# Тема 5. Сортировка

## *5.3. Обменная сортировка*

Обменная сортировка некоторым систематическим образом меняет местами пары имен, не отвечающие порядку до тех пор, пока такие пары существуют. Рассмотрим два алгоритма обменных сортировок.

### *5.3.1. Пузырьковая сортировка*

*Пузырьковая сортировка* (алгоритм 5.3) основана на просмотре пар смежных имен последовательно слева направо и перемене мест тех имен, которые не отвечают порядку. Переменная $b$ используется для предотвращения избыточного просмотра имен в правой части таблицы, про которые известно, что они находятся на своих окончательных позициях. В начале цикла **while** значение переменной $b$ равно наибольшему индексу $t$, такому, что про имя $x_t$ еще не известно, стоит ли оно на окончательной позиции. Процесс работы алгоритма и изменения значений элементов вектора инверсий после каждого прохода для таблицы из $n = 8$ имен представлены на рис. 5.4.

$$
\begin{array}{l}
b \leftarrow n \\
\textbf{while}\ b \neq 0\ \textbf{do} \begin{cases} t \leftarrow 0 \\ \textbf{for}\ \ j \leftarrow 1\ \ \textbf{to}\ \ b-1\ \ \textbf{do} \\ \quad \textbf{if}\ \ x_j > x_{j+1}\ \ \textbf{then} \begin{cases} x_j \leftrightarrow x_{j+1} \\ t \leftarrow j \end{cases} \\ b \leftarrow t \end{cases}
\end{array}
$$

Алгоритм 5.3. Пузырьковая сортировка

| Проход | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $d_6$ | $d_7$ | $d_8$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 27 | 07 | 54 | 42 | 14 | 10 | 25 | 17 | 0 | 1 | 0 | 1 | 3 | 4 | 3 | 4 |
| 1 | 07 | 27 | 42 | 14 | 10 | 25 | 17 | 54 | 0 | 0 | 0 | 2 | 3 | 2 | 3 | 0 |
| 2 | 07 | 27 | 14 | 10 | 25 | 17 | 42 | 54 | 0 | 0 | 1 | 2 | 1 | 2 | 0 | 0 |
| 3 | 07 | 14 | 10 | 25 | 17 | 27 | 42 | 54 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 4 | 07 | 10 | 14 | 17 | 25 | 27 | 42 | 54 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 07 | 10 | 14 | 17 | 25 | 27 | 42 | 54 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Рис. 5.4. Процесс работы алгоритма пузырьковой сортировки

Эффективность рассматриваемого алгоритма зависит от трех факторов: числа проходов (числа выполнений тела цикла **while**), числа сравнений «$x_j > x_{j+1}$» и числа обменов «$x_j \leftrightarrow x_{j+1}$».

Очевидно, что число обменов $M(n)$ равно сумме элементов вектора инверсий. Поэтому

$M_{\min}(n) = 0$ в лучшем случае,

$M_{\text{ave}}(n) = \frac{1}{4}n(n-1) = O(n^2)$ в среднем,

$M_{\max}(n) = \frac{1}{2}n(n-1) = O(n^2)$ в худшем случае.

При сравнительном анализе различных алгоритмов необходимо иметь в виду, что для многих языков программирования операция обмена отсутствует. В таких случаях операция обмена эквивалентна трем операциям пересылки (присваивания): $t \leftarrow x_j$, $x_j \leftarrow x_{j+1}$ и $x_{j+1} \leftarrow t$.

Анализ изменений вектора инверсий в процессе пузырьковой сортировки показывает, что каждый проход (исключая последний) уменьшает на единицу каждый ненулевой элемент вектора инверсий и сдвигает вектор на одну позицию влево. Таким образом, число проходов $A(n)$ равно наибольшему элементу вектора инверсий плюс единица, т. е.

$$A(n) = 1 + \max(d_1, d_2, \ldots, d_n).$$

Следовательно, в лучшем случае имеется всего один проход, в худшем случае – $n$ проходов. Для определения среднего числа проходов необходимо найти математическое ожидание $\sum_{k=1}^{n} kP_k$, где $P_k$ – вероятность того, что потребуется ровно $k$ проходов, т. е. вероятность того, что наибольшим элементом вектора инверсий является $k - 1$. Число векторов инверсий, содержащих только такие элементы, значения которых меньше $k$ $(1 \le k \le n)$, равно $k^{n-k}k!$. Тогда число векторов инверсий с наибольшим элементом, равным $k - 1$, будет равно $k^{n-k}k! - (k-1)^{n-k+1}(k-1)!$. Следовательно, вероятность того, что потребуется ровно $k$ проходов, равна

$$P_k = \frac{1}{n!}(k^{n-k}k! - (k-1)^{n-k+1}(k-1)!).$$

Теперь можно вычислить среднее значение

$$\sum_{k=1}^{n} kP_k = n+1-\sum_{k=1}^{n}\frac{k^{n-k}k!}{n!} = n+1-F(n),$$

где $F(n)$ – функция, асимптотическое поведение которой описывается формулой

$$F(n)=\sqrt{\pi n/2}-2/3-O(1/\sqrt{n}).$$

Таким образом, общее число проходов цикла **while**, осуществляемых алгоритмом пузырьковой сортировки, составляет

$A_{\min}(n) = 1$ в лучшем случае,

$A_{\text{ave}}(n) = n-\sqrt{\pi n/2}+5/3+O(1/\sqrt{n}) = O(n)$ в среднем,

$A_{\max}(n) = n$ в худшем случае.

Общее число сравнений $C(n)$ исследовать несколько сложнее. Поэтому приведем только результаты анализа:

$C_{\min}(n) = n - 1 = O(n)$ в лучшем случае,

$C_{\mathrm{ave}}(n) = \dfrac{1}{2}(n^2 - n \ln n) + O(n) = O(n^2)$ в среднем,

$C_{\max}(n) = \dfrac{1}{2} n(n-1) = O(n^2)$ в худшем случае.

Проведенный анализ показывает, что алгоритм пузырьковой сортировки имеет асимптотическую временную сложность $O(n^2)$ в среднем и в худшем случаях. При этом наилучшим для алгоритма является случай, когда исходная таблица уже упорядочена, а наихудшим – когда имена в исходной таблице первоначально расположены в обратном порядке.

Усовершенствованием пузырьковой сортировки является так называемая *шейкер-сортировка*, которая предполагает попеременные проходы в противоположных направлениях. Это позволяет несколько сократить число сравнений, но не число обменов. Как пузырьковая, так и шейкер-сортировка существенно уступают по эффективности простой сортировке вставками.

Другое усовершенствование основано на идее сортировки Шелла (сортировки с убывающим шагом), когда осуществляется обмен именами, расположенными не в соседних позициях, а на больших расстояниях друг от друга. Отличием от сортировки Шелла является то, что в качестве $h$-сортировки используется пузырьковая сортировка. Это позволяет улучшить асимптотические характеристики, но полученный алгоритм все равно будет существенно уступать по эффективности сортировке Шелла.

### 5.3.2. *Быстрая сортировка*

Как и в простой сортировке вставками, в пузырьковой сортировке основным источником неэффективности является то, что обмены дают слишком малый эффект, так как в каждый момент времени имена сдвигаются только на одну позицию. Такие алгоритмы всегда требуют порядка $n^2$ операций, как в среднем, так и в худшем случаях. В быстрой сортировке используется прием, приводящий к тому, что каждый обмен совершает больше работы.

Идея метода *быстрой сортировки* заключается в том, что в таблице выбирается одно из имен, которое используется для разделения таблицы на две подтаблицы, состоящие соответственно из имен меньших и больших выбранного. Разделение можно реализовать, одновременно просматривая таблицу слева направо и справа налево, меняя местами имена в неправильных частях таблицы. Имя, используемое для расщепления таблицы (*расщепляющее*, или *базовое*, имя), затем помещается между двумя подтаблицами. Полученные подтаблицы сортируются рекурсивно.

Рекурсивный алгоритм быстрой сортировки (алгоритм 5.4) сортирует таблицу $x_f, x_{f+1}, \ldots, x_l$, где $x_f$ является базовым именем, используемым для разбиения таблицы на подтаблицы.

**procedure** $QUICKSORT\,(f, l)$

// отсортировать $x_f, x_{f+1}, \ldots, x_l$

**if** $f > l$ **then return**

// разделить таблицу

$i \leftarrow f+1$

**while** $x_i < x_f$ **do** $i \leftarrow i+1$

$j \leftarrow l$

**while** $x_j > x_f$ **do** $j \leftarrow j-1$

$$\textbf{while } i < j \textbf{ do} \begin{cases} \text{// в этот момент } i < j,\ x_i \ge x_f \ge x_j \\ x_i \leftrightarrow x_j \\ i \leftarrow i+1 \\ \textbf{while } x_i < x_f \textbf{ do } i \leftarrow i+1 \\ j \leftarrow j-1 \\ \textbf{while } x_j > x_f \textbf{ do } j \leftarrow j-1 \end{cases}$$

$x_f \leftrightarrow x_j$

// отсортировать подтаблицы рекурсивно

$QUICKSORT\,(f, j-1)$

$QUICKSORT\,(j+1, l)$

**return**

Алгоритм 5.4. Рекурсивный алгоритм быстрой сортировки

Алгоритм предполагает, что имя $x_{l+1}$ определено и больше, чем $x_f$, $x_{f+1}$, …, $x_l$, т. е. перед первым обращением к процедуре $QUICKSORT(1, n)$ необходимо установить $x_{n+1} \leftarrow \infty$ в качестве сторожа. В начале цикла «**while** $i < j$» индексы $i$ и $j$ указывают соответственно на первое и последнее имена, о которых известно, что они находятся не в тех частях таблицы, в которых требуется. Когда $i$ и $j$ встречаются, т. е. когда $i \geq j$, все имена находятся в соответствующих частях таблицы, а имя $x_f$ помещается между двумя частями, меняясь местами с $x_j$. Процесс разбиения алгоритмом 5.4 таблицы на две подтаблицы показан на рис. 5.5.

| | $x_f$ | $x_{f+1}$ | $x_{f+2}$ | | | | ... | | | | | $x_{l-1}$ | $x_l$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $i$ | | | | | | | | | | | $j$ |
| Начало | 22 | 07 | 54 | 42 | 14 | 27 | 25 | 17 | 49 | 63 | 16 | 03 | 44 |
| | | | $i$ | | | | | | | | | $j$ | |
| 1 – й обмен | 22 | 07 | 54 | 42 | 14 | 27 | 25 | 17 | 49 | 63 | 16 | 03 | 44 |
| | | | | $i$ | | | | | | | $j$ | | |
| 2 – й обмен | 22 | 07 | 03 | 42 | 14 | 27 | 25 | 17 | 49 | 63 | 16 | 54 | 44 |
| | | | | | | $i$ | | $j$ | | | | | |
| 3 – й обмен | 22 | 07 | 03 | 16 | 14 | 27 | 25 | 17 | 49 | 63 | 42 | 54 | 44 |
| | | | | | | $j$ | $i$ | | | | | | |
| $i > j, x_f \leftrightarrow x_j$ | 22 | 07 | 03 | 16 | 14 | 17 | 25 | 27 | 49 | 63 | 42 | 54 | 44 |
| Результат | 17 | 07 | 03 | 16 | 14 | 22 | 25 | 27 | 49 | 63 | 42 | 54 | 44 |

Рис. 5.5. Процесс разбиения таблицы на две подтаблицы

Определим общее число сравнений имен «$x_i < x_f$» и «$x_j > x_f$». В конце цикла «**while** $i < j$» все имена $x_{f+1}$, $x_{f+2}$, …, $x_l$ сравнивались с $x_f$ по одному разу, исключая имена $x_s$ и $x_{s+1}$ (где просмотры встретились), которые сравнивались с $x_f$ дважды. Следовательно, для таблицы из $n$ имен ($n = l - f + 1$) к моменту разделения на подтаблицы выполняется $n + 1 = l - f + 2$ сравнений. Поскольку разбиение таблицы производится только в случае, если $f < l$, для тривиальных таблиц (вырожденных, когда $n = 0$ при $f > l$, или состоящих из одного имени, когда $n = 1$ при $f = l$) число сравнений равно нулю.

Пусть $C_{\text{ave}}(n)$ – среднее число сравнений имен для сортировки таблицы из $n$ разных имен в предположении, что все $n!$ перестановок таблицы равновероятны. Для тривиальных таблиц $C_{\text{ave}}(0) = C_{\text{ave}}(1) = 0$. В общем случае имеем

$$C_{\text{ave}}(n) = n + 1 + \sum_{s=1}^{n} p_s (C_{\text{ave}}(s-1) + C_{\text{ave}}(n-s)), n \geq 2 \text{.}$$

Здесь $p_s$ – вероятность того, что $x_1$ (расщепляющее имя) есть $s$-е наименьшее имя. Поскольку две подтаблицы, порожденные разбиением, случайны, т. е. все $(s-1)!$ перестановок имен в левой подтаблице равновероятны и все $(n-s)!$ перестановок имен в правой подтаблице равновероятны, $p_s = 1/n$. Таким образом,

$$C_{\text{ave}}(n) = n + 1 + \frac{1}{n} \sum_{s=1}^{n} (C_{\text{ave}}(s-1) + C_{\text{ave}}(n-s)), n \geq 2 \text{.}$$

Поскольку эта сумма равна

$$\begin{aligned} &C_{\text{ave}}(0) + C_{\text{ave}}(n-1) + C_{\text{ave}}(1) + C_{\text{ave}}(n-2) + \cdots + \\ &+ C_{\text{ave}}(n-2) + C_{\text{ave}}(1) + C_{\text{ave}}(n-1) + C_{\text{ave}}(0), \end{aligned}$$

получаем

$$C_{\text{ave}}(n) = n + 1 + \frac{2}{n} \sum_{s=0}^{n-1} C_{\text{ave}}(s), n \geq 2.$$

Использование известных способов решения подобных рекуррентных соотношений (см. п. 1.4) дает

$$C_{\mathrm{ave}}(n) = (\ln 4)\, n \log n + O(n) \approx 1.386\, n \log n,$$

т. е. среднее число сравнений имен равно $O(n \log n)$.

Очевидно, что худшим для алгоритма быстрой сортировки является случай, когда каждый раз для разбиения таблицы берется наибольшее или наименьшее имя. Тогда на каждом этапе одна из подтаблиц будет вырожденной ($n = 0$), а другая будет состоять из $n - 1$ имен. Таким образом, если алгоритм применяется к уже отсортированным или отсортированным в обратном порядке таблицам, то производится

$$C_{\max}(n) = (n+1) + n + (n-1) + \cdots + 3 = \frac{1}{2}n^2 + \frac{3}{2}n - 2 = O(n^2)$$

сравнений имен.

Аналогичный (но более сложный) анализ показывает, что среднее число $M_{\text{ave}}(n)$ обменов «$x_i \leftrightarrow x_j$» и «$x_f \leftrightarrow x_j$»

$$M_{\text{ave}}(n) = \frac{1}{6}n + \frac{2}{3} + \frac{2}{n}\sum_{s=0}^{n-1} M_{\text{ave}}(s) \approx 0.231 n \log n ,$$

т. е. среднее число обменов имен равно $O(n \log n)$. В худшем случае $M_{\max}(n) \le n \log n$, т. е. $M_{\max}(n) = O(n \log n)$.

Таким образом, рекурсивный алгоритм быстрой сортировки требует $O(n \log n)$ операций в среднем, но $O(n^2)$ операций в худшем случае. Кроме того, поскольку рекурсия используется для записи подтаблиц, рассматривающихся на более поздних этапах, в худших случаях (когда таблица уже отсортирована) глубина рекурсии может равняться $n$. Следовательно, для стека, реализующего рекурсию, необходима память, пропорциональная $n$. Для больших $n$ такое требование становится неприемлемым.

Очевидно, что основным источником неэффективности быстрой сортировки является выбор расщепляющего имени. В алгоритме расщепляющим именем является первое имя таблицы (подтаблицы). Можно несколько улучшить быструю сортировку, если использовать для разделения таблицы случайно выбранное имя $x_{\mathrm{rand}(f,\, l)}$. Для этого достаточно в алгоритм добавить операцию $x_f \leftrightarrow x_{\mathrm{rand}(f,\, l)}$ непосредственно перед операцией $i \leftarrow f + 1$. Дополнительное время для выбора случайного целого числа несущественно, а случайный выбор является вполне приемлемой защитой от наихудшей ситуации. Более сильным улучшением является использование для разделения таблицы *медианы малой случайной выборки* из $k$ (обычно $k$ нечетное) имен или даже *медианы медиан*. Медианой является среднее по величине имя в множестве из $k$ случайным образом выбранных из исходной таблицы имен. Например, если $k = 3$, то $C_{\mathrm{ave}}(n) \approx 1{,}188\, n \log n$.

Одним из наиболее эффективных алгоритмов внутренней сортировки является итерационный вариант быстрой сортировки (алгоритм 5.5), который свободен от некоторых недостатков рекурсивного алгоритма. В алгоритме стек $S$ ведется явно; элементом стека является пара $(f, l)$, показывающая, что нужно отсортировать имена $x_f, x_{f+1}, \ldots, x_l$. В стек помещается большая из двух подтаблиц и продолжается обработка меньшей подтаблицы. Это уменьшает глубину стека в худшем случае до $\lfloor \log(n+1)/3 \rfloor$, $n \geq 2$, т. е. для организации стека необходима память, пропорциональная $\log n$.

Учитывая то, что для небольших таблиц длины меньше некоторого $m$ среднее число сравнений имен в быстрой сортировке больше среднего числа сравнений, например, в простой сортировке вставками, для быстрой сортировки можно сделать значительное усовершенствование. Вместо того, чтобы использовать быструю сортировку для всех подтаблиц, можно применять ее только к подтаблицам длины не меньше $m$, а для подтаблиц длины меньше $m$ воспользоваться простой сортировкой вставками.

$$
\begin{array}{l}
S \Leftarrow (0,\,0)\\
f \leftarrow 1\\
x_{n+1} \leftarrow \infty\\
l \leftarrow n\\
\textbf{while}\ f < l\ \textbf{do} \left\{
\begin{array}{l}
x_f \leftrightarrow x_{\mathrm{rand}(f,\,l)}\\
i \leftarrow f+1\\
\textbf{while}\ x_i < x_f\ \textbf{do}\ i \leftarrow i+1\\
j \leftarrow l\\
\textbf{while}\ x_j > x_f\ \textbf{do}\ j \leftarrow j-1\\
\textbf{while}\ i < j\ \textbf{do} \left\{
\begin{array}{l}
//\ \text{имеет место}\ i < j,\, x_i \ge x_f \ge x_j\}\\
x_i \leftrightarrow x_j\\
i \leftarrow i+1\\
\textbf{while}\ x_i < x_f\ \textbf{do}\ i \leftarrow i+1\\
j \leftarrow j-1\\
\textbf{while}\ x_j > x_f\ \textbf{do}\ j \leftarrow j-1
\end{array}\right.\\
x_f \leftrightarrow x_j\\
\textbf{case} \left\{
\begin{array}{l}
j-1 \le f\ \textbf{and}\ l \le j+1 : \left\{
\begin{array}{l}
//\ \text{обе подтаблицы}\\
//\ \text{тривиальны}\\
(f,\,l) \Leftarrow S
\end{array}\right.\\
j-1 \le f\ \textbf{and}\ l > j+1 : \left\{
\begin{array}{l}
//\ \text{нетривиальна только}\\
//\ \text{правая подтаблица}\\
f \leftarrow j+1
\end{array}\right.\\
j-1 > f\ \textbf{and}\ l \le j+1 : \left\{
\begin{array}{l}
//\ \text{нетривиальна только}\\
//\ \text{левая подтаблица}\\
l \leftarrow j-1
\end{array}\right.\\
j-1 > f\ \textbf{and}\ l > j+1 : \left\{
\begin{array}{l}
//\ \text{обе подтаблицы}\\
//\ \text{нетривиальны,}\\
//\ \text{поместить большую}\\
//\ \text{в стек}\ S\\
\\
\textbf{if}\ \ j-f > l-j\\
\quad \textbf{then} \left\{
\begin{array}{l}
//\ \text{левая в}\ S\\
S \Leftarrow (f,\, j-1)\\
f \leftarrow j+1
\end{array}\right.\\
\quad \textbf{else} \left\{
\begin{array}{l}
//\ \text{правая в}\ S\\
S \Leftarrow (j+1,\, l)\\
l \leftarrow j-1
\end{array}\right.
\end{array}\right.
\end{array}\right.
\end{array}\right.
\end{array}
$$

Алгоритм 5.5. Итерационный алгоритм быстрой сортировки

Интересным аналогом быстрой сортировки является так называемая *цифровая обменная сортировка*. В этом методе сортировки используется двоичное представление имен. Разделение таблицы осуществляется на основании самого старшего разряда имен: если этот разряд равен единице, то имя пересылается в правую часть таблицы, если этот разряд равен нулю – в левую часть таблицы. На следующих этапах расщепления основываются на следующих разрядах имен и т. д. Таким образом, метод цифровой обменной сортировки позволяет исключить проблему выбора расщепляющего имени.