АиСД_дз4

CovarianceMomentum

1term

1

Значит так, это вроде просто бинпоиск. Давайте что-нибудь про него докажем, что ли. Делает он следующее:

1. Положим $r = 1$;
2. Пока r-тый элемент массива меньше, чем x, умножаем r на 2.
3. Запускается от полуинтервала $[\frac{r}{2};\ r)$;
4. Проверяет, в какой половине отрезка находится искомый элемент;
5. Повторяем 3-5 шаги для этой половинки отрезка;
6. Когда на отрезке всего один элемент, мы нашли то, что нам надо.

В результате первых двух шагов мы сделаем не больше, чем $\log p$ шагов, так как в результате их имеем $\frac{r}{2} < p < r$. Тогда шаги с третьего по шестой суммарно займут $\mathcal{O}(\log \frac{r}{2}) \leq \mathcal{O}(\log p)$. Значит, суммарная асимптотика алгоритма будет равна $\mathcal{O}(\log p)$.

Ниже приведён код с реализацией этого алгоритма.

```cpp
#include <bits/stdc++.h>
using namespace std;

const int INF = 2147483647;

int main() {
    int n, x;
    cin >> n >> x;
    int arr[n];
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    int l = 0, m, r = 1;

    while (arr[r - 1] < x && r < n) {
        r*=2;
    }
    if (r >= n) {
        r = n;
    }

    l = r/2;

    while (l + 1 < r) {
        m = (l + r)/2;
        if (arr[m] > x) {
            r = m;
        } else {
            l = m;
        }
    }
    cout << l + 1;
    return 0;
}
```

1.cpp

2

Воспользуемся методом двух указателей. Будем поддерживать l и r — левую и правую границы рабочего отрезка соответственно, а так же $cnt[]$ — количества элементов каждого типа на рабочем отрезке. Теперь опишем сам алгоритм:

1. Двигаем правую границу массива, пока на нём меньше k различных элементов: когда проходим по элементу, увеличиваем его счётчик на 1, если раньше счётчик был 0, то количество различных на рабочем отрезке увеличилось на 1.
2. Записываем позицию, на которой количество различных элементов на отрезке впервые стало равно k — $leftBorder$;
3. Идём дальше по массиву, пока количество различных элементов равно k, записывая последнюю позицию, на которой это выполняется — $rightBorder$. Сейчас в массиве $cnt[]$ лежат элементы с отрезка $[l;\ rightBorder]$.
4. После этого начинаем двигать l направо пока количество различных элементов равно k. Эта часть делится на два этапа — до тех пор, пока мы не дошли до $leftBorder$.

То, что этот алгоритм работает за $\mathcal{O}(n)$ очевидно — оба указателя проходят по каждому элементу не больше одного раза. Почему он находит ответ правильно, тоже очевидно — он находит *максимальный* отрезок, на котором ровно k различных элементов, и добавляет.

А пусть он ничего не добавляет, решение-то неправильное. Давайте нормальное покажу.

Давайте решать немного другую задачу — будем считать, сколько всего есть непустых подотрезков, на которых различных элементов *не больше*, чем k. Назовём их количество S_k. Это довольно просто, и делается теми же двумя указателями, поддерживающими всё тот же массив $cnt[]$. Будем для каждой левой границы находить все правые границы, которые нам подходят:

1. Двигаем правую границу, пока у нас на отрезке не окажется $> k$ различных элементов.
2. Поймём, что теперь $[l; r)$ и все подотрезки вида $[l; r']$, где $r' \leq r$. Добавим их количество к ответу. Всего их будет $r - l$.
3. Сдвинем левую границу на один и перейдём к пункту 1.

Для того, чтобы вернуться к исходной задаче, нам достаточно посчитать S_k и S_{k-1}, тогда искомое значение просто будет равно $S_k - S_{k-1}$, так как первое включает в себя все отрезки с не менее, чем k элементами, а второе — с не более, чем $k - 1$, значит их разницу составляют только отрезки с ровно k элементами. Ассимптотикой этого решения будет $\mathcal{O}(n)$, так как мы совершаем два прохода двумя указателями по массиву, а это не больше, чем $4n$ итераций алгоритма.

P.S.: для $k = 1$ достаточно посчитать только S_1, так как оно и будет ответом. А S_0 не существует.

3

НУО $n < m$. Пусть меньший массив называется $a[n]$, больший — $b[m]$. Начнем связывать факты. Пусть в слиянии наших массивов до k-ой позиции стоит префикс из i элементов $a[]$ и префикс из $k - i$ элементов $b[]$. Теперь поймём, может ли такой префикс **на самом деле** быть в слиянии. Это будет означать, что выполнено одно из двух условий:

- $b_{k-i} \leq a_i \leq b_{k-i+1}$
- $a_i \leq b_{k-i} \leq a_{i+1}$

Чтобы не заморачиваться с концами массива, будем считать $a[n]$ и $b[m]$ максимальным значением нашего типа данных. Если таких нет, ну, придётся ручками рассматривать, но это мелочи.

Почему нужно, чтобы выполнялись именно эти условия? Ну, они просто означают, что именно конкретный элемент (в первом случае a_i, во втором — b_{k-i}) корректно завершает префикс слияния.

Окей, теперь мы умеем проверять, подходит нам такое i или нет. Теперь, в случае если оно не подходит, научимся понимать, меньше или больше настоящее i, чем текущее. Опять же, проверяем несколько условий:

- Если $a_{i+1} \leq b_{k-i}$, то i слишком маленькое.
- Если $b_{k-i+1} \leq a_i$, то i слишком большое.

Опять же, понятно, почему условия именно такие — если последний элемент префикса массива $a[]$ слишком большой, то надо взять префикс поменьше, а если последний элемент префикса массива $b[]$ слишком большой, то надо взять префикс $a[]$ побольше.

Теперь запустим бинпоиск по массиву $a[]$, который будет искать такой номер i, что соответствующие ему префиксы легитимно сливаются. Условие вызода указано выше, правило выбора, в какую половину рабочего отрезка переходить — тоже. Бинпоиск работает за $\mathcal{O}(\log n)$, а так как мы приняли $n < m$, то это именно то, что нам нужно.