Задача A. Спортивные программисты

Давайте проведём два наблюдения которые приводят к решению этой задачи:

1. Очевидно что если в итоговом массиве для позиции j существует такая позиция i что $a_i > a_j$ и $i < j$ то бегун j обязательно должен был остановиться.

2. Никогда не выгодно останавливаться более чем 1 раз. Данное утверждение легко доказывается конструктивно. Давайте возьмём участника который прибыл последним. Он остановится и будет ждать пока его не обгонят все люди за ним. Предпоследний прибывший ученик остановится и будет ждать пока его не обгонят все люди кроме последнего. И аналогичное действие проделает каждый бегун в порядке от последнего к первому.

Если мы обозначим p_i как количество остановок бегуна i, то можно будет описать данные наблюдения более формально. В оптимальном ответе выполняются: $\forall i((p_i > 0 : \exists j((a_j > a_i) \wedge (j < i))) \wedge (p_i \leqslant 1))$

Используя второе наблюдения можно понять что нужно посчитать минимальное количество людей которые останавливались хотя бы один раз. Используя первое наблюдение можно посчитать это количество в тупую за время $O(N^2)$. Но храня значение $mx_i = max(a_1, a_2, \ldots, a_i)$, мы можем проверять остановится ли бегун i за O(1). Следовательно проверка каждого бегуна займёт $O(N)$.

Задача B. Уникальная задача

Для решения данной задачи нужно было придумать динамику $dp_{i,j}$, данная динамика обозначает количество различных разбиений первых i элементов на j отрезков.

Давайте распишем переходы и базу данной динамики :

$$dp_{0,0} = 1.$$

$$\texttt{If}(a_i == b_j) dp_{i,j} = \sum_{k=prev[i]}^{k<i} dp_{k,j-1}, \texttt{else if(j > 0)} dp_{i,j} = dp_{i-1,j}$$

.

В данном случае $prev[i]$ это ближайший элемент j перед i который равен a_i, если такого элемента не существует то $prev[i] = 0$. Наивная реализация данной динамики будет работать примерно $O(N * (N + M))$ и получит 60 баллов из 100 возможных. Данное время работы получилось из-за того что все элементы в массиве b уникальны и следовательно первое условие запустится максимум один раз для одного i, то есть для i однозначно определяется j, такое что $a_i = b_j$.

Мы можем оптимизировать решение путём применения префиксных сумм для быстрого нахождения переходов в динамике ($pref_{i,j}$ = $pref_{i-1,j} + dp_{i,j}$, $pref_{i-1,j-1} - pref_{prev_i-1,j-1}$ = $\sum_{k=prev[i]}^{k<i} dp_{k,j-1}$). Также можно оптимизировать память храня динамику послойно dp_j будет обозначать количество способов разбить элементы до текущего на j отрезков. Благодаря этим преобразованиям теперь мы можем подсчитывать динамику за $O(N)$ времени и $O(M)$ памяти, однако появляется проблема в том что мы должны считать префикс суммы, а для их подсчёта мы используем $O(N * M)$ времени и памяти.

Остаётся только понять что на самом деле в динамике нам нужно будет использовать только $2 * N$ значений из префиксных сумм. И мы можем узнать позиции этих значений ещё до того как начать подсчёт динамики. Для i-о элемента эти позиции будут равны $[i-1, j-1]$ и $[prev_i - 1, j-1]$ $(b_j = a_i)$. Давайте запомним данные

позиции и при приходе в них будем запоминать нужную префикс сумму. Теперь нам остаётся научиться узнавать префиксную сумму на позиции без подсчёта всех сумм до неё. Давайте хранить три значения для каждого j $[sum_j, lastupdate_j, cur_j]$, sum_j = $pref_{lastupdate_j-1,j}$, а cur_j это значение dp_j на данный момент, $lastupdate_j$ это позиция последнего обновления dp_j. Теперь мы можем легко узнавать сумму до текущего i зная три этих значения : $pref_{i,j} = sum_j + (i - lastupdate_j + 1) * cur_j$. Данные значения изменяются только при изменении dp_j, следовательно изменять мы их будем только n раз. Теперь мы умеем запоминать значения префикс суммы в нужных позициях за $O(N)$. Итоговое время работы программы $O(N + M)$.

Задача C. Восстановление строки

Каждый запрос можно разбить на два запросы длины 2^k, где 2^k это максимальное натуральное число меньшее чем x. Тогда вместо запроса вида $[l, L, x]$, мы получим запросы $[l, L, 2^k]$ и $[l + x - 2^k, L + x - 2^k, 2^k]$. Разделим все запросы таким образом.

Будем идти по уменьшению k, до тех пор пока $k > 0$. Когда мы зафиксируем k давайте получим все запросы вида $[l, L, 2^k]$ и построим граф в котором соединены вершины l и L. В каждой компоненте получившегося графа построим любое остовное дерево, рёбра которые не вошли в остовное дерево не играют роли, так как и без этих рёбер их компоненты остаются связными. Оставшееся количество рёбер не будет превышать N.Если у нас в остове осталось ребро которое обозначает запрос $[l, L, 2^k]$ разделим его на два запроса $[l, L, 2^{k-1}]$ и $[l + 2^{k-1}, L + 2^{k-1}, 2^{k-1}]$.

После того как мы всё сделаем у нас останется граф в котором не больше $M + 2 * N$ рёбер. Суммарное время данной части решения будет $O(N * logN)$, так как на каждом слое одновременно будет не больше $2 * N + M$ рёбер, А остов можно построить за линейное время.

Теперь нужно заполнить вопросительные знаки. Если в компоненте с вопросительным знаком больше двух букв то ответ -1, если одна буква то вставим эту букву вместо вопросительного знака, если нет букв то вставим букву $'a'$ вместо вопросительного знака. Проверив все компоненты выведем строку как ответ, если раньше мы не выводили -1. Общее время работы решения $O(N * logN)$

P.S. Есть ещё одно решение за $O(N * log^2(N))$, в нём используется переливайка, фенвик, хэши, СНМ и бинпоиск.

Автор : Сахмолдин Мухаммадариф