

Лекция 5 от 26.01.2016

Быстрая сортировка. Продолжение

Говоря об алгоритме быстрой сортировки (QSort), мы рассматривали только случаи, когда все элементы различны. Однако это далеко не всегда так. Если в входном массиве есть равные элементы, то алгоритм может застопориться. Для того, чтобы избежать этого, изменим алгоритм PARTITION. Попытаемся преобразовывать массив таким образом, чтобы в левой части стояли элементы строго меньше опорного, в правой — строго большие, а в середине — равные ему:

$$\boxed{\quad} \dots \boxed{x} \dots \boxed{\quad} \longrightarrow \boxed{< x} \boxed{= x} \boxed{> x}$$

Обозначим за опорный элемент последний. Будем проходиться по массиву от начала до конца, выставляя элементы в нужном порядке (? — ещё не просмотренные элементы):

$$\begin{array}{ccccc} \boxed{< x} & \boxed{> x} & \boxed{?} & \boxed{= x} & \boxed{x} \\ [1, i) & [i, j) & [j, k) & [k, n) & n \end{array}$$

Algorithm 1 Модифицированный алгоритм PARTITION

```
1: function PARTITION( $a$ )
2:    $i := 1$ 
3:    $j := 1$ 
4:    $k := n - 1$ 
5:   while  $j < k$  do
6:     if  $a[j] = a[n]$  then
7:        $k := k - 1$ 
8:        $a[j], a[k] := a[k], a[j]$ 
9:     else
10:      if  $a[j] < a[n]$  then
11:         $a[i], a[j] := a[j], a[i]$ 
12:         $j := j + 1$ 
13:         $i := i + 1$ 
14:      else
15:         $j := j + 1$ 
```

Заметим, что $j = k$ (так как алгоритм не закончит работу до тех пор, пока это не станет верно). Тогда на выходе получится массив вида:

$$\begin{array}{ccc} \boxed{< x} & \boxed{> x} & \boxed{= x} \\ [1, i) & [i, j) & [k, n] \end{array}$$

Остаётся только переставить части массива:

```
1:  $j := n$ 
2: while  $i < k$  and  $j \geq k$  do
3:    $a[i], a[j] := a[j], a[i]$ 
4:    $i := i + 1$ 
5:    $j := j - 1$ 
```

В: Самая быстрая из наших сортировок — $O(n \log n)$. А можно ли быстрее?

О: На основе только сравнений — нет.

Использовать разобранные нами сортировки можно на любых сущностях, для которых определена операция сравнения.

Предположим теперь, что мы сортируем натуральные числа, не превосходящие некоторого числа C .

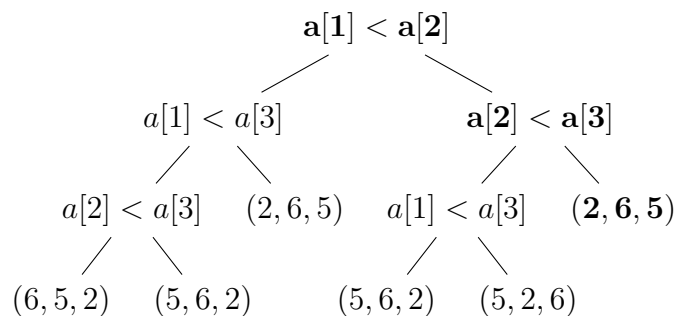
Создадим массив b размера C , заполненный нулями. Будем проходить по исходному массиву a и на каждом шаге будем добавлять 1 к соответствующему элементу массива b :

$$b[a[i]] := b[a[i]] + 1$$

Потом, проходя по получившемуся массиву b , будем восстанавливать исходный массив уже в отсортированном виде.

Такая сортировка будет работать за $O(n)$, однако, она не универсальна.

Вернёмся к универсальным сортировкам. Рассмотрим дерево для массива $a = [6, 5, 2]$:



Подобное дерево можно составить для любого детерминированного¹ алгоритма сортировки, зафиксировав n . Сложность алгоритма будет являться высота h дерева. Посчитаем это h :

- Так как алгоритм должен работать на любой перестановке из n элементов, то у дерева не может быть меньше, чем $n!$ листов.
- Так как сравнение — бинарная операция, то у каждой вершины не более двух потомков. Тогда в дереве не может быть больше, чем 2^h листьев.
- Тогда $2^h \geq n! \iff h \geq \log_2 n!$. Заметим, что:

$$n! = 1 \cdot 2 \cdot \dots \cdot \left\lfloor \frac{n}{2} \right\rfloor \cdot \underbrace{\left(\left\lfloor \frac{n}{2} \right\rfloor + 1 \right) \cdot \dots \cdot (n-1) \cdot n}_{\text{каждый из } \frac{n}{2} \text{ элементов не меньше } \frac{n}{2}} \geq \left(\frac{n}{2} \right)^{\frac{n}{2}}$$

Тогда $h \geq \log_2 \left(\frac{n}{2} \right)^{\frac{n}{2}} = \frac{n}{2} \log_2 \frac{n}{2} = \Omega(n \log n)$. Из этого следует, что отсортировать произвольный массив с помощью только сравнений меньше, чем за $\Omega(n \log n)$ операций, невозможно.

Поиск медианы

Медиана — такой элемент массива, что не меньше половины элементов меньше неё, и не меньше половины — больше.

Для отсортированного массива размера n медиана будет находиться под номером $\frac{n+1}{2}$ для нечётных n и $\frac{n}{2}$ для чётных n . Пример: для массива $(8, 1, 3, 5, 6, 9)$ медианой будет являться 5.

Как же найти медиану? Очевидно, что можно отсортировать и взять средний — $\Theta(n)$.

¹Детерминированный алгоритм — алгоритмический процесс, который выдаёт предопределённый результат для заданных входных данных. Например, QSort, выбирающий опорный элемент случайным образом, не является детерминированным.

А можно ли найти медиану ли за линейное время? Можно. Напишем алгоритм, находящий элемент, стоящий на k -ом месте в массиве, получающемся из входного после сортировки. Это называется поиском k -ой порядковой статистики. Составим этот алгоритм, немного модифицировав QSort:

Algorithm 2 Поиск k -ой порядковой статистики

```

1: function SELECT( $a, k$ )
2:   choose pivot  $a[p]$ 
3:    $i := \text{PARTITION}(a, p)$ 
4:   if  $i := k$  then
5:     return  $a[i]$ 
6:   if  $i > k$  then
7:     return SELECT( $a[1 \dots i - 1], k$ )
8:   else
9:     return SELECT( $a[i + 1 \dots n], k - i$ )

```

Как и в быстрой сортировке, неправильно выбранный опорный элемент портит скорость до n^2 . Будем выбирать опорный элемент случайным образом. Попробуем посчитать время работы в среднем случае.

j -подзадача размера n' . $\left(\frac{3}{4}\right)^{j+1} n < n' \leq \left(\frac{3}{4}\right)^j n$

Как и в QSort, в среднем мы потратим две попытки на переход к следующему j .

Максимальное $j = O(\log_{\frac{4}{3}} n)$

$$T(n) \leq \sum_{j=0}^{\log_{\frac{4}{3}} n} 2 \cdot c \cdot \left(\frac{3}{4}\right)^j n = 2cn \sum_{j=0}^{\log_{\frac{4}{3}} n} \left(\frac{3}{4}\right)^j \leq 2cn$$

Время работы алгоритма в худшем случае всё ещё $O(n^2)$. Худший случай — когда на каждом шаге мы отщеплем всего один элемент. Для достижения лучшего случая, на каждом шаге нужно выбирать в качестве опорного элемента медиану.

Медиана медиан

Попробуем несколько модифицировать наш алгоритм. Разобьём входной массив на группы по 5 элементов. Отсортируем каждую такую группу. Так как размер каждой группы зафиксирован, время сортировки не зависит от n . Зависит только количество сортировок. Возьмём медиану в каждой группе и применим алгоритм нахождения медианы к получившемуся массиву медиан. Выберем её в качестве опорного элемента.

Algorithm 3 Поиск k -ой порядковой статистики 2

```

1: function SELECT( $a, k$ )
2:   Divide  $a$  into groups of 5
3:   Choose medians  $m_1, \dots, m_{\frac{n}{5}}$ 
4:    $x = \text{SELECT}([m_1, \dots, m_{\frac{n}{5}}], \frac{n}{10})$ 
5:   choose  $x$  as pivot  $a[p]$ 
6:    $i := \text{PARTITION}(a, p)$ 
7:   if  $i := k$  then
8:     return  $a[i]$ 
9:   if  $i > k$  then
10:    return SELECT( $a[1 \dots i - 1], k$ )
11:  else
12:    return SELECT( $a[i + 1 \dots n], k - i$ )

```

$$T(n) \leq cn + T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right)$$

$$T(n) \leq ln \text{ для некоторого } l$$

$$T(n) \leq cn + T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}\right) \leq cn + \frac{ln}{5} + \frac{7}{10}ln$$