Студент: Маргарита Чудова
Группа: 2
Дата: 27 апреля 2022 г.

1. Пусть в нашем отсортированном массиве $O(n)$ элементов. АВЛ дерево — это двоичное дерево поиска, то есть дерево, в котором левый ребенок меньше или равен родителя, а правый больше или равен.
   Можно построить процедуру, которая поддерживает этот инвариант. Будем в нашем отсортированном массиве искать медиану за $\mathcal{O}(1)$ (учитывая, что массив отсортированный, это легко сделать, достаточно взять элемент номер $n//2$), ее значение кладем в корень нашего дерева. Потом в кажом из получившихся «подмассивов», на которые медиана разделила наш массив тоже находим медианы — это будут левый и правый ребенок соответственно для левого и правого подмассива. Делаем так, пока элементы не закончатся. Операций всего n, поэтому потратим $\mathcal{O}(n)$ времени
   Почему получится именно АВЛ-дерево, будет ли соблюдаться условие на глубину поддеревьев? Рассмотрим некоторый подмассив, который мы делим медианой пополам. Понятно, что левая половина подмассива — это левая поддерево на некотором шаге, а правая — правое по построению. При этом эти половины не будут отличаться друг от друга больше, чем на 1, потому что медиана либо делит подмассив на две равные половины в случае нечетного количества элементов, либо на части, количество элементов в которых отличается на 1 в случае четного количества элементов.

2. Мы умеем объединять отсортированные массивы за $\mathcal{O}(n_1+n_2)$, и при этом умеем получать отсортированный массив из АВЛ-дерева за $\mathcal{O}(size(T))$. Сделаем так: получим два отсортированных массива из деревьев $\mathcal{O}(size(T_1) + size(T_2))$, сделаем из них один отсортированный массив, а потом с помощью только что решенной первой задачи за $\mathcal{O}(size(T_1) + size(T_2))$ слепим из него АВЛ-дерево.

3. Будем, как и на семинаре, считать, что в каждой вершине записаны размеры поддерева. Если мы создаем дерево с помощью энного количесва функций add, то нужно просто завести атрибут size для каждой вершины, и, при добавлении новой вершины, для всех вершин v_i на пути до ее места прибалять к $v_i.size$ единицу, у самой же добавленной вершины $v.size = 1$. На асимптотику это все особо не повлияет.

   Перейдем к *get_position*. Будем искать элемент, равный x так же, как мы делали в фунции find, но параллельно с этим будем для всех вершин, по которым идем, считать pos — позицию элемента в упорядоченном массиве значений дерева.
   Как ее считать? Для корня позиция элемента — это количество элементов в левом поддереве плюс один

$$pos[root] = root.left.size + 1$$

   Допустим, мы спуслились в какую-то вершину v и через нее хотим посчитать позиции левого и правого ребенка.Для левого нам надо учесть, что все элементы правее него больше, чем он, при этом были меьше, чем родитель, и при этом за счет спуска мы уменьшили позицию на 1. То есть:

$$pos[v.left] = pos[v] - v.left.right.size - 1$$

   Для правого по той же логике:

$$pos[v.right] = pos[v] + v.right.left.size + 1$$

   Таким образом, когда находим $v.value = x$, возращаем $pos[x]$. Работает это, как и find, за $\mathcal{O}(\log n)$

4. Сначала найдем корень первого дерева во втором, пусть это будет вершина v. Чтобы сделать деревья эквивалентными, нужно для начала «подвесить» второе дерево за корень первого.
   Сделаем это следующим образом: если v является левым ребенком, сделаем малое левое вращение от ее родителя, чтобы она встала на его место, если правым — правое. Будем повторять до тех пор, пока она не станет корнем.
   Вращение поддерева не меняет связей вне этого поддерева, поэтому мы можем перейти в преобразованном дереве от корня к левому и правому поддеревьям и проделать там все то же самое. Продолжаем до тех пор, пока не получим из второго дерева первое.