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. 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 B. 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 C. 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).

Задача D. Голубцы

Автор и разработчик: Туров Макар

Научимся решать задачу при фиксированном h. Построим массив b, и теперь нас спрашивают, какой максимальный MEX можно достичь, если можно добавить не более c единиц. Для этого напишем жадный алгоритм. Будем итерироваться по возможному значению MEX от 1 до n. Для

получения MEX $= i$ надо довести какое-то из чисел до i. Выберем среди всех максимальное k, не превосходящее i. Вычтем из c: $i - k$, чтобы учесть добавление единиц, а также выкинем k из кандидатов на увеличение, так как его мы уже увеличили. Если после i-й итерации c стало отрицательным или нет кандидатов на увеличение, то значит, максимальный MEX равен i.

Теперь решим основную задачу.

Переберём все возможные h и для каждого из них найдём максимальный MEX, которого можно достичь.

Изначально посчитаем для каждого MEX минимальное число добавлений, которое требуется, чтобы его получить. Сохраним эту информацию в массив $need$, если MEX достичь невозможно, то сохраняем $+\infty$.

Теперь воспользуемся методом двух указателей. Перебирая h, мы легко можем сохранить количество свободного фарша, пусть это переменная $last$. Тогда, пока MEX $< h$ и $need_{\text{MEX}+1} < last$, увеличиваем MEX, это значит, что текущее h позволяет увеличить MEX на 1, не уменьшая h.

Однако нам потребовалось ограничение на MEX $< h$. Это нужно, так как для бóльших MEX наш предпосчёт $need$ неверен в силу появления новых маленьких значений массива b.

Научимся бороться с ситуацией MEX $\geqslant h$, пусть h' — как раз тот самый индекс h, при котором перестало выполняться неравенство. Для начала решим задачу в лоб при данном $h = h'$. Для этого явно построим массив b по h' и жадно наберём максимальный MEX, пересчитаем ответ для этого массива.

Утверждение: для всех $h < h'$ требуется увеличивать только элементы массива b, которые равны h. Пока что поверим в это утверждение. В таком случае при переборе оставшихся h нам достаточно найти такое максимальное $m = \text{MEX} - 1$, что $need_h + \dfrac{(m-h)\cdot(m-h+1)}{2} \leqslant last$, то есть мы будем увеличивать только числа, равные h, тогда это просто сумма арифметической прогрессии от 0 до $m - h$, а также требуется потратить $need_h$ добавлений фарша, чтобы достичь MEX $= h$, при этом у нас ресурса всего $last$. Дабы найти такое m, можно использовать бинарный поиск.

В итоге мы для каждого h посчитали максимальный достижимый MEX. Выбираем лучшее произведение $h \cdot \text{MEX}$ — это ответ на задачу.

Доказательство утверждения Посмотрим на переход от h к $h - 1$. Заметим, что на шаге $h - 1$ у нас к счёту добавится количество индексов i массива a, для которых $a_i \geqslant h$. Пусть на этом шаге $h-1$ нам потребовалось увеличить какие-то числа, которые $< (h-1)$, но тогда мы их увеличивали и на шаге h, в силу жадности. Тогда по индукции они были увеличены и на шаге h', но при $h = h'$ мы посчитали ответ в лоб, а значит, нашли этот случай, на котором достигается максимальный MEX. Тогда не имеет смысла рассматривать варианты, при которых увеличиваются числа, меньшие h, ч.т.д.

Задача E. Групповой тренинг

Автор:	Кильметова Алина
Разработчик:	Харгелия Сергей

Будем представлять клиентов как вершины графа. Тогда событие первого типа соответствует объединению компонент связности, в которых находятся вершины с номерами a и b. Событие второго типа соответствует следующему запросу: правда ли, что на отрезке $[l, r]$ находится чётное число вершин из каждой компоненты связности графа.

Первые две подзадачи можно решить, просто поддерживая текущие компоненты связности в СНМ. В первой подзадаче нужно пройтись по всем вершинам с l-й по r-ю и посчитать количество вхождений для каждой компоненты. Во второй подзадаче достаточно проверять, что в графе нет компонент связности нечётного размера (их количество можно легко пересчитывать при объединении двух компонент в одну).

В третьей подзадаче гарантируется, что размер каждой компоненты связности не превосходит двух, а значит, нам нужно проверять, что для каждой вершины из отрезка $[l, r]$ для неё есть парная вершина, и она тоже содержится на этом отрезке. Наличие парных вершин (не обязательно в пределах отрезка) можно проверить, поддерживая все одиночные вершины в сете. Пусть теперь мы знаем,

что у каждой вершины из отрезка где-то есть пара, и хотим проверить, что все парные вершины тоже попали в отрезок. Заведём два дерева отрезков: в первом дереве будем хранить информацию о левых вершинах из пар, а во втором — о правых. А именно, в первом дереве для каждой пары (a, b), где $a < b$, будем хранить значение a на позиции b. Во втором дереве, наоборот, будем хранить значение b на позиции a. Тогда достаточно проверить, что минимум в первом дереве на отрезке $[l, r]$ хотя бы l, а максимум во втором дереве на отрезке $[l, r]$ не более r.

В четвёртой подзадаче граф сначала полностью строится, а потом к нему делаются get-запросы. Создадим вспомогательный массив, в котором в i-ю ячейку запишем номер компоненты связности i-й вершины в получившемся графе. Тогда относительно этого массива нам нужно отвечать, правда ли, что на заданном отрезке каждое значение встречается чётное количество раз. На такие запросы можно отвечать с помощью алгоритма Мо, просто поддерживая для каждого значения количество вхождений, а также количество значений с нечётным числом вхождений.

Перейдём к полному решению задачи. Представим, что каждой компоненте связности графа мы сопоставили случайное число. Тогда проверить, что для отрезка вершин каждая компонента связности встречается чётное количество раз, можно, посчитав XOR чисел, соответствующих компонентам вершин на отрезке. Если каждая компонента встречалась чётное число раз, то XOR обязан быть равен нулю. Иначе с большой вероятностью он не равен нулю. Действительно, пусть каждое случайное число выбиралось равновероятно из отрезка $[1, X]$. Тогда $x_1 \oplus \ldots \oplus x_k = 0$ можно переписать как $x_k = x_1 \oplus \ldots \oplus x_{k-1}$. Таким образом, при любых зафиксированных значениях $(x_1, \ldots, x_{k-1})$ нам подходит не более одного значения x_k из отрезка $[1; X]$. Значит, вероятность ошибки можно оценить как $1/X$.

Сначала для каждой компоненты сгенерируем по случайному числу для каждой вершины исходного графа. В любой момент времени каждой компоненте связности будет соответствовать число для её вершины-представителя. Будем поддерживать нужную нам информацию с помощью двух структур данных. Во-первых, нам понадобится дерево отрезков с операцией XOR, в i-й позиции которого мы будем хранить случайное число, соответствующее текущей компоненте связности вершины i. Кроме того, в СНМ для каждой компоненты будем хранить список вершин, принадлежащих ей. Тогда при объединении двух компонент мы можем сделать переливания от меньшей компоненты к большей. Для каждой вершины из компоненты меньшего размера нам понадобится изменить её значение в дереве отрезков.

Так как за всё время работы каждая вершина будет переливаться не более чем $\log_2 n$ раз, к ДО будет сделано $\mathcal{O}(n \log n)$ запросов изменения. Итоговая асимптотика составит $\mathcal{O}(n \log^2 n)$.