Студент: Тимур Хабибуллин
Группа: 1
Дата: 5 мая 2022 г.

---

1. **Гайки и болты**

Пусть нам дано 2 массива: с гайками (`A`) и болтами (`B`). Элемент каждого массива - номер пары, к которой принадлежит гайка или болт (пусть с ростом номера у нас увеличивается диаметр болта и гайки - тогда можно будет сравнивать болты и гайки разных размеров и понимать, у кого номер пары больше, а у кого - меньше). Наша задача - построить отображение, переводящее номер пары в пару индексов, соответсвующих элементам в исходных массивах.
В каждом массиве заменим на пары $\mathtt{X}[i] \to (i, \mathtt{X}[i])$, $\mathtt{X} \in \mathtt{A}, \mathtt{B}$. Потом реализуем быструю сортировку, в которой опорный элемент будем брать из одного массива и сравнивать его с элементами другого по 2 значению в парах:

1. Случайным образом выберем гайку $\mathtt{A}[i]$, используем ее как опорный и сделаем `Partition` в массиве болтов `B`, сравниваясь с выбранной гайкой. Т.е. переставим болты так, чтобы болты с меньщим номером пары лежали левее болта, соответсующего выбранной гайке, с права - больше.
2. Затем сделаем `Partition` в массиве гаек, используя болт из предыдущего пункта.
3. Рекуррентно вызываемся в каждую половинку и аналогично делаем `Partition` в соответсвующих подмассивах болтов и гаек.

В результате мы совместим массивы `A` и `B` и для каждого номера пары теперь можно сопоставить номер гайки и болта из исходного массива.

2. **Детерменированная $k-$порядковая статистика**

(a) Действуем как на паре - разделим исходый массив `A` на блоки по 7 частей (считаем длину массива $n = 7l$). Хотим найти элемент, который гарантированно разделит `A` в какое-то число раз. Пусть это будет медиана среди медиан в блоках по 7. Тогда за $O(n)$ находим медианы в блоках, их $\frac{n}{7}$ штук, тогда в каждом блоке медиана не меньше чем 4 элемента (ключая саму медиану), а потом среди медиан рекуррентно за $T(n/7)$ ищем медиану и она не меньше чем половина от набора медиан. Получаем

$$n_1 = \frac{n}{14} \cdot 4 = \frac{2n}{7}$$

Значит, найдя медиану медиан, мы гарантированно отсечем $\frac{2n}{7}$ элементов, когда сделаем `Patition`, и будем искать поряковую статистику в оставшемся куске из $\frac{5n}{7}$ элементов. Тогда рекуррентное соотношение:

$$T(n) \leqslant T(5n/7) + T(n/7) + O(n)$$

Как и на паре легко проверить, что оно дает $T = O(n)$. Если известно, что $T(m) \leqslant Cm$ для $m < n$, то

$$T(n) \leqslant C\frac{5n}{7} + C\frac{n}{7} + C_1 n = n\left(\frac{6}{7}C + C_1\right) \leqslant Cn$$

для $C_1 \leqslant C/7$.

(b) Теперь будем делить на блоки по 3 (не умаляя общности, считаем что $n = 3l$). Тогда при поиске медианы медиан в каждом блоке медиана не меньше чем 2 элемента, всего медиан $\frac{n}{3}$ штук, а медиана медиан - меньше чем половина последних. Тогда мы гарантированно будем больше и меньше чем
$$n_1 = \frac{n}{3}\frac{2}{2} = \frac{n}{3}$$
элементов. Получаем рекуррентное соотношение:
$$T(n) = T(2n/3) + T(n/3) + O(n)$$
Видно, что мы уже не получим $T(n) = O(n)$, потому что на каждом шаге мы будем совершать $O(n)$ операций (в отличие от предыдущего случая, когда на каждом шаге число операций уменьшается и получается георметрическая прогрессия). Получается, что асимптотика должна иметь вид
$$T(n) = N(n)O(n),$$
где $N(n)$ - сколько шагов сделает алгоритм. По сути, это число уровней в дереве рекурсии. Его просто найти - это $N(n) = \log_{3/2}(n)$. Получаем асимптотику
$$T(n) = \Theta(n\log(n))$$

3. $k-$**ближайших соседей в метрике**

(a) $d(x, \texttt{median}) = |\texttt{pos}(x) - \texttt{pos}(\texttt{median})|$
Из прошлого семинара узнали (5 задача), что можно за $O(n)$ искать $k-$порядковую статистику с помощью модифицированного `MergeSort`.
1. Сначала в исходном массиве `A`) найдем за $O(n)$ медиану, при этом мы фактически сделаем `Partition` и левая часть массива будет меньше `median`, а правая - больше.

2. Затем в левой половине за $O(n)$ найдем $(m - k//2)-$порядковую статистику (обозначим a), а в правом - $(m + k//2 + (k \mod 2))-$порядковую статистику (обозначим b). При этом при их поиске мы в каждой половине делаем `Partition`, поэтому все x, левее a, обязательно $x < a$, а те x, которые правее b: $b < x$. Тем самым элементы $[a...m)$ и $(m...b]$ (они неотсортированны) являются ближайшими к `median` в осортированном массиве.

В результате вокруг медианы мы получим два отрезка суммарной длины k, с одной стороны ближайших и меньших `median`, с другой - ближайших и больших.

(b) $d(x, \texttt{median}) = |x - \texttt{median}|$
Как в предущем пункте находим медиану. Нам надо найти в массиве k элементов, чтобы они были как можно ближе к `median`, т.е.
$$x \in \texttt{A}: \; d(x, \texttt{median}) = |x - \texttt{median}| \to \min$$
Тогда пройдемся по массиву и вместо каждого значения x запишем пару $(x, d(x, \texttt{median}))$. Значит, для того чтобы найти нужные элементы, надо найти $(k+1)-$порядковую статистику в массиве по 2му аргументу (у нас в массиве будет ноль, соответсвующий `median` - он нам не нужен, поэтому возьмем на 1 элемент больше). Тем самым мы найдем k элементов, которые минимально отличаются от медианы

4. (*)

5. **Найти k минимумов**
Прочитаем первые $2k$ массивов и найдем среди них за $O(2k)$ медиану. Слева от нее будут элементы, меньше чем медиана, справа- больше. Выкинем из этого массива последние k элементов и прочитаем следующие k, найдем медиану и снова выкинем последние k элементов. В результате мы совершим n/k шагов, на каждом из которых мы за $O(k)$ находим медиану и отбрасываем элементы, большие ее. Тем самым в конце алгоритма мы получим k минимумов за $O(n)$ времени и $O(k)$ памяти.

6. **Число сравнений**

   (a) Для того, чтобы определить максимум необходимо, чтобы каждое число поучаствовало в сравнении хотя бы один раз, иначе мы не будем иметь информации о об этом числе. Поскольку у в сравнении участвует 2 числа, то число сравнений должно быть не меньше чем $n/2$, иначе мы забыли сравнить какое-то число.

   (b) Пусть нам дан массив $\{x_1, x_2, ..., x_n\}$ размера n. Чтобы найти максимум, возьмем первые два элемента, сравним их и выберем максимум (обозначим его m_1) Затем возьмем следующий элемент в массиве, x_3 и сравним его с m_1. Обозначим максимум из последней пары m_2 и сравним со следующим числом и тд. Продолжая в том же духе, мы переберем все элементы и сделаем $n-1$ сравнение. Полученное число будет максимумом, поскольку на каждом $k-$м шаге алгоритма мы сравниваем x_k и m_{k-2}, который является максимумом среди предыдущих $k-1$ чисел. Тем самым найдем m_{k-1}, т.е. максимум среди первых k элементов. Перебрав все числа мы найдем максимум во всем массиве.
   Нельзя найти максимум за меньшее число сравнений, потому что у нас тогда нарушится цепочка сравнений. Т.е. если мы одно сравнение уберем, то у нас станет 2 группы чисел, в которых мы знаем, как числа соотносятся друг с другом, но между этими группами у нас нет связи и поэтому мы не можем сравнить элементы разных групп. По сути, если числа представить как вершины графа, а ребра - как сравнение пары чисел, то для того, чтобы найти максимум, нам надо получить связный граф, потому что нам надо иметь информацию о том, как соотносятся любые 2 числа, т.е. уметь проходить от одной вершины к другой. А условие на минимальное число ребер решается тем, что граф - дерево, а у него n-1 ребро.

7. **Число сравнений (*)**

   (a) Поскольку у нас $2n$ чисел, разобьем их на n пар и проведем сравнения, после чего получим группы из n максимумов и n минимумов. Так как нам надо найти максимум и минимум среди всех $2n$ чисел, то максимум должен быть в группе с n максимумами, а минимум - с минимумами. Поэтому в первой группе мы ищем максимум, а во второй - минимум. Для этого в каждой группе, по предыдущей задаче, нам понадобится не более чем $n-1$ сравнение. Тем самым мы мы находим нужные элементы за $3n-2$ сравнения.

   (b) Можно еще придумать другой способ. Возьмем первые 2 числа, сравним их за 1 операцию и получим текущий максимум и минимум. Осталось $2n-2$ элемента. Затем на каждом следующем шаге будем брать пару чисел, ее сравнивать и максимум из пары стравнивать с текущим максимумом, а минимум - с текущим минимумом, и в случае чего обновлять. Тогда мы будем делать по 3 сравнения. Здесь не сделать 2, потому что из новой пары не понятно-кто из них больше, а кто - меньше, чтобы потом обновить текущие значения минимума и максимума. Если брать за раз больше элементов, например, k, то надо будет на каждом шаге делать $N(k)$ сравнений и еще 2 для обновление текущих значений, в итоге $2 + N(k)$. Тогда формула для общего числа сравнений:

$$N(n) = 1 + \frac{2n-2}{k} \cdot (2 + N(k))$$

Я не успеваю это оценить, почему надо брать $k = 2$, потому что осталось 5 минут до дедлайна, но по идее должно получиться, что для того, чтобы $N(n)$ было минимальным, надо сделать $k = 2$, т.е. брать по 2 значения. Таким образом, получается $N_{min} = 3n - 2$.