Разбор задачи «Новые технологии»

Будем заполнять исходный массив последовательно, поддерживая последнее известное значение суммы. Если $p_i = -1$, то положим $a_i = m$, потому что меньшее значение записать нельзя, а большее возможно помешает в дальнейшем.

При этом, если $p_i \neq -1$, то положим $a_i = p_i - pref\_sum$, где $pref\_sum$ — сумма уже заполненных элементов a. Если в таком случае, a_i оказалось меньше m, то решения не существует, так как $pref\_sum$ минимально возможный по построению.

Итоговая сложность представленного решения $O(n)$.

Разбор задачи «Не так грубо!»

Первая подзадача может быть решена простым разбором случаев.

Вторая подзадача решается простой реализацией того, что описано в условии: переберем границы выбранной подстроки и проверим, что она подходит, перебрав все пары символов в ней. Из всех подходящих подстрок выберем самую длинную. Асимптотика времени работы решения составляет $O(n^4)$.

Научимся считать количество пар символов, первый из которых равен «a», а второй — «b», за линейное время. Для этого переберем символ строки, и если он равен «b», то прибавим к ответу количество символов, равных «a», стоящих до него. Заметим, что эту величину можно не считать каждый раз заново, а последовательно идти по символам строки, храня количество пройденных символов «a», и если очередной символ равен «b», то прибавлять к ответу это количество.

Используя этот метод для поверки каждой из подстрок, получаем решение за $O(n^3)$.

Чтобы еще ускорить это решение, заметим, что если зафиксировать начало подстроки и перебирать ее конец в порядке возрастания индекса, то не нужно каждый раз заново запускать алгоритм подсчета количества искомых пар символов, ведь необходимые значения для подстроки $s_l s_{l+1} \dots s_r$ можно получить из значений для подстроки $s_l s_{l+1} \dots s_{r-1}$ так же, как это делается в алгоритме. Асимптотика времени работы решения — $O(n^2)$.

Для того, чтобы получить полное решение задачи, воспользуемся методом двух указателей. Для каждого левого конца подстроки найдем максимальный из правых концов, для которых соответствующая подстрока имеет грубость не более c. Заметим, что при увеличении индекса левой границы индекс правой границы не будет уменьшаться. Таким образом, мы можем поддерживать текущие левую и правую границы, сдвигать левую на один символ вправо, а далее расширять подстроку, пока это возможно. Для этого нам надо научиться пересчитывать количество искомых пар символов при добавлении символа в конец строки и удалении символа из начала строки.

Давайте модифицируем для этого алгоритм нахождения количества искомых пар, описанный выше. Будем хранить количество искомых пар, количество символов «a», которое содержит текущая подстрока, а также количество символов «b», которое она содержит. Тогда при добавлении символа в конец строки, если это символ «b», надо увеличить количество искомых пар на количество символов «a», а при удалении, если это символ «a», надо уменьшить количество искомых пар на количество символов «a».

Асимптотика данного решения — $O(n)$.

Разбор задачи «Вафелька»

Формализуем задачу, у нас дано две строки состоящих из 0 и 1 (0 — нет шоколада, 1 — есть), нужно найти количество позиций, в которых можно "приложить"вторую строку к первой. Вторую строку можно приложить начиная с данной позиции первой строки, если каждой единице из второй строки будет соответствовать 1 из первой строки.

В первой подзадаче требовалось просто написать то, что написано в условии.

Для того, чтобы решить вторую подзадачу, требовалось заметить, что для того, чтобы проверить возможность приложить вторую строку в данной позиции, нужно фактически сделать битовое "и между ней, сдвинутой на нужное число позиций, и первой строкой и проверить, что число единиц в нем равно числу единиц во второй строке. Такие операции умеет делать структура данных "bitset за $O(n/w)$, где n — длина строки, а w — размер машинного слова. Итого решение работает за $O(n * n/w)$, где $w = 64$ на большинстве современых компьютеров.

Для того, чтобы получить полный балл по данной задаче, нужно было обратить внимание, что вторая строка имеет по условию специфический вид. Чтобы проверить, что ее можно приложить на позиции i, нужно проверить, что в первой строке на позициях $i \dots i+a-1, i+a+b \dots i+a+b+a-1, \dots, i+(m-1)\cdot(a+b), \dots, i+(m-1)\cdot(a+b)+a-1$ находятся единицы. Но это означает, что надо проверить, что начиная с позиций $i, \dots, i+(m-1)\cdot(a+b)$ идет подряд хотя бы a единиц. Заведем массив, где в позиции хранится 1, если начиная с нее a позиций в первой строке содержат единицы и ноль иначе. Итого нам надо проверить что сумма чисел на позициях $i, i+a+b, \dots, i+(m-1)\cdot(a+b)$, равна m. поскольку расстояния между позициями, на которых мы берем сумму, равны, то если мы разобьем все позиции на классы, то в зависимости от остатка по модулю $a+b$, фактически надо будет брать сумму на отрезке, а это стандартная задача. Итоговая асимптотика $O(n)$.

Разбор задачи «Урок арифметики»

Будем поддерживать множество битов, которые одинаковы у всех чисел. Изначально это множество пустое. Заметим, что это множество изменяется не более $\log(2^{20}) = 20$ раз, потому что когда какой-то бит у всех чисел стал одинаковым, он никогда не станет вновь разным. Изменять это множество может только запрос `and` — он выставляет значения некоторых битов у всех чисел равными 0. Пусть $mask$ — маска битов, которые одинаковы у всех чисел. Обозначим за cur_i i-е число, в котором стоит 0 на тех позициях, где в $mask$ стоит 1. Тогда будем отдельно хранить значение общей для всех чисел части b, отдельно a_i — массив чисел длины n, и отдельно c — число, такое, что a_i `xor` $c = cur_i$.

Теперь, когда поступает запрос `and`, если он не изменяет множество битов, одинаковых для всех чисел, то он изменяет только b. Это изменение можно обработать за $O(1)$. А именно, $b = b$ `and` x, где x — число из запроса. Если текущий запрос `and` изменяет множество битов, одинаковых для всех чисел, пробежимся по всем числам и суммарно за $O(n)$ обновим все a_i ($a_i = (a_i$ `xor` $c)$ `and` x). А также, обновим b, $mask$ и c ($b = b$ `and` x, $mask = mask$ `or` (`not` x), $c = 0$).

Если поступает запрос `xor`, мы сразу можем обновить значение b, $b = b$; `xor` (x `and` $mask$). А значения a_i мы оставим без изменения, но сделаем $c = c$ `xor` (x `and` (`not` $mask$)).

Когда поступает запрос '`?`', нужно найти количество чисел a_i, таких что $l - b \leqslant (a_i$ `xor` $c) \leqslant r - b$. Для этого, будем поддерживать дерево отрезков. Если бы c было равно 0, то требовалось бы просто найти сумму на отрезке значений с $l - b$ по $r - b$. Но когда c не равно 0, заметим, что нам нужно находить сумму на отрезке массива, к которому предварительно применили перестановку $p_i = i$ `xor` c. Применение такой перестановки к листам полного двоичного дерева равносильно изменению порядка детей у некоторых вершин. А именно, если у c i-й бит равен 1, нужно поменять порядок детей у всех вершин на глубине i. Конечно, явно менять порядок детей не нужно. Нужно лишь в функции, которая считает сумму на отрезке, в таком случае, вместо левого сына идти в правого, а вместо правого в левого. Эта модификация дерева отрезков не изменяет его время работы.

Итоговая асимптотика складывается из $O(\log(n))$ перестроений дерева отрезков, каждое из которых работает за $O(n)$. И $O(q \cdot \log(n))$ на все запросы к дереву отрезков.