Состав жюри: Базилевич Григорий
Балдин Даниил
Кильметова Алина
Кориненко Матвей
Лизунов Михаил
Овчаров Иван
Разумков Ростислав
Туров Макар
Харгелия Сергей

Спонсоры олимпиады: Белоногов Иван
ЛКЛ

Отдельная благодарность: Шульгина Галина Михайловна

Задача A. Странные формулы

Автор и разработчик: Лизунов Михаил

Заметим, что первая формула превращается в $f(n) = 1$. Вторая же формула более сложная, но входные ограничения дают n не меньше 10, поэтому $g(n) = 2025$.

Для решения этой задачи было достаточно вывести число 2026.

Задача B. Киноплёнка

Автор: Лизунов Михаил
Разработчик: Кориненко Матвей

Для получения 40 баллов за первую подзадачу можно наивно выполнить циклический сдвиг массива кадров вправо на 1 элемент не более n раз, каждый раз проверяя, получилась ли неубывающая последовательность номеров кадров. Сложность решения $\mathcal{O}(n^2)$.

Для полного решения можно пройтись по массиву слева направо, проверяя, что следующий элемент больше текущего либо равен ему. Для последнего элемента массива кадров нужно посмотреть на элемент с индексом 1 как на следующий. Если это условие не выполнилось более чем 1 раз, значит, ответа не существует, последовательность нельзя сдвинуть так, чтобы она была неубывающей. В ином случае правый элемент из пары индексов, для которых это условие не выполнилось, должен стоять на первом месте последовательности (пусть его индекс i). Чтобы его туда подвинуть, нужно сделать циклический сдвиг последовательности вправо на $(n - i + 1) \mod n$ элементов. Это будет решение за $\mathcal{O}(n)$.

Задача C. Женя едет на работу

Автор и разработчик: Базилевич Григорий

Для решения задачи воспользуемся методом динамического программирования. За $dp[i][j]$ обозначим минимальную стоимость поездки до i-й парковки при расходовании j бесплатных поездок.

Пусть последняя поездка начинается из парковки $m < i$.

Если расстояние между парковками не превосходит длину бесплатной поездки, то есть $x_i - x_m \leqslant t$, то можно доехать бесплатно, поэтому стоимость не изменится:

$$dp[i][j] = \min(dp[i][j], dp[m][j-1]).$$

Иначе бесплатная поездка покрывает только расстояние t, а оставшуюся часть необходимо оплатить:

$$dp[i][j] = \min\left(dp[i][j], dp[m][j-1] + (x_i - x_m - t) \cdot p\right).$$

Таким образом, итоговый переход имеет вид:

$$dp[i][j] = \min_{m<i} \begin{cases} dp[m][j-1], & x_i - x_m \leqslant t, \\ dp[m][j-1] + (x_i - x_m - t) \cdot p, & x_i - x_m > t. \end{cases}$$

Базой динамики, очевидно, является $dp[i][0] = x[i] \cdot p$. Пересчет будем выполнять по слоям j, проходя для каждого слоя все i последовательно по возрастанию. Ответ на задачу будет лежать в $dp[n][k]$.

Для первой ветви перехода можно поддерживать минимальные $dp[m][j-1]$ методом «скользящего окна», что возможно, т.к. $x_{i-1} < x_i$ из условия.

Для быстрого вычисления второй ветви произведем некоторые преобразования над выражением:

$$dp[m][j-1] + (x_i - x_m - t) \cdot p = (dp[m][j-1] - (x_m + t) \cdot p) + x_i \cdot p$$

Нетрудно видеть, что для фиксированного i необходимо минимизировать выражение в первых скобках, не зависящее от i, при ограничении на m вида $x_i - x_m > t$. Все эти m были когда-то ранее пройдены, попали в «скользящее окно» и были удалены из него. Значит, этот минимум можно поддерживать при переходе, как и само «скользящее окно».

Очевидно, что «скользящее окно» не влияет на асимптотику, и итоговое решение будет работать за $O(n \times k)$.

Для решения подзадач можно было воспользоваться следующими идеями:

Подзадача 1: ответ есть $x_n \cdot p$ при $k = 0$ и $\max(0, x_n - t) \cdot p$ при $k > 0$.

Подзадача 2: ответ есть $\sum_{i=1}^{n-1} \max(0, x_{i+1} - x_i - t) \cdot p$.

Подзадача 3: тут можно было реализовать требуемую динамику неоптимально либо с усложненным состоянием вида $dp[i][j][k]$, где k — количество оставшихся минут у последней поездки.

Подзадача 4: ответ есть $\max(x_n - t, 0) \cdot p$.

Задача D. Старый дембельмометр

Автор и разработчик: Балдин Даниил

Нам необходимо найти минимальное натуральное число X, которое невозможно представить в виде наименьшего общего кратного (НОК) ни для какого подмножества заданных периодов из массива a.

Сделаем важное наблюдение: искомое число X всегда является либо единицей, либо степенью простого числа (p^k, где p — простое, $k \geqslant 1$). При этом единицей оно является всегда, когда её нет в наборе a, поэтому будем искать решение для p^k.

Предположим, что минимальное недостижимое число $X > 1$ не является степенью простого числа. Тогда его можно разложить на два взаимно простых множителя, каждый из которых строго больше единицы: $X = u \cdot v$, где $\gcd(u, v) = 1$, $u > 1$ и $v > 1$.

Поскольку $u < X$ и $v < X$, а X — самое маленькое число, которое нельзя получить, значит, числа u и v получить можно. Это означает, что существуют какие-то подмножества расписаний S_u и S_v, такие что $\text{НОК}(S_u) = u$ и $\text{НОК}(S_v) = v$.

Если мы возьмем объединение этих двух подмножеств $S_u \cup S_v$, то НОК их элементов будет равен:

$$\text{НОК}(S_u \cup S_v) = \text{НОК}(\text{НОК}(S_u), \text{НОК}(S_v)) = \text{НОК}(u, v)$$

Так как u и v взаимно просты ($\gcd(u, v) = 1$), то $\text{НОК}(u, v) = u \cdot v = X$. Получается, что X получить можно! Мы пришли к противоречию. Следовательно, X обязано быть степенью простого числа.

Чтобы НОК каких-то чисел равнялся строго p^k, эти числа не должны иметь никаких других простых делителей (иначе они попадут в НОК). Значит, все элементы подмножества могут быть

только степенями p от p^0 до p^k. При этом, чтобы максимальная степень простого числа p в НОК была равна k, в подмножестве обязательно должно присутствовать само число p^k.

Число p^k достижимо тогда и только тогда, когда оно напрямую присутствует во входном массиве a.

Тогда давайте для всех простых $p < 2^{20}$ будем по очереди проверять все $p^k \leqslant 2^{20}$ на наличие в множестве, очевидно, что среди всех не присутствующих нас интересует минимальное значение.

Нам необходимо найти все простые числа в искомом диапазоне. Сделать это можно, например, решетом Эратосфена. Верхнее значение выбрано как 2^{20}, так как $10^6 < 2^{20}$, оно является степенью простого, что значит, что ответ не может быть больше него, но и при этом оно не сильно больше, чем максимальное значение n, а значит не ухудшит время работы.

Далее в массиве или же map запомним присутствие числа в a. После чего проверим все возможные p^k и выберем из них наименьшее не присутствующее в множестве.

Асимптотика работы будет $\mathcal{O}(n \log\log n)$ по времени от решета Эратосфена, $\mathcal{O}(n)$ от ввода и $\mathcal{O}(n)$ от перебора p^k. Итого $\mathcal{O}(n \log\log n)$ по времени и $\mathcal{O}(n)$ по памяти.

Задача E. Бусы для Галины Сергеевны

Автор: Харгелия Сергей
Разработчик: Кильметова Алина

Введем стоимость отрезка как:

$$cost(l, r) = \sum_{l \leqslant i < j \leqslant r} (a_i \oplus a_j)$$

В задаче требовалось разбить массив на минимальное количество отрезков так, чтобы для каждого отрезка выполнялось $cost(l, r) \leqslant X$. Т.е. задача сводится к поиску минимального числа отрезков с ограничением на стоимость каждого.

Т.к. $0 \leqslant x \oplus y$, то при увеличении размера отрезка стоимость никогда не уменьшается:

$$cost(l, r+1) = cost(l, r) + \sum_{i=l}^{r} (a_i \oplus a_{r+1}) \geqslant cost(l, r)$$

Таким образом, для каждого возможного начала l можно найти максимальное допустимое r_l, при котором $cost(l, r_l) \leqslant X$.

Аналогичное верно для $cost(l-1, r) \geqslant cost(l, r)$. Из этого следует, что числа r_i упорядочены по неубыванию, т.е. $r_i \leqslant r_{i+1}$.

Поэтому брать для разбиения с позиции l максимальный возможный отрезок можно и оптимально, так как использование более короткого отрезка только уменьшит возможности по разбиению дальше. Докажем от противного. Пусть это не так и для отрезка, начинающегося с l выгоднее взять отрезок $[l,\ t]$, где $t < r_l$. Пусть далее отрезки разбиты каким-то оптимальным образом. Тогда замена первого отрезка на более длинный $[l,\ r_l]$:

- не нарушает ограничение задачи;
- оставляет не больше элементов справа;
- значит, не увеличивает число оставшихся секций (т.к. мы переносим элементы с одной секции в другую и, возможно, удаляем пустые секции).

Следовательно, существует оптимальное решение, использующее максимальный допустимый отрезок.

Таким образом, для решения задачи требовалось по началу отрезка l находить число r_l и переходить к следующему $l' = r_l + 1$.

Подзадачи 1 и 3: При ограничении на $n \leqslant 100$ или $n \leqslant 1000$ выполнять разбиение можно было путем нахождения r_l в лоб — за $\mathcal{O}(n^3)$ или $\mathcal{O}(n^2)$.

Подзадача 4: Из свойств операции побитового исключающего ИЛИ (XOR) известно, что $a_i \oplus a_j = 0$ тогда и только тогда, когда $a_i = a_j$. Значит, для решения этой подзадачи необходимо было посчитать количество позиций i таких, что $a_{i-1} \neq a_i$, и вывести это число, увеличенное на один.

Подзадача 2: Ограничение $a_i \leqslant 1$ дает возможность подсчитывать cnt_0 и cnt_1 — количество встреченных в текущем рассматриваем отрезке нулей и единиц. Таким образом, при увеличении размера отрезка на одну позицию пересчет *cost* был тривиален: в зависимости от числа, добавляемого в отрезок, к *cost* прибавлялось cnt_0 (при добавлении 1) или cnt_1 (при добавлении 0), далее *cnt* так же пересчитывались.

Эта подзадача подводила к полному решению.

Полное решение: Рассмотрим вклад каждого бита независимо.

Если для фиксированного бита b в отрезке есть $cnt_0^{(b)}$ чисел с нулём и $cnt_1^{(b)}$ чисел с единицей, то количество пар, дающих ненулевой XOR на этом бите равно $cnt_0^{(b)} \cdot cnt_1^{(b)}$. Тогда вклад бита в *cost*:

$$(cnt_0^{(b)} \cdot cnt_1^{(b)}) \cdot 2^b$$

Итоговая стоимость:

$$\sum_b cnt_0^{(b)} \cdot cnt_1^{(b)} \cdot 2^b$$

Так как $a_i \leqslant 10^{18}$, то достаточно рассматривать $\lceil log_2(10^{18}) \rceil = 60$ битов.

Разбиваем отрезок, начинающийся в позиции l. Будем хранить $ones[b]$ — количество чисел с установленным битом b в текущем отрезке $[l,\ i]$. Также поддерживаем текущую стоимость отрезка.

При **добавлении числа** x в отрезок для каждого его бита:

- если бит равен 1, то новые пары образуются со всеми нулями, вклад: $(len - ones[b]) \cdot 2^b$, здесь len — длина отрезка до добавления числа;
- иначе новые пары образуются со всеми единицами, вклад в стоимость $ones[b] \cdot 2^b$.

После этого обновляем счётчики.

Как только стоимость превышает X мы отстанавливаемся, обнуляем стоимость, перезаполняем массив $ones[b]$, увеличиваем ответ на один и начинаем в этой позиции новый отрезок.

Каждое число добавляется в отрезок один раз, для каждой операции добавления обрабатывается константное число битов. Сложность решения: $\mathcal{O}(n)$ по времени и памяти.

Ограничение на X (подзазадча 5) позволяло получить баллы за задачу без аккуратной обработки переполнений, которая требовалась в подзадаче 6 (либо использование 128-битного типа данных, например, __int128).