

## 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.

$b \leftarrow n$

**while**  $b \neq 0$  **do**  $\left\{ \begin{array}{l} t \leftarrow 0 \\ \text{for } j \leftarrow 1 \text{ to } b-1 \text{ do} \\ \quad \text{if } x_j > x_{j+1} \text{ then } \left\{ \begin{array}{l} x_j \leftrightarrow x_{j+1} \\ t \leftarrow j \end{array} \right. \\ b \leftarrow t \end{array} \right.$

Алгоритм 5.3. Пузырьковая сортировка

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.

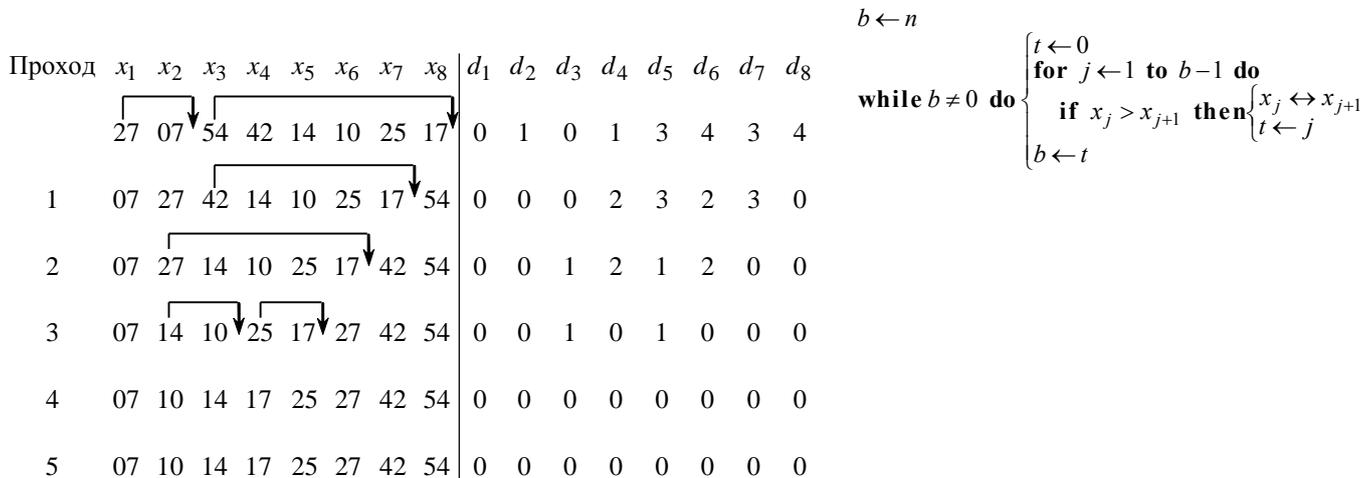


Рис. 5.4. Процесс работы алгоритма пузырьковой сортировки

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.

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

Следовательно, в лучшем случае имеется всего один проход, в худшем случае –  $n$  проходов. Для определения среднего числа проходов необходимо найти математическое ожидание  $\sum_{k=1}^n k P_k$ , 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  $k$  ( $1 \leq k \leq n$ ), 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

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

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,

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

$C_{\max}(n) = \frac{1}{2}n(n-1) = O(n^2)$  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  $x_f, x_{f+1}, \dots, x_l$ , где  $x_f$  является базовым именем, используемым для разбиения таблицы на подтаблицы.

```

procedure QUICKSORT ( $f, l$ )
    // отсортировать  $x_f, x_{f+1}, \dots, x_l$ 
    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  $j \leftarrow j - 1$ 
    while  $i < j$  do {
        // в этот момент  $i < j, x_i \geq x_f \geq x_j$ 
         $x_i \leftrightarrow x_j$ 
         $i \leftarrow i + 1$ 
        while  $x_i < x_f$  do  $i \leftarrow i + 1$ 
         $j \leftarrow j - 1$ 
        while  $x_j > x_f$  do  $j \leftarrow j - 1$ 
    }
     $x_f \leftrightarrow x_j$ 
    // отсортировать подтаблицы рекурсивно
    QUICKSORT ( $f, j - 1$ )
    QUICKSORT ( $j + 1, l$ )
return

```

Алгоритм 5.4. Рекурсивный алгоритм быстрой сортировки

Алгоритм предполагает, что имя  $x_{i+1}$  определено и больше, чем  $x_f, x_{f+1}, \dots, x_l$ , 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.

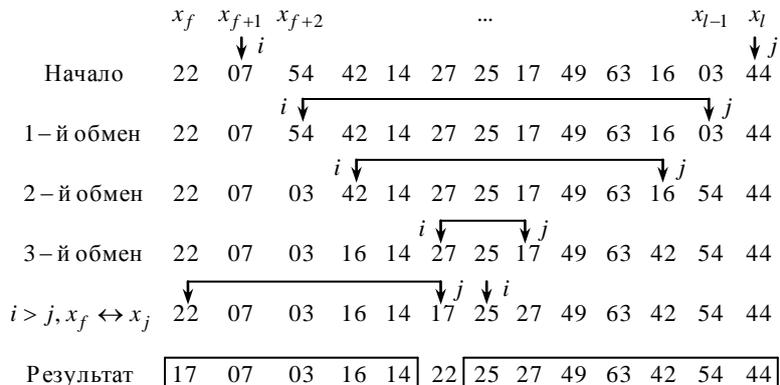


Рис. 5.5. Процесс разбиения таблицы на две подтаблицы

Определим общее число сравнений имен « $x_i < x_j$ » и « $x_j > x_i$ ». В конце цикла «**while**  $i < j$ » все имена  $x_{f+1}, x_{f+2}, \dots, x_l$  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

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

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,

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

Поскольку эта сумма равна

$$C_{\text{ave}}(0) + C_{\text{ave}}(n-1) + C_{\text{ave}}(1) + C_{\text{ave}}(n-2) + \dots + C_{\text{ave}}(n-2) + C_{\text{ave}}(1) + C_{\text{ave}}(n-1) + C_{\text{ave}}(0),$$

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

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

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

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

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  $M_{\text{max}}(n) \leq n \log n$ , т. е.  $M_{\text{max}}(n) = O(n \log n)$ .

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  $x_{\text{rand}(f, l)}$ . Для этого достаточно в алгоритм добавить операцию  $x_f \leftrightarrow x_{\text{rand}(f, l)}$  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  $C_{\text{ave}}(n) \approx 1,188 n \log n$ .

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  $x_f, x_{f+1}, \dots, x_l$ . 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.

$S \leftarrow (0, 0)$   
 $f \leftarrow 1$   
 $x_{n+1} \leftarrow \infty$   
 $l \leftarrow n$

```

{
   $x_f \leftrightarrow x_{\text{rand}(f, l)}$ 
   $i \leftarrow f + 1$ 
  while  $x_i < x_f$  do  $i \leftarrow i + 1$ 
   $j \leftarrow l$ 
  while  $x_j > x_f$  do  $j \leftarrow j - 1$ 
  while  $i < j$  do {
    // имеет место  $i < j, x_i \geq x_f \geq x_j$ 
     $x_i \leftrightarrow x_j$ 
     $i \leftarrow i + 1$ 
    while  $x_i < x_f$  do  $i \leftarrow i + 1$ 
     $j \leftarrow j - 1$ 
    while  $x_j > x_f$  do  $j \leftarrow j - 1$ 
  }
   $x_f \leftrightarrow x_j$ 
  while  $f < l$  do {
    case {
       $j - 1 \leq f$  and  $l \leq j + 1$ : { // обе подтаблицы
        // тривиальны
         $(f, l) \leftarrow S$ 
      }
       $j - 1 \leq f$  and  $l > j + 1$ : { // нетривиальна только
        // правая подтаблица
         $f \leftarrow j + 1$ 
      }
       $j - 1 > f$  and  $l \leq j + 1$ : { // нетривиальна только
        // левая подтаблица
         $l \leftarrow j - 1$ 
      }
      // обе подтаблицы
      // нетривиальны,
      // поместить большую
      // в стек S
       $j - 1 > f$  and  $l > j + 1$ : {
        if  $j - f > l - j$ 
        then { // левая в S
           $S \leftarrow (f, j - 1)$ 
           $f \leftarrow j + 1$ 
        }
        else { // правая в S
           $S \leftarrow (j + 1, l)$ 
           $l \leftarrow j - 1$ 
        }
      }
    }
  }
}

```

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.