Работа с массивами: поиск, сортировка

А. Г. Фенстер, http://info.fenster.name

6 февраля 2009 г.

Разберём несколько алгоритмов, работающих с массивами, и оценим время их работы.

Бинарный поиск

Бинарный поиск применяется для поиска элемента в упорядоченном по возрастанию или убыванию массиве. Пусть для определённости массив упорядочен по возрастанию. Если нужно определить, в какой позиции находится в массиве A размера N число k, мы можем сравнить его со значением среднего элемента массива и продолжить процесс с левой или правой половиной массива в зависимости от того, больше или меньше числа k окажется средний элемент. Формально алгоритм будет записан так:

1. Присвоить $l = 0$, $r = N - 1$.
2. Если $l > r$, закончить выполнение: число k в массиве A отсутствует.
3. Вычислить $m = \frac{l+r}{2}$.
4. Если $k = A_m$, закончить выполнение: число k найдено в позиции m.
5. Если $k < A_m$, присвоить $r = m - 1$, в противном случае присвоить $l = m + 1$.
6. Перейти к шагу 2.

Если в массиве не найдётся элемент, равный k, мы сделаем наибольшее количество проверок. Выведем оценку времени работы алгоритма в этом случае. На шаге 5 мы уменьшаем количество рассматриваемых элементов в 2 раза (после первого выполнения этого шага у нас останется $\frac{N}{2} - 1$ элементов, и так далее). Если в массиве было $2^p - 1$ элементов, то делить его пополам мы можем p раз, таким образом, для N-элементного массива цикл выполнится $[\log_2 N] + 1$ раз (через $[x]$ обозначается целая часть числа x). Можно сказать, что время работы бинарного поиска – $O(\log_2 N)$.

Сортировка массива

Сортировка методом выбора

Сортировка выбором относится к простым алгоритмам сортировки. Идея её заключается в следующем: сначала мы найдём в массиве минимальный элемент и обменяем его местами с первым элементом массива (после этого он займёт своё место в массиве). Затем в части массива, начиная со второго элемента, снова найдём минимальный элемент и поставим его уже на второе место. Повторяя эти операции в цикле, получим отсортированный массив.

Время работы алгоритма просто подсчитать, оценив количество выполнений внутреннего цикла (цикла поиска минимального элемента). Ясно, что алгоритм выполнит порядка N^2 операций, т. е. время работы его — $O(N^2)$.

Быстрая сортировка

Быстрая сортировка реализуется рекурсивно: функции `quicksort` на вход подаётся массив A, который необходимо отсортировать, и два натуральных числа l и r: левая и правая границы подмассива, с которым происходит работа.

```
void quicksort(int *A, int l, int r);
```

Функция должна упорядочить элементы массива от l-го до r-го включительно в неубывающем порядке.

При вызове функция переставляет элементы массива так, чтобы левее некоторого элемента все числа были меньше либо равны некоторому x, а правее — больше либо равны ему. Если элементы переставлены требуемым образом, функция вызывает себя для левого и правого подмассивов (для каждого из подмассивов рекурсивный вызов происходит только в том случае, если в подмассиве больше одного элемента).

Разделение массива происходит следующим образом. Сначала выбирается опорный элемент — это то самое число x, относительно которого массив будет разделён на две части. В качестве x можно брать, вообще говоря, любое число, лежащее в интервале $[min, max]$, где min и max — минимальный и максимальный элемент подмассива $A[l], \ldots, A[r]$. Вот некоторые часто используемые варианты:

$$x = \frac{A[l] + A[r]}{2}$$

$$x = A[l]$$

Выбрав опорный элемент, заводим два счётчика i и j с начальными значениями l и r соответственно. Двигаясь слева направо (увеличивая i), ищем элемент массива, больший либо равный числу x. Двигаясь справа налево (уменьшая j), ищем элемент массива, меньший либо равный числу x. Когда два таких элемента нашлись (т.к. x выбрано в интервале $[min, max]$, i и j при поиске не выйдут за пределы массива), сравниваем i и j: если $i \leqslant j$, меняем местами элементы $A[i]$ и $A[j]$, сдвигаем счётчики ещё на один элемент и продолжаем поиск. Процесс заканчивается, когда i и j встречаются. Сейчас все элементы, расположенные левее i-го, меньше либо равны x, а все элементы, расположенные правее j-го — больше либо равны x, соответственно, требуемое разделение массива найдено. Если $i < r$, рекурсивно вызываем функцию `quicksort` для подмассива от i-го до r-го элемента; если $l < j$, делаем рекурсивный вызов для подмассива от l-го до j-го элемента.

Пирамидальная сортировка

Пирамидальная сортировка основана на многократном приведении массива к виду пирамиды. *Пирамидой* называется такой массив A размера N, для которого выполняются следующие условия:

$$A[i] \geqslant A[2i+1] \ \forall i \text{ при условии, что } 2i+1 < N,$$

$$A[i] \geqslant A[2i+2]\ \forall i \text{ при условии, что } 2i+2 < N.$$

Рассмотрим *процедуру просеивания*, принимающую на вход массив, его длину и номер текущего элемента:

```
void sift(int *A, int N, int k);
```

Функция предполагает, что часть массива, начиная с $k+1$-го элемента, уже образует пирамиду, и добавляет к этой пирамиде k-й элемент. Для этого она сравнивает его с $2k+1$-м (если $2k+1 < N$) и $2k+2$-м (если $2k+2 < N$) и если условие пирамиды для k-го элемента нарушено, меняет его местами с максимальным из них. При этом может нарушиться условие пирамиды для позиции, с которой произведён обмен, поэтому необходимо либо рекурсивно вызвать функцию просеивания для этой позиции, либо выполнять описанную операцию в цикле (пока k не вышло за пределы массива).

Для произвольного массива длины N элементы, начиная с середины, уже образуют пирамиду (т.к. их просто не с чем сравнивать). Вызывая функцию просеивания последовательно для всех элементов от среднего до нулевого, можно привести массив к виду пирамиды.

Из определения пирамиды следует, что максимальный элемент в таком массиве находится на нулевой позиции. Обменяем его местами с последним элементом (максимальный элемент займёт своё место) и уменьшим число N. Если не рассматривать этот последний элемент, то все элементы, кроме нулевого, уже образуют пирамиду. Чтобы добавить в пирамиду нулевой элемент, достаточно вызвать для него функцию просеивания. При этом на нулевую позицию выйдет элемент, наибольший из оставшихся. Поставим его на второе от конца места и снова уменьшим N. Повторив процесс $N-1$ раз, получим упорядоченный массив.

Временные оценки алгоритмов сортировки

В таблице представлены оценки времени работы основных алгоритмов сортировки. Под «простыми сортировками» понимаются сортировки методом выбора, вставки и пузырьковая сортировка.

Метод сортировки	**Время работы**	
	среднее	**наихудшее**
Простые сортировки	$O(N^2)$	$O(N^2)$
Быстрая сортировка	$O(N \log_2 N)$	$O(N^2)$
Пирамидальная сортировка	$O(N \log_2 N)$	$O(N \log_2 N)$