О поиске медианы массива за линейное время

А. С. Малистов

В сборнике «Математическое просвещение», сер. 3, вып. 11 (М.: МЦНМО, 2007) на с. 164 опубликована следующая задача 11:

Дано $2n+1$ грузов попарно различной массы и чашечные весы без гирь. Докажите, что за $100n$ взвешиваний можно найти *медиану* (т. е. средний по массе груз). (*Фольклор*)

Рассмотрим следующую более общую задачу.

Описать алгоритм поиска медианы массива за линейное время.

Прежде чем мы подробно опишем требуемый алгоритм, определимся с терминологией. Напомним, что такое медиана массива, что значит время работы алгоритма и в каком случае это время можно считать линейным.

Пусть дано конечное линейно упорядоченное множество M, содержащее n объектов $\{x_1, x_2, \ldots, x_n\}$. Это означает, что для любых двух его элементов задано отношение порядка $\leqslant$, удовлетворяющее условиям рефлексивности ($\forall\, a\ \ a \leqslant a$), транзитивности ($\forall\, a, b, c\ \ a \leqslant b,\ b \leqslant c\ \Rightarrow\ a \leqslant c$) и антисимметричности ($\forall\, a, b\ \ a \leqslant b,\ b \leqslant a\ \Rightarrow\ a = b$). Если упорядочить все элементы множества M по возрастанию, то i-й *порядковой статистикой* называется i-й по возрастанию элемент. Если n — нечётное число, то *медианой* множества M называется $((n+1)/2)$-я статистика, которая находится ровно посередине. Если n — чётное число, то медиана не может быть однозначно определена и различают нижнюю и верхнюю медианы с индексами $n/2$ и $n/2+1$ соответственно. Иногда в качестве медианы в случае чётного n берут среднее арифметическое между нижней и верхней медианами. Нам требуется описать алгоритм поиска медианы за линейное время, но мы будем решать более общую задачу поиска i-й порядковой статистики за линейное время, которая покрывает задачу поиска медианы.

Теперь попробуем понять, что значит время работы алгоритма и в каком случае это время называют линейным. Чтобы исключить неоднозначность в определении времени, требуется абстрактная модель компьютера

с универсальными инструкциями. Такие абстрактные компьютеры называются *моделями вычислений* или, по-другому, *вычислительными моделями*. Хронологически самой первой такой моделью считается *машина Тьюринга*, которая, правда, недостаточно хорошо моделирует современные компьютеры. Мы будем использовать модель машины с произвольным доступом к памяти, которая гораздо лучше подходит для современных вычислительных устройств. Это некоторый абстрактный компьютер с памятью, состоящей из ячеек, адреса которых нумеруются последовательно, начиная с нуля. Каждая ячейка может хранить целое число, ограниченное по модулю некоторым положительным числом L. Значение L зависит от входных данных решаемой на компьютере задачи. Кроме того, компьютер имеет конечное число регистров для проведения арифметических операций. Разрешены операции чтения чисел из памяти в регистр и записи из регистра в память, арифметические операции сложения, умножения, вычитания, целочисленного деления, взятия остатка. За одну операцию также можно сравнить значения любых двух чисел и, в зависимости от результата, перейти к любому шагу алгоритма. Кроме того, разрешён непрямой доступ к памяти, когда адрес обрабатываемой ячейки может быть задан в одном из регистров. *Временем работы* алгоритма называется число операций, которые алгоритм выполняет для решения задачи.

Пусть элементы множества M записаны в последовательных ячейках памяти нашего компьютера. Адреса этих ячеек находятся в диапазоне $[s, s+n-1]$, где s — начальный адрес, с которого начинается вход алгоритма. Требуется предложить алгоритм, который найдёт i-ю порядковую статистику входного массива. Отметим, что элементы множества M необязательно должны быть числами и могут иметь произвольное происхождение. Важно лишь, чтобы их можно было записать в ячейки памяти компьютера, и должна существовать возможность сравнивать эти объекты, чтобы определить, какой элемент меньше, а какой больше. Под этой возможностью понимается, что существует вспомогательная процедура, принимающая на вход два адреса ячеек памяти, которые содержат элементы a и b, сравнивает эти элементы и возвращает 1, если $a \leqslant b$, и 0 в противном случае. Время работы этой процедуры принимается условно за единицу.

Говорят, что алгоритм выполняется за *линейное время*, если количество шагов алгоритма ограничено многочленом первой степени от n для всех $n \in \mathbb{N}$. Иначе говоря, существует такой многочлен $an + b$, что для любого входа длины n время работы алгоритма не превысит $an + b$. Обратим внимание на распространённую ошибку. Увеличение входа, скажем, вдвое вовсе не означает, что алгоритм, работающий за линейное время, станет тратить времени ровно в два раза больше. Более того, это время вообще может

оказаться не определённым однозначно, так как существует огромное количество различных входов длины n и каждый из них может требовать разное число операций. Линейное время означает лишь, что число таких операций ограничено сверху линейным многочленом. Иногда используют альтернативное определение, которое требует, чтобы существовали такие константы c и n_0, что время работы алгоритма меньше cn для всех $n > n_0$. Нетрудно убедиться, что эти определения эквивалентны. Второе следует из первого, если задать $c = |a| + |b|$, $n_0 = 1$, а первое следует из второго, если считать, что $a = c$, а b — это максимальное время работы для всех входов с длиной не больше n_0.

Существует универсальная запись, которая обобщает оценку времени работы алгоритма на случай не только линейных функций. Пусть с входными данными связан некоторый целочисленный параметр n, например, количество чисел в массиве. Этот параметр n часто называют функцией длины входа. Будем говорить, что алгоритм выполняется за $O(f(n))$, если существуют такие константы C и n_0, что для всех $n > n_0$ число операций, которые выполняет алгоритм на любом входе длины n, не превышает $Cf(n)$. Тогда фраза «алгоритм работает за линейное время» эквивалентна фразе «алгоритм работает за время $O(n)$».

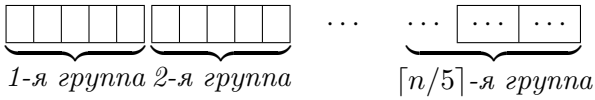
Найти медиану массива, а также любую другую порядковую статистику можно нехитрым способом, предварительно отсортировав массив по возрастанию. Так как мы используем машину с произвольным доступом к памяти, после сортировки можно обратиться к ячейке с адресом $s + i - 1$, чтобы получить i-ю статистику (s — адрес первой ячейки отсортированного массива). Известные алгоритмы быстрой сортировки, например сортировка слиянием, работают за время $O(n \log_2 n)$. При этом для любого n, начиная с некоторого n_1, всегда существует вход длины n, на котором эта верхняя оценка достижима, то есть число операций больше $cn \log_2 n$ для некоторой ненулевой константы c. Если не использовать внутреннюю структуру элементов массивов, а опираться лишь на попарные сравнения элементов, то не существует алгоритма сортировки, который работал бы за линейное время. Тем не менее, если сортировать массив не требуется, а нужно лишь найти i-ю порядковую статистику, то сделать это можно за линейное время. Алгоритм, который мы опишем, был придуман Блюмом, Флойдом, Праттом, Ривестом и Тарьяном и опубликован в 1973 году [1]. Иногда его называют BFPRT-алгоритмом по первым буквам фамилий разработчиков.

Сперва опишем вспомогательную процедуру, которая называется *разбиением массива*. Дан массив, состоящий из n элементов, и дополнительный элемент x. Необходимо упорядочить элементы массива таким образом, чтобы сначала шли элементы, не превосходящие x (т. е. все такие a, для которых $a \leqslant x$), а затем все остальные элементы. Оказывается, сделать такое

разбиение можно за линейное время, причём не используя существенно дополнительную память. Нам потребуется всего лишь два дополнительных счётчика. В начале алгоритма первый счётчик должен указывать на первый элемент массива, а второй счётчик — на последний. Первый счётчик будет увеличивать своё значение на единицу до тех пор, пока для элемента a, на который он указывает, выполняется $a \leqslant x$. Второй счётчик должен уменьшать своё значение на единицу до тех пор, пока для элемента b, на который он указывает, неверно, что $b \leqslant x$. Когда первый счётчик станет указывать на элемент a, больший, чем x, а второй на такой элемент b, что $b \leqslant x$, необходимо поменять элементы a и b местами, а затем продолжить процедуру. Алгоритм заканчивает работу, когда второй счётчик перестаёт быть больше первого. Число изменений каждого счётчика не превышает n, а число сравнений элементов и их возможных перестановок не превосходит [1] $\lfloor n/2 \rfloor$.

Теперь мы можем описать *основной алгоритм*, который должен найти i-ю порядковую статистику.

A1. Входной массив разбивается на одинаковые небольшие группы, например на $\lceil n/5 \rceil$ групп по 5 элементов в каждой, кроме, возможно, одной группы, в которой может быть меньше элементов. Элементы в каждой группе стоят по порядку: в первой группе находятся элементы с 1-го по 5-й, во второй — с 6-го по 10-й, и т. д.

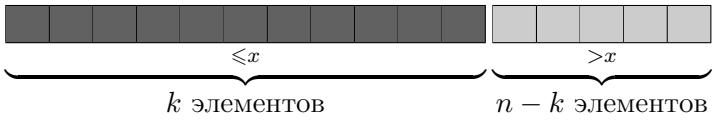


A2. Сначала каждая из маленьких групп сортируется по возрастанию любым известным методом сортировки, а затем в каждой отсортированной группе выбирается медиана. Если $n \leqslant 5$, то работа алгоритма на этом заканчивается.

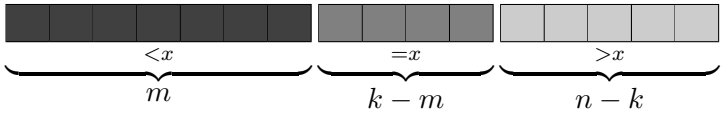
A3. С помощью того же алгоритма для $\lceil n/5 \rceil$ определяется *верхняя* медиана x из $\lceil n/5 \rceil$ медиан, найденных на втором шаге (другими словами, $\lceil \lceil n/5 \rceil /2 \rceil$-я порядковая статистика в массиве медиан).

A4. С помощью процедуры *разбиения массива*, описанной выше, входной массив делится относительно медианы медиан x. Пусть в нижнюю часть разбиения попало k элементов, а в верхнюю часть попало $n - k$ элементов.

[1] Напомним, что $\lfloor t \rfloor$ обозначает наибольшее целое число, не превосходящее t, а $\lceil t \rceil$ — наименьшее целое число, не меньшее чем t.



A5. С помощью процедуры, аналогичной *разбиению массива*, нижняя часть входного массива делится на две части: в первую входят все числа, которые не равны x (то есть меньше x), а во вторую — те, которые равны x. Пусть элементов, меньших x, оказалось ровно m.



A6. Если $m < i \leqslant k$, то возвращается значение x. Если $i < m$, то процедура вызывается повторно для нижней части разбиения. Если $i > k$, то процедура вызывается повторно для верхней части разбиения, но ищется $(i-k)$-я порядковая статистика.

Корректность алгоритма достаточно очевидно вытекает из описания последнего шага. На шестом шаге мы повторно запускаем алгоритм на уменьшенной выборке, корректно меняя искомый индекс статистики, если потребуется. Алгоритм всегда завершается за конечное число шагов, так как разбиение на четвёртом и пятом шагах всегда уменьшает размер обрабатываемого массива.

Теперь разберёмся, почему алгоритм работает за линейное время. Главную роль в этом играет медиана медиан x, относительно которой производится разбиение. Дело в том, что половина медиан, найденных на втором шаге, меньше или равна медиане медиан x. Более точно, таких медиан будет не меньше чем $\bigl\lceil \lceil n/5 \rceil / 2 \bigr\rceil$. Исключим из рассмотрения маленькую группу, в которой может быть менее пяти элементов, и группу, которая включает медиану медиан x. Внутри каждой оставшейся полной группы, в которой медиана меньше или равна x, находятся 5 элементов, три из которых вместе с медианой меньше или равны x. Таким образом, количество Q элементов, не превышающих x, удовлетворяет следующему неравенству:

$$Q \geqslant 3\Bigl(\Bigl\lceil \frac{\lceil n/5 \rceil}{2} \Bigr\rceil - 2\Bigr) \geqslant 3\Bigl(\frac{n}{10} - 2\Bigr) = \frac{3n}{10} - 6.$$

Аналогично количество элементов, которые больше или равны x, не меньше $3n/10 - 6$. Отсюда следует, что к шестому шагу остаётся на дальнейшую обработку не более чем $\min\{7n/10 + 6, n - 1\}$ элементов (даже если $7n/10 + 6$ больше n, как минимум один элемент выкидывается — это медиана медиан x).

Пусть $T(n)$ — максимальное время работы алгоритма по всем входам длины n. Индукцией по n получаем, что для выполнения шагов 1, 2, 4 и 5 требуется линейное время работы. Пусть это время ограничено многочленом an. Тогда $T(n)$ удовлетворяет следующему неравенству:

$$T(n) \leqslant an + T\Big(\Big\lceil \frac{n}{5} \Big\rceil\Big) + T\Big(\min\Big\{n-1,\, \Big\lfloor \frac{7n}{10} + 6 \Big\rfloor\Big\}\Big). \tag{1}$$

Пусть $T_0 = \max_{n \leqslant 700} T(n)$ — максимальное время по всем входам длины не больше 700. Нам осталось доказать, что существует такая константа c, что $T(n) < cn$ для всех $n \in \mathbb{N}$. Если эту константу взять больше, чем T_0, то это сразу докажет наше утверждение для всех $n \leqslant 700$. Для $n > 700$ требуется шаг индукции. Если $n > 700$, то

$$\min\Big\{n-1,\, \Big\lfloor \frac{7n}{10} + 6 \Big\rfloor\Big\} = \frac{7n}{10} + 6,$$

поэтому согласно (1) и предположению индукции получаем

$$\begin{aligned} T(n) &\leqslant an + c\Big\lceil \frac{n}{5} \Big\rceil + c\Big\lfloor \frac{7n}{10} + 6 \Big\rfloor \leqslant \\ &\leqslant an + c\Big(\frac{n}{5} + 1\Big) + \frac{7cn}{10} + 6c = cn + \Big(an - \frac{cn}{10} + 7c\Big). \end{aligned}$$

Неравенство будет верным, если выбрать константу c такой, что

$$an - \frac{cn}{10} + 7c < 0$$

для всех $n > 700$. Решая последнее неравенство относительно c и учитывая, что $n/10 > 7$ при $n > 700$, получаем, что достаточно выбрать

$$c > \frac{an}{n/10 - 7} = \frac{10a}{1 - 70/n}.$$

Подойдёт $c = 12a$, так как при $n > 700$ всегда $12a > \dfrac{10a}{1 - 70/n}$.

Список литературы

[1] *Blum M., Floyd R. W., Pratt V., Rivest R. L., Tarjan R. E.* Time bounds for selection // Journal of computer and system sciences. V. 7, № 4. 1973. P. 448–461.

Алексей Сергеевич Малистов, ООО «Яндекс»
alexey@malistov.ru