Тема 5. Сортировка

5.2. Сортировка вставками

Сортировка вставками основана на том, что имена таблицы просматриваются по одному, и на каждом этапе просмотра новое имя вставляется в подходящее место среди ранее отсортированных имен. Рассмотрим два алгоритма из этого класса.

5.2.1. Простая сортировка вставками

Алгоритм 5.1 представляет собой *простую сортировку вставками*. Процесс сортировки проходит через этапы $j = 2, 3, \ldots, n$; на этапе j имя x_j вставляется на свое правильное место среди $x_1, x_2, \ldots, x_{j-1}$ уже отсортированных имен. При вставке имя x_j временно размещается в переменной t и просматриваются имена $x_{j-1}, x_{j-2}, \ldots, x_1$; они сравниваются с t и сдвигаются вправо, если они больше t. Фиктивное имя-сторож $x_0 = -\infty$ служит для остановки просмотра слева. На рис. 5.2 показан процесс работы этого алгоритма на примере таблицы из $n = 8$ имен.

$$
\begin{array}{l}
x_0 \leftarrow -\infty \\
\textbf{for}\ \ j \leftarrow 2\ \ \textbf{to}\ \ n\ \ \textbf{do}
\begin{cases}
i \leftarrow j-1 \\
t \leftarrow x_j \\
\textbf{while}\ t < x_i\ \ \textbf{do}
\begin{cases}
x_{i+1} \leftarrow x_i \\
i \leftarrow i-1
\end{cases} \\
x_{i+1} \leftarrow t
\end{cases}
\end{array}
$$

Алгоритм 5.1. Простая сортировка вставками

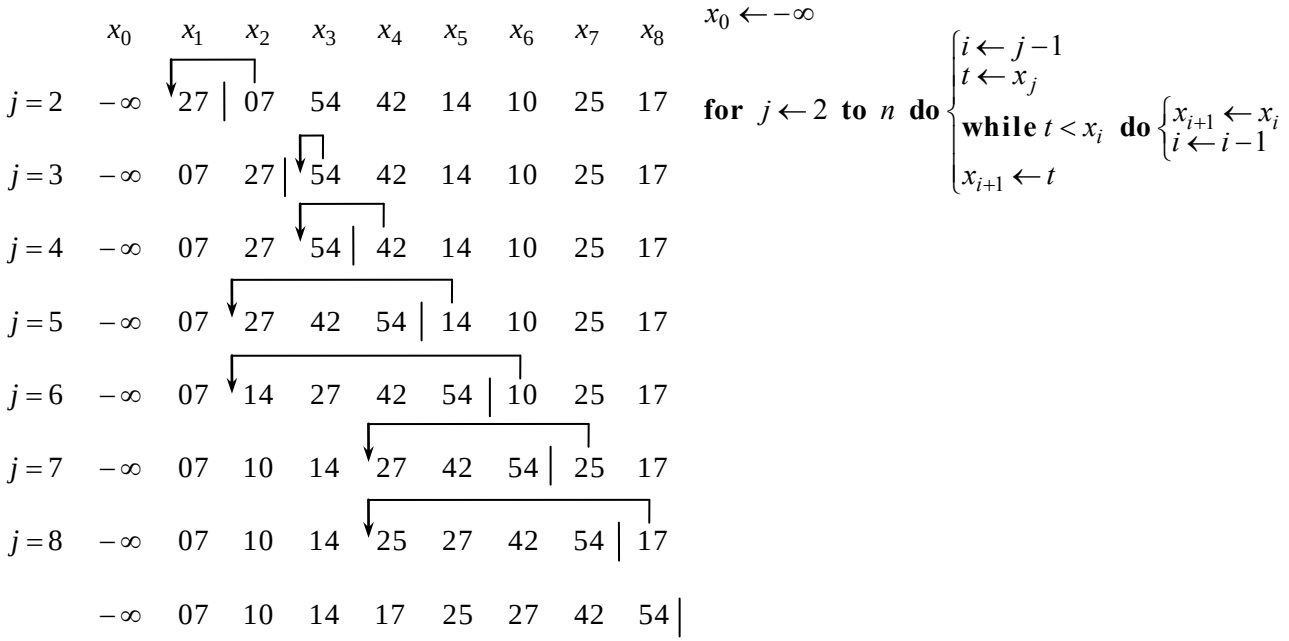


Рис. 5.2. Процесс работы алгоритма простой сортировки вставками

Эффективность этого алгоритма, как и большинства алгоритмов сортировки, зависит от числа сравнений имен и числа пересылок данных, осуществляемых в трех случаях: худшем, среднем (в предположении, что все $n!$ перестановок равновероятны) и лучшем. Ключом к анализу алгоритма является число проверок условия «$t < x_i$» в цикле **while**, поскольку, если это число известно, легко выводится число пересылок имен «$t \leftarrow x_j$», «$x_{i+1} \leftarrow x_i$» и «$x_{i+1} \leftarrow t$».

Число сравнений имен $C(n)$ (число проверок условия «$t < x_i$») полностью зависит от значений элементов вектора инверсий $D = (d_1, d_2, \ldots, d_n)$ для сортируемой таблицы, где d_j – число имен, больших x_j и расположенных слева от него. На j-м этапе выполняется $C_j = d_j + 1$ сравнений имен, а всего выполняется $n - 1$ этапов предопределенного цикла **for** ($j = 2, \ldots, n$). Следовательно,

$$C(n) = \sum_{j=2}^{n}(d_j + 1) = n - 1 + \sum_{j=2}^{n} d_j .$$

Так как $0 \le d_j \le j - 1$ и величины d_j независимы, сумма элементов вектора инверсий $\sum_{j=2}^{n} d_j$ имеет минимальное значение, равное нулю, в случае уже упорядоченной исходной таблицы, максимальное значение, равное

$$0 + 1 + 2 + \ldots + n - 1 = \sum_{j=1}^{n-1} j = \frac{1}{2} n(n-1),$$

имеет в случае, когда элементы исходной таблицы первоначально расположены в обратном порядке. Среднее значение $\sum_{j=2}^{n} d_j$ по всем перестановкам равно $\frac{1}{2}\sum_{j=1}^{n-1} j = \frac{1}{4} n(n-1)$. Таким образом, общее число сравнений имен, осуществляемых алгоритмом простой сортировки вставками, составляет

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

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

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

Что касается числа пересылок $M(n)$ (присваиваний имен), то очевидно, что на j-м этапе выполняется $C_j + 1$ пересылок ($C_j - 1$ пересылок в цикле **while** и две пересылки «$t \leftarrow x_j$» и «$x_{i+1} \leftarrow t$»). Следовательно,

$$M(n) = \sum_{j=2}^{n}(C_j + 1) = \sum_{j=2}^{n}(d_j + 2) = 2(n-1) + \sum_{j=2}^{n} d_j = (n-1) + C(n).$$

Таким образом, общее число пересылок имен составляет

$M_{\min}(n) = 2(n - 1) = O(n)$ в лучшем случае,

$M_{\text{ave}}(n) = \frac{1}{4}(n-1)(n+8) = O(n^2)$ в среднем,

$M_{\max}(n) = \frac{1}{2}(n-1)(n+4) = O(n^2)$ в худшем случае.

Проведенный анализ показывает, что алгоритм простой сортировки вставками 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.

Алгоритм сортировки вставками можно достаточно просто усовершенствовать. Учитывая, что вставка j-го имени осуществляется среди $x_1, x_2, \ldots, x_{j-1}$ уже отсортированных имен, можно использовать бинарный поиск для определения места вставки (*сортировка бинарными вставками*). В этом случае уменьшается число сравнений имен до $O(n \log n)$, но сама вставка требует $O(n^2)$ пересылок имен, т. е. алгоритму все равно потребуется порядка n^2 операций. Следует отметить, что сортировка бинарными вставками уже отсортированной таблицы потребует больше времени, чем использование простой сортировки вставками.

Другой путь усовершенствования заключается в использовании связного списка для представления таблицы (*сортировка вставками в связный список*). Этим достигается более эффективная вставка. Однако в связном списке невозможен бинарный поиск для определения места вставки, можно использовать только последовательный поиск. Для такого алгоритма сокращается число пересылок, но число сравнений все равно остается порядка n^2, т. е. алгоритму все равно потребуется $O(n^2)$ операций.

5.2.2. Сортировка Шелла

Основная причина неэффективности алгоритма простой сортировки вставками заключается в том, что в каждый момент времени имена сдвигаются только на одну позицию. Поэтому для улучшения необходим некоторый механизм, с помощью которого имена могли бы перемещаться на большие расстояния. Такой механизм используется в *сортировке с убывающим шагом*, названной в честь ее изобретателя *сортировкой Шелла*.

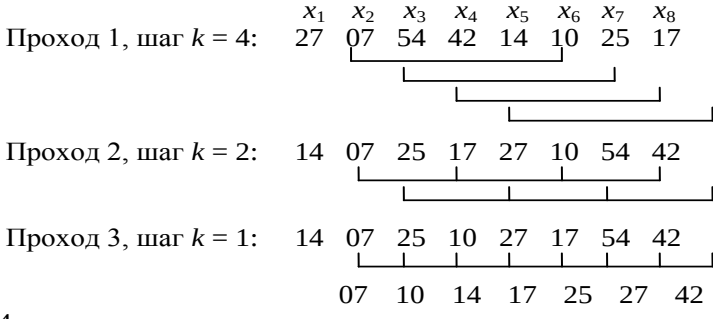
Такая сортировка заключается в следующем. Задается последовательность шагов, представляющая собой последовательность положительных целых чисел $h_1 > h_2 > \ldots > h_t = 1$. Необходимо обратить внимание на то, что последнее значение этой последовательности обязательно должно быть равно единице. Сначала отдельно группируются и сортируются (обычно при помощи метода простых вставок) имена, отстоящие друг от друга на расстоянии h_1 (h_1-сортировка). Затем имена перегруппировываются в соответствии с шагом h_2 и снова сортируются (h_2-сортировка). Процесс h-сортировок продолжается до тех пор пока не выполнится h_t-сортировка. На последнем проходе (при h_t-сортировке) все имена таблицы образуют одну группу, поскольку $h_t = 1$. Детали сортировки Шелла показаны в алгоритме 5.2. В этом алгоритме для упрощения условия окончания поиска для определения места вставки использованы имена-сторожа. Очевидно, что каждая h-сортировка нуждается в своем собственном имени-стороже, поэтому приходится расширять исходную таблицу не на один компонент x_0 (как это делается в алгоритме простой сортировки вставками), а на $h_1 - 1$ компонентов.

for $i \leftarrow -(h_1 - 1)$ **to** 0 **do** $x_i \leftarrow -\infty$ // установка сторожей

for $l \leftarrow 1$ **to** t **do**
- // h_l – сортировка
- $k \leftarrow h_l$
- **for** $j \leftarrow k + 1$ **to** n **do**
  - $i \leftarrow j - k$
  - $t \leftarrow x_j$
  - **while** $t < x_i$ **do**
    - $x_{i+k} \leftarrow x_i$
    - $i \leftarrow i - k$
  - $x_{i+k} \leftarrow t$

Алгоритм 5.2. Сортировка Шелла

Процесс работы алгоритма сортировки Шелла для последовательности шагов (4, 2, 1) представлен на рис. 5.3 (имена-сторожа не указаны). На первом проходе отдельно сортируются четыре группы по два имени в каждой группе: (x_1, x_5), (x_2, x_6), (x_3, x_7), (x_4, x_8). На втором проходе сортируются две группы по четыре имени: (x_1, x_3, x_5, x_7), (x_2, x_4, x_6, x_8). Процесс завершается третьим проходом, во время которого сортируются все восемь имен.




```
for i ← −(h_1 − 1) to 0 do x_i ← −∞   // установка сторожей
for l ← 1 to t do
    // h_l − сортировка
    k ← h_l
    for j ← k + 1 to n do
        i ← j − k
        t ← x_j
        while t < x_i do
            x_{i+k} ← x_i
            i ← i − k
        x_{i+k} ← t
```


Рис. 5.3. Процесс работы алгоритма сортировки Шелла

Хорошая эффективность сортировки Шелла объясняется тем, что на первых проходах, когда шаги являются большими, сортируемые группы имен малы, поэтому сортировка вставками работает достаточно быстро. Сортировка таких подгрупп приводит к тому, что вся таблица становится ближе к отсортированному виду, т. е. существенно уменьшается число инверсий. Поэтому на последующих проходах, хотя и используются шаги с меньшими значениями и, следовательно, сортируются большие группы имен, тем не менее сортировка вставками остается достаточно эффективной.

Анализ эффективности сортировки Шелла математически сложен. Приведем только некоторые результаты. Существенным фактором, влияющим на эффективность, является выбор последовательности шагов. Одно из требований состоит в том, что элементы в последовательности шагов должны быть взаимно простыми числами. Последовательность шагов рекомендуется выбирать следующим образом [8]:

$h_t = 1$, $h_{l-1} = 3h_l + 1$ и $t = \lfloor \log_3 n \rfloor - 1$, т. е. последовательность (в обратном порядке) 1, 4, 13, 40, … , тогда сортировка потребует времени $O(n(\log n)^2)$,

либо $h_t = 1$, $h_{l-1} = 2h_l + 1$ и $t = \lfloor \log_2 n \rfloor - 1$, т. е. последовательность (в обратном порядке) 1, 3, 7, 15, … , с временной сложностью $O(n^{1,2})$.