Homework

Panov Ivan, M3139

01-11-2019

Задача 1

Требуется найти k-ую порядковую статистику при помощи кучи. Заведем дополнительную кучу, в которой будем хранить пары значений: key - значение элемента, $index$ - индекс в изначальной куче. Положим в новую кучу значение корня изначальной и ее индекс в куче($index = 0$, так как корень). Проведем такую операцию: добавим в новую кучу детей текущего корня из старой кучи и удалим корень в новой, найти детей мы можем за $\mathcal{O}(1)$, так как знаем индекс их отца. Повторим такую операцию $k-1$ раз. После выполнения значение в корень и будет являться k-ой порядковой статистикой. Докажем коректность алгоритма. Пусть было проделано какое-то количество операций. Заметим, что следующий по минимальности элемент, который не лежит в новой куче, может являться только сыном одного из элементов, уже находящихся в куче(из свойств кучи). Так же заметим, что этот элемент будет добавлен в кучу, не позже чем будут удалены элементы с меньшими значениями, так как не может произойти такого, что какая-то новая добавленная вершина будет рассмотрена раньше отца следующего по минимальности(новая вершина точно больше следующей по минимальности, из этого следует, что она больше и отца следующей по минимальности вершины). Из этого следует, что вершины попадают в корень в порядке сортировки по возрастанию. Получается, что на каждой итерации в корне кучи храниться i-й минимум, где i - номер итерации. Теперь докажем время работы алгоритма. Каждый раз при удалении вершины, добавляется 2 новые, следовательно куча увеличивает размер на один, тогда ее максимальный размер не больше k. Получается, что каждая операция работает не больше $\mathcal{O}(\log(k))$. Суммарное время работы алгоритма: $\mathcal{O}(k\log(k))$.

Задача 2

Давайте реализуем функцию $changeKeys\ h\ x$ следующийм образом: будем в каждой вершине кучи хранить дополнительное значение $delt$(значение на которое нужно изменить вершины всего поддерева, включая данную вершину), при вызове $changeKeys\ h\ x$ будем прибавлять x к $delt$ в корне кучи h. Теперь при вызове любой функции будем лениво обновлять значения в вершинах, если мы хотим обратиться к какай-то вершине, то предварительно прибавим значение $delt$ текущей вершины к $delt$ детей, потом прибавим $delt$ текущей вершины к самому значению в вершине и занулим $delt$.

Задача 3

Заведем две двоичные кучи: одна на максимум $maxH$, другая на минимум $minH$, в $minH$ будем хранить $\lceil \frac{n}{2} \rceil$ максимальных элементов, где n общее количество элементов, а в $minH$ в все оставшиеся, теперь надо поддерживать такой инвариант. Реализуем функцию $insert$: добавим элемент в $minH$ если новый элемент больше корня $maxH$, иначе добавим в $maxH$,

если количество элементов в $minH$ стало больше $\lceil \frac{n}{2} \rceil$, то положим корень во вторую кучу и удалим его из $minH$, или если количество элементов в $maxH$ стало больше $n - \lceil \frac{n}{2} \rceil$, то положим корень во вторую кучу и удалим его из $maxH$. Реализуем функцию $medianElement$: просто выведем значение корня $minH$. Реализуем функцию $deleteMedian$: удалим корень из $minH$, теперь количество элементов в $minH$ могло стать меньше $\lceil \frac{n}{2} \rceil$, если такое произошло, то просто возьмем корень из $maxH$, поместим его в $minH$, удалим из $minH$. Все эти функции сохраняют иварикант того, что в одной куче элементы больше чем во второй. Поэтому структура данных коректно поддерживает все операции. Ассимптотика работы каждой функции: $\mathcal{O}(\log(n))$(из размеров куч).
Научимся строить такую структуру за линейное время. Давайте найдем медиану в исходном массиве за $\mathcal{O}(n)$ алгоритмом поиска k-ой порядковой статистики($partition$). Теперь все элементы левее медианы меньше, а правее больше. Просто воспользуеся алгоритмом построения кучи на минимум за линейное время для элементов меньших медианы, получим $maxH$, так же построим кучу на максимум, получим $minH$.

Задача 4

Результатом алгоритма получится матрица, в которой элементы расположены в следующем порядке: элементы будут лежать в порядке сортировки, сначала в первой строке m минимальных, во второй m следующих по минимальности, и так далее. В четных строках элементы будут отсортированы по убыванию, а в не четных — по возрастнию. Заметим, что такое расположение элементов не будет меняться после итерации алгоритма. Количество итераций которое совершит алгоритм равняется $\mathcal{O}(\log(m))$.