Задача A. Лунки

Разделим ряд на блоки длины $= \sqrt{N}$ подряд идущих лунок. Если N не является полным квадратом, то $= \sqrt{N}$, округленное вниз. Для каждой лунки помимо силы ее выброса (`power[i]`) запомним первую лунку из другого блока, в которую из нее можно попасть при помощи последовательных прыжков (будем считать что за краем ряда находится фиктивная лунка из отдельного блока, в которую мы попадаем при вылете за край из любой лунки) — `next[i]`. Так же для каждой лунки запомним количество прыжков, требуемое для того чтобы попасть в первую лунку из другого блока — `count[i]`.

Для того чтобы обработать запрос вбрасывания шарика, просто из лунки i будем перепрыгивать сразу в `next[i]`, прибавляя к ответу `count[i]`, пока не придем в фиктивную лунку. Таким образом мы сделаем не более N/K прыжков. Для обработки запроса изменения силы лунки i, в начале поменяем `power[i]`, `count[i]` и `next[i]`, а затем для всех лунок из того же блока что и i, которые находятся раньше i в порядке убывания обновим `next` и `count`. Таким образом мы сделаем не более обновлений. Сложность алгоритма выходит $\mathcal{O}\left(N\sqrt{N}\right)$.

Задача D. Сычи и ACM

Будем называть вершину тяжёлой, если она соединена с более чем $\sqrt{n}$ другими вершинами, и лёгкой в противном случае.

Очевидно, что тяжёлых вершин будет $\mathcal{O}(\sqrt{n})$. Теперь найдём количество треугольников из трёх лёгких вершин. Для этого зафиксируем ребро, соединяющее две лёгкие вершины a и b и найдём за $\mathcal{O}(dega + degb) = \mathcal{O}(\sqrt{n})$ количество лёгких вершин c, смежных с обоими вершинами.

Теперь найдём количество треугльников, содержащих хотя бы одну тяжёлую вершину. Для этого зафиксируем тяжёлую вершину a (всего их $\mathcal{O}(\sqrt{n})$) и отметим в графе все смежные с ней вершины за $\mathcal{O}(m)$. Теперь мы хотим найти количество треугольников, содержащих в себе эту тяжёлую вершину. Для этого за линию переберём все рёбра в графе $b - c$ и проверим, что и вершина b, и вершина c отмечены, то есть, они соединены с вершиной a. Заметим тогда, что треугольник с одной тяжёлой вершиной мы таким образом посчитаем единожды, с двумы – дважды, а стремя – трижды.

Задача G. Различные числа

Для каждой ячейки массива i есть ближайшее справа число с таким же значеним: next [i] > i, a [next [i]] == a [i]. Чтобы посчитать количество различных чисел на отрезке $[l..r]$, нам нужно посчитать количество таких $i : l \leqslant i \leqslant r$ и $\text{next}_i > r$

Решение в offline: переберём r в порядке возрастания, будем поддерживать множество таких $i : i \leqslant r < \text{next}_i$

Получили решение за $\mathcal{O}((n + m) \log n)$, где $n-$ длина массива, $m-$ количество запросов.

Решение в online: в offline мы решили задачу сканирующей прямой с деревом Фенвика (или деревом отрезков). Сделаем дерево Фенвика (дерево отрезков) персистентным. Сохраним все версии. Ответ на запрос $[l..r]$ равен $\text{tree}_r.get(1)$.

Получили решение за $\mathcal{O}(n \log n)$ времени и памяти на предподсчёт и $\mathcal{O}(\log n)$ на запрос.

Решение алгоритмом МО: напишем алгоритм МО (`https://codeforces.com/blog/entry/7383`).

Задача I. Варенье

Разобьем все запросы на кусочки по k запросов. Выполним k запросов с помощью "сканирующей точки"следующим образом: прибавим в специальный массив в точке начала отрезка, к которому прибавляют прогрессию x, а в точке конца $-x$. Пойдем слева направо по всему массиву и будем поддерживать прибавляемое к данному элементу значение. После того, как мы пересчитали все эти значения, нам надо посмотреть, для каких новых i значения стали больше b_i. Для них нам надо по отдельности проделать каждую из этих k операций, чтобы понять, после какой именно стало больше. Таким образом алгоритм работает за $\mathcal{O}(n^2/k + nk)$, что равно $\mathcal{O}(n\sqrt{n})$, если взять $k = \sqrt{n}$.