_{i-3}$. Если там $B$ или $C$, то $[i-3, j]$ хороший, значит $S_{i-3} = A$, а это значит, что $[i-3, j+3]$ хороший.

**Пусть элемент $S_{i-1}$ равен $A$**. Тогда плохой случай только если $|A| + 1 = |B|$. Тогда посмотрим на $S_{j+1}$. Если это $B$, то все будет симметрично случаю, когда $S_{i-1}$ равен $B$, а если это $C$, то $[i, j + 1]$ хороший. Значит $S_{j+1} = A$. Тогда или всё уже хорошо, или снова $|A| = x$, $|B| = x + 1$ и $|C| = x + 2$ для некоторого $x$.

```
??A...A??
```

Смотрим на $S_{j+1}$, если это $A$ или $C$, то строка $[i - 1, j + 2]$ хорошая, значит это $B$. Дальше все рассуждения аналогичны предыдущему пункту начиная с момента, когда мы сравняли количества $A$, $B$ и $C$. А исходный случай, когда элемент $S_{i-1}$ равен $C$ тривиален.

Когда мы доказали конструкцию, можно перебрать эти несколько претендентов на ответ и посчитать количество элементов хоть наивным способом.

# Задача M. Домножения и повороты

Сначала отметим, что с помощью вращений строк и столбцов мы можем разместить произвольные элементы в заданных строках или столбцах. Например, если мы хотим разместить элемент в первом столбце, в строке $i$, то

- Если элемент уже находится в первом столбце, мы убираем его оттуда, вращая строку на одну позицию.
- Мы вращаем столбец, содержащую элемент, так чтобы он оказался в $i$-й строке.
- Вращаем строку, содержащую элемент, чтобы он оказался в первом столбце.

Так мы можем разместить любые $n$ элементов в первом столбце или любые $m$ элементов в первой строке. В идеале, мы хотим избавиться от всех отрицательных элементов, но мы должны поменять знак у $an+bm$ элементов. Отсортируем имеющиеся элементы, а дальше посчитаем динамикой какой наибольшей суммы мы можем добиться (переходы это поменять знак последним $n$ или $m$ элементам).

Для каждого элемента нужно три операции, чтобы он встал на место, и одна смена знака.