# Задача A. Получить зачет

Перефразируем условие задачи: дан массив $a$ из $n$ целых чисел, необходимо найти наименьший суффикс массива, что сумма элементов этого суффикса не меньше $T$, или сказать, что такого суффикса нет.

После этого преобразования формулировки решение становится очевидно: необходимо пойти с конца массива, считая сумму элементов суффикса. Если в какой-то момент мы получили сумму не меньшую $T$, то выведем ответ — это будет номер текущей позиции $i$ в 0-индексации, и завершимся. Если мы полностью прошли массив, но не получили сумму, не меньшую $T$, то выведем $-1$ и завершимся.

Обратите внимание, что необходимо аккуратно обрабатывать случаи с $T = 0$ и/или $n = 0$.

# Задача B. Астрономическая сумма

Распишем астрономическую сумму:

$$2 \times A + 2 \times 2 \times A + \cdots + 2^N \times A + A = 2^0 \times A + 2^1 \times A + \cdots + 2^N \times A = A \times (1 + 2 + \cdots + 2^N)$$

Заметим, что астрономическая сумма представляет собой произведение числа $A$ на геометрическую прогрессию с первым членом $b_1 = 1$ и знаменателем $q = 2$. Тогда посчитаем сумму геометрической прогрессии и умножим ее на число $A$, это и будет астрономическая сумма.

Для нахождения суммы геометрической прогрессии воспользуемся формулой $S = \dfrac{b_1(q^N - 1)}{q - 1} = 2^N - 1$.

Заметим, что при $N \geqslant 32$ сумма геометрической прогрессии не влезет в 32-х битный тип данных, поэтому нас просят вывести ответ по модулю $10^9 + 7$ (остаток от деления на это число).

Для этого нам потребуется свойства модульной арифметики:

$$(a + b) \mod M = ((a \mod M) + (b \mod M)) \mod M$$

$$(a \times b) \mod M = ((a \mod M) \times (b \mod M)) \mod M$$

$$(a - b) \mod M = ((a \mod M) - (b \mod M) + M) \mod M$$

С помощью этих свойств и формулы геометрической прогрессии мы можем переписать ответ в форму (пусть $M = 10^9 + 7$ для краткости):

$$((A \mod M) \times (((2^N \mod M) - 1 + M) \mod M)) \mod M$$

Теперь возникает вопрос, как найти $2^N$ по модулю, если при $N \geqslant 64$ 64-х битная арифметика переполнится, а средства длинной арифметики есть не во всех языках? Для этого нужно самостоятельно написать возведение в степень с помощью свойства модульной арифметики. Но тут возникает другая проблема: как сделать это быстро, потому что за $O(N)$ это будет выполняться несколько лет? Воспользуемся идеей бинарного возведения в степень. Пусть нам надо возвести число $X$ в степень $T$. Тогда мы можем возвести $X$ в степень $\frac{T}{2}$ и возвести результат в квадрат: $(X^{\frac{T}{2}})^2 = X^{\frac{T}{2} \times 2} = X^T$, но это работает только в тех случаях, когда $T$ делится на 2. В ином случае можно перейти к задаче возведения $X$ в степень $T - 1$, домножив потом результат на $X$. Так как на каждом шаге мы делим $T$ на 2, то алгоритм дойдет до возведения $X$ в 1 степень (которое уже известно) за $O(\log T)$ шагов. Например, этот алгоритм можно реализовать рекурсивно.

Вспомним, что у нас может возникнуть переполнение и будем в качестве результата возведения возвращать остаток от деления этой степени на $10^9 + 7$. Для этого еще раз воспользуемся свойством модульной арифметики.

Получили, что для решения задачи нужно возвести число 2 в степень $N$ по модулю $10^9 + 7$ методом бинарного возведения в степень, а потом умножить полученное число на A и еще раз взять остаток от деления на $10^9 + 7$.

# Задача C. Юный химик

Для решения задачи достаточно простого моделирования описанных в условии процессов. Нам понадобится структура данных очень похожая на «Стек» чтобы сделать модель пробирки. Вещество,

помещаемое в пробирку, поступает на вершину стека. Взаимодействие может происходить только между двумя верхними элементами стека. После взаимодействия два верхних элемента снимаются со стека, а их место занимает вновь образованное вещество. Моделировать такой «Стек» удобно простым одномерным массивом, считая вершиной стека последний элемент массива.

Остается выбрать структуру данных для хранения возможных результатов взаимодействий. При больших значениях $N$ не получится создавать двумерный массив. Идея хранить все взаимдействия как взвешенный граф, для представления которого используются списки смежности, может проводить на некоторых тестах к превышению лимита времени (если каждый список смежности делать одномерным массивом и часто осуществлять поиск в таком массиве за линейное время).

Наиболее простым и эффективным решением является использование словаря (dict в Python или map в C++). Пусть результат взаимодействия задан тройкой $(A, B, C)$. Ключом в словаре можно делать пару $(A, B)$, а значением $C$. Использование map в C++ позволит извлекать и информацию о наличии взаимодействия за O(log $M$), использование dict в Python — еще быстрее за O(1).

Нужно понимать, что если элемент $A$, взаимодействуя с элементом $B$, образует элемент $C$, то при изменении порядка исходных элементов реакция тоже происходит. Поэтому, считывая тройку $(A, B, C)$, в словарь взаимодействий нужно заносить также тройку $(B, A, C)$.

Чтобы считать несколько слоев одного и того же вещества в пробирке единым слоем достаточно сделать в словаре взаимодействий тройки $(X, X, X)$, которые заставят вещество «прореагировать» с самим собой, создавая себя же. Так будет решена проблема возникновения подряд идущих одинаковых веществ в ответе.

# Задача D. Картинная галерея

Скорректируем исходную длину стены $L$, заменив её на $L - D \cdot (N + 1)$. Если $L - D \cdot (N + 1) <= 0$, то выдадим ответ «NO».

В противном случае попытаемся расположить картины плотно друг к другу так, чтобы их суммарная длина была равна в точности $L - D \cdot (N + 1)$. Если это получится, то выдадим ответ «YES», иначе выдадим «NO».

Будем использовать массив $A$, у которого $N$ строк и $L - D \cdot (N + 1)$ столбцов. Изначально массив $A$ заполнен нулями.

Используя метод динамического программирования, будем заполнять массив $A$ по строкам числами 1,2,3,...,$N$ так, чтобы для всех $k$ и $i$ (где $1 \leqslant k \leqslant N, 1 \leqslant i \leqslant L - D \cdot (N + 1)$) выполнялось равенство $A[k, i] = k$, если первые $k$ картин можно развесить на стене так, чтобы их суммарная длина была равна в точности $i$.

Выполним $N$ шагов.

На первом шаге изменим отдельные элементы 1-й строки массива $A$. А именно: положим $A[1, w[1]] = 1$ и $A[1, h[1]] = 1$, где $w[1]$ и $h[1]$ — это ширина и высота 1-й картины.

На $k$-м шаге (где $1 < k \leqslant N$) изменим отдельные элементы $k$-й строки строки массива $A$. А именно: положим $A[k, i + w[k]] = k$ и $A[k, i + h[k]] = k$, если $A[k - 1, i] = k - 1$, где $w[k]$ и $h[k]$ — это ширина и высота $k$-й картины.

После $N$ шагов сделаем проверку. Если $A[N, L - D \cdot (N + 1)] = N$, то выдадим ответ «YES», иначе выдадим «NO».

Сложность такого решения не превосходит $O(N \cdot L)$.

Замечание: поскольку при заполнении очередной строки массива $A$ нужно знать только её предыдущую строку, то вместо $N$ строк в массиве $A$ можно использовать всего две строки.

**Альтернативное решение**

Давайте сведем исходную задачу к упрощенной вариации задачи о рюкзаке. В этой задаче у нас есть $N$ предметов, $i$-й предмет имеет вес $w_i$ и нам нужно проверить, можно ли взять какое-то подмножество предметов так, чтобы суммарный вес взятых предметов был в точности $X$. Решение данной задачи можно найти в интернете (например, по запросу «0-1 knapsack») и данная задача решается методом динамического программирования за $O(n \cdot X)$.

Аналогично исходному решению, сначала скорректируем исходную длину стены $L$, заменив ее на $L - D \cdot (N + 1)$. Теперь давайте поставим каждую картину наименьшей стороной $min(a_i, b_i)$, так как мы все равно обязаны поставить каждую картину. После этого полчим оставшееся место, которое

нужно заполнить: $L - D \cdot (N+1) - S$, где $S$ — сумма минимальных сторон картин. Если это значение меньше нуля, то выдадим «NO» (так как меньше места мы никак не сможем занять). В противном случае мы можем для каждой картины либо оставить ее в таком положении, либо повернуть и тогда занимаемое пространство увеличится на $max(a_i, b_i) - min(a_i, b_i)$.

Уже начинает вырисовываться задача о рюкзаке, у нас есть $n$ предметов, вес каждого предмета $w_i$ равен $max(a_i, b_i) - min(a_i, b_i)$. Если мы берем предмет, это значит, что мы повернули картину и занимаемое место увеличилось, а если не берем то оставили в исходном положении. $X = L - D \cdot (N+1) - S$, то есть нам нужно заполнить оставшееся пространство.

Сложность решения $O(N \cdot L)$.

# Задача E. Электрический пейзаж

Для начала рассмотрим частичные решения задачи.

В первой подзадаче можно было просто перебрать все возможные варианты расположения гирлянды и выбрать из них наилучший. Нетрудно видеть, что количество вариантов не превышает $11^5 \approx 1.6 \cdot 10^5$.

Во второй подзадаче не было ограничений на размер гирлянды, но было дополнительное условие: $a_i = b_i$ для всех $i$. Заметим, что при таком условии у нас есть ровно один способ расположить гирлянду: для всех $i$ разместить $i$-ю лампочку на высоте $a_i$. Поэтому для решения подзадачи требовалось просто вывести это расположение и посчитать для него ответ.

В третьей подзадаче было другое дополнительное условие: для всех $i$ и $j$ выполнено $a_i \leqslant b_j$. Данное условие эквивалентно условию $\max a_i \leqslant \min b_j$. Обозначим $A = \max a_i$, $B = \min b_j$ (мы знаем, что $A \leqslant B$). Утверждается, что при данном условии можно расположить гирлянду горизонтально на высоте $A$. Действительно, $A \geqslant a_i$ для всех $i$ и $A \leqslant B \leqslant b_j$ для всех $j$. Т. е. такое расположение допустимо. В этом случае, ответ равен 0.

В четвёртой подзадаче уже нет никаких специальных ограничений, поэтому нужно придумать решение для общего случая.

Рассмотрим упрощённую задачу: пусть нам дано $h$ — ограничение на разницу высот двух соседних лампочек, и мы должны проверить, существует ли расположение гирлянды с этим ограничением. Решать её можно следующим образом: будем идти по лампочкам слева направо и для текущей лампочки поддерживать отрезок из всех высот, на которых она может находиться, если она и все лампочки до неё расположены правильно (т. е. каждая в своем отрезке допустимых высот, и разницы высот у всех пар соседних лампочек не превосходят $h$).

Как считать такие отрезки? Для первой лампочки ответ — отрезок $[a_1, b_1]$. Теперь пусть мы знаем ответ для $i$-й лампочки — отрезок $[l_i, r_i]$, тогда ответ для $(i+1)$-й лампочки можно посчитать как $[l_i - h, r_i + h] \cap [a_{i+1}, b_{i+1}]$. Действительно, если $i$-я лампочка может находиться только в отрезке $[l_i, r_i]$, а $(i+1)$-я отличается от неё по высоте не более чем на $h$, то $(i+1)$-я лампочка должна лежать где-то в отрезке $[l_i - h, r_i + h]$. Поскольку у $(i+1)$-й лампочки есть своё ограничение на высоту $[a_{i+1}, b_{i+1}]$, мы должны пересечь полученные отрезки.

Если отрезок для некоторой лампочки получился пустым, ответ на упрощённую задачу отрицательный. Если мы, в итоге, смогли дойти до конца, и все полученные отрезки оказались не пусты, ответ на упрощённую задачу положительный. Доказательство этого факта очень простое — оставим его в качестве упражнения.

Нетрудно видеть, что, во-первых, ответ всегда является целым (поскольку во всех корректных расположениях гирлянды высоты целые), а во-вторых, он не превышает $\max b_i$. Поэтому для решения четвёртой подзадачи можно было перебрать все целочисленные значения $h$, которые не превышают $\max b_i$, проверить их описанным выше алгоритмом и выбрать минимальное подходящее. Сложность такого решения $O(nH)$, где $H$ — ограничение на числа $a_i$ и $b_i$.

Для решения пятой подзадачи нужно заметить, что если нам подходит какое-то значение $h$, то нам подходят и все большие значения. Тогда можно избавиться от перебора всех возможных $h$ и искать ответ с помощью двоичного поиска, используя для проверки описанный выше алгоритм. Сложность решения $O(n \log H)$.

Шестая подзадача отличается от пятой необходимостью восстановить оптимальное расположение гирлянды. Для этого сначала найдём оптимальное значение $h$ таким же способом, как и в

предыдущей подзадаче, а потом запустим описанный выше алгоритм проверки для найденного значения и запомним все отрезки, построенные в ходе этой проверки.

Тогда ответ можно восстановить следующим способом: возьмём произвольную точку на отрезке $[l_n, r_n]$ и скажем, что её координата — это $ans_n$, искомая высота $n$-й лампочки. Для $ans_n$ обязательно найдётся точка из отрезка $[l_{n-1}, r_{n-1}]$, которая отличается от $ans_n$ не более чем на $h$ (мы строили отрезки так, чтобы это было верно). Скажем, что координата найденной точки — это $ans_{n-1}$. И так далее: идём с конца и набираем точки так, чтобы условие не нарушалось. Каждая следующая точка обязательно найдётся, это следует из построения отрезков. Каждая найденная высота будет удовлетворять ограничениям, т. к. для всех $i$ верно, что $ans_i \in [l_i, r_i] \subset [a_i, b_i]$.

Итого, сложность восстановления ответа $O(n)$, а сложность решения всей задачи $O(n \log H)$.