Тема 5. Сортировка

5.7. Сортировка слиянием

Слияние является процессом объединения двух или более упорядоченных таблиц в одну упорядоченную таблицу. Ограничимся рассмотрением задачи слияния только двух таблиц, поскольку рассматриваемые методы достаточно легко можно обобщить для большего числа таблиц. Таким образом, необходимо решить задачу слияния двух отсортированных таблиц $x_1 \le x_2 \le \ldots \le x_n$ и $y_1 \le y_2 \le \ldots \le y_m$ в одну отсортированную таблицу $z_1 \le z_2 \le \ldots \le z_{n+m}$.

Очевидный способ слияния, называемый *прямым слиянием*, заключается в следующем: таблицы, подлежащие слиянию, просматривают параллельно, выбирая на каждом шаге меньшее из двух имен и помещая его в выходную таблицу. В алгоритме 5.10 этот процесс упрощен добавлением имен-сторожей x_{n+1} и y_{m+1}. Переменные i и j указывают соответственно на последние имена в двух входных таблицах, которые еще не были помещены в выходную таблицу.

$$
\begin{array}{l}
x_{n+1} \leftarrow y_{m+1} \leftarrow \infty \\
i \leftarrow j \leftarrow 1 \\
\textbf{for } k \leftarrow 1 \textbf{ to } n+m \textbf{ do} \left\{ \begin{array}{l}
\textbf{if } x_i < y_j \\
\quad \textbf{then} \left\{ \begin{array}{l} z_k \leftarrow x_i \\ i \leftarrow i+1 \end{array} \right. \\
\quad \textbf{else} \left\{ \begin{array}{l} z_k \leftarrow y_i \\ j \leftarrow j+1 \end{array} \right.
\end{array} \right.
\end{array}
$$

Алгоритм 5.10. Прямое слияние двух отсортированных таблиц

Анализ этого алгоритма прост. Сравнение имен «$x_i < y_j$» производится точно один раз для каждого имени, помещенного в выходную таблицу, т. е. выполняется $n + m$ сравнений. Среднее число сравнений имен можно несколько уменьшить, если отказаться от имен-сторожей x_{n+1} и y_{m+1}, а на каждом шаге проверять, достигли соответствующие индексы правых границ исходных таблиц или нет. Если на некотором этапе одна из таблиц полностью исчерпана (все ее имена помещены в выходную таблицу), оставшиеся имена из другой таблицы просто перемещаются в выходную таблицу без сравнения имен. Очевидно, что такой алгоритм в худшем случае потребует $n + m - 1$ сравнений имен.

При $n \approx m$ прямое слияние является оптимальным. Если же одна из таблиц существенно больше другой, существует более эффективный метод слияния, называемый *бинарным слиянием*. Например, при $m = 1$ таблицы можно слить, используя бинарный поиск для отыскания места, куда нужно вставить y_1.

Пусть $n \geq m$. Идея бинарного слияния состоит в том, чтобы в самой правой части большей таблицы выделить подтаблицу, в которой будет осуществляться бинарный поиск места вставки последнего имени y_m из меньшей таблицы. Поскольку бинарный поиск наиболее эффективно работает на таблицах размера $2^k - 1$, предпочтительнее вариант, когда в выделяемой подтаблице будет $2^{\lfloor \log n/m \rfloor} - 1$ имен. Пусть имя x_l непосредственно предшествует выделенной подтаблице, т. е. $l = n + 1 - 2^{\lfloor \log n/m \rfloor}$. Выполняется сравнение самого правого имени y_m из меньшей таблицы с именем x_l. Если $y_m < x_l$, тогда x_l и вся выделенная подтаблица большей таблицы вкладываются в выходную таблицу. Если $y_m \geq x_l$, то осуществляется бинарный поиск места вставки имени y_m в выделенной подтаблице (обозначим через k наибольшее целое, такое, что $y_m > x_k$). Затем y_m и все имена $x_{k+1}, \ldots, x_n$ (т. е. имена из подтаблицы, о которых стало известно, что они больше y_m), помещаются в выходную таблицу. Далее процесс повторяется для изменившихся размеров m и n таблиц.

На рис. 5.13 показан первый шаг слияния двух таблиц: $X = \{x_1, \ldots, x_{20}\}$ и $Y = \{y_1, \ldots, y_4\}$. В большей таблице X выделяется подтаблица размера $2^{\lfloor \log 20/4 \rfloor} - 1 = 3$, состоящая из имен x_{18}, x_{19}, x_{20}. Если $y_4 < x_l = x_{17}$, то $x_{17}, x_{18}, x_{19}, x_{20}$ можно поместить в выходную таблицу, и продолжается слияние таблиц $x_1, x_2, \ldots, x_{16}$ с y_1, y_2, y_3, y_4. Если $y_4 \geq x_{17}$, то используется бинарный поиск для отыскания позиции y_4 среди x_{18}, x_{19}, x_{20} за два сравнения и помещаются y_4 и все $x_i > y_4$ в выходную таблицу. Процесс слияния продолжается для таблиц $x_1, x_2, \ldots, x_k$ и y_1, y_2, y_3, где k – наибольшее целое, такое что $y_4 > x_k$.

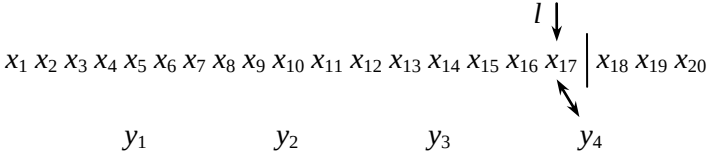

Рис. 5.13. Один шаг бинарного слияния

Общая процедура бинарного слияния представлена алгоритмом 5.11, где альтернативы **then** и **else**, относящиеся к «**if** $m \leq n$», одинаковы во всем, кроме того, что роли x и y (соответственно n и m) меняются местами.

Число сравнений имен, производимых алгоритмом бинарного слияния при объединении таблиц $x_1, x_2, \ldots, x_n$ и $y_1, y_2, \ldots, y_m$ в худшем случае, составляет [8]

$$C(m, n) = m + \lfloor n / 2^t \rfloor - 1 + t\,m \quad \text{при } m \leq n, \text{ где } t = \lfloor \log n/m \rfloor.$$

При $m = n$ имеем $C(n, n) = 2n - 1$, а при $m = 1$ получаем

$$C(1, n) = 1 + \lfloor n / 2^{\lfloor \log n \rfloor} \rfloor - 1 + \lfloor \log n \rfloor = 1 + \lfloor \log n \rfloor,$$

т. е. алгоритм бинарного слияния работает как прямое слияние при $m = n$ и как бинарный поиск при $m = 1$, а также достаточно эффективно работает для промежуточных значений m.

while $n \neq 0$ **and** $m \neq 0$ **do**

if $m \leq n$

then

$t \leftarrow \lfloor \log\ n/m \rfloor$

if $y_m < x_{n+1-2^t}$

then

Поместить $x_{n+1-2^t}, \ldots, x_n$
в выходную таблицу

$n \leftarrow n - 2^t$

else

Используя t сравнений имен, бинарным
поиском в $x_{n+2-2^t}, \ldots, x_n$
определить k – наибольшее
целое, такое, что $y_m > x_k$.

Поместить $y_m, x_{k+1}, \ldots, x_n$
в выходную таблицу

$m \leftarrow m - 1$

$n \leftarrow k$

else

$t \leftarrow \lfloor \log\ m/n \rfloor$

if $x_n < y_{m+1-2^t}$

then

Поместить $y_{m+1-2^t}, \ldots, y_m$
в выходную таблицу

$m \leftarrow m - 2^t$

else

Используя t сравнений имен, бинарным
поиском в $y_{m+2-2^t}, \ldots, y_m$
определить k – наибольшее
целое, такое, что $x_n > y_k$.

Поместить $x_n, y_{k+1}, \ldots, y_m$
в выходную таблицу

$n \leftarrow n - 1$

$m \leftarrow k$

if $n = 0$

then Поместить $y_1, y_2, \ldots, y_m$ в выходную таблицу

else Поместить $x_1, x_2, \ldots, x_n$ в выходную таблицу

Алгоритм 5.11. Бинарное слияние

Рассмотрим применение слияния для сортировки таблицы $T = \{x_1, x_2, \ldots, x_n\}$. Поскольку для слияния необходимо иметь по крайней мере две отсортированные последовательности, возникает вопрос о разбиении исходной таблицы на упорядоченные подтаблицы и их слияния. Такие упорядоченные сегменты данных называются *отрезками* (или *сериями*).

Одним из вариантов сортировки слиянием является *естественное двухпутевое слияние*, которое выбирает из исходной таблицы упорядоченные подпоследовательности (отрезки) и объединяет их в более длинные. Процесс сортировки проиллюстрирован на рис. 5.14, где вертикальными линиями отмечены границы между отрезками, а стрелки показывают направление упорядочения внутри отрезков.

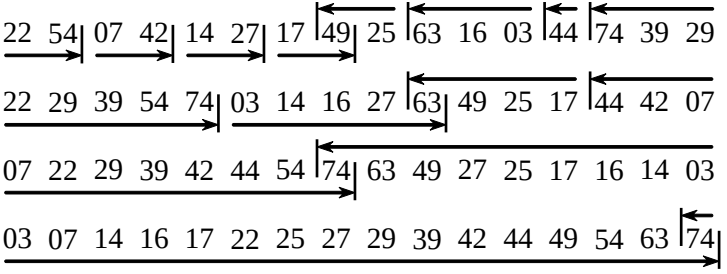

Рис. 5.14. Процесс сортировки естественным двухпутевым слиянием

Исходная таблица анализируется слева и справа, двигаясь к середине. Это необходимо для того, чтобы был доступ к двум отрезкам для слияния: один отрезок в левой части, а второй – в правой части таблицы, причем отрезок правой части должен быть упорядочен справа налево. Вторая строка формируется из первой следующим образом. Слева имеется отрезок (22, 54), а справа, если читать справа налево, отрезок (29, 39, 74). Слияние этих отрезков дает последовательность (22, 29, 39, 54, 74), которая помещается в левую часть вспомогательной таблицы (строка 2). Затем отрезок (07, 42) сливается с отрезком (44) и результат (07, 42, 44) записывается в правую часть вспомогательной таблицы. Результат (03, 14, 16, 27, 63) слияния отрезков (14, 27) и (03, 16, 63) помещается в левую часть вслед за ранее записанной последовательностью. Наконец, отрезок (17, 49) сливается с отрезком (25, 49) и полученный результат (17, 25, 49) записывается в правую часть перед ранее записанной последовательностью. Затем производится переключение таблиц (вспомогательная таблица становится исходной, а исходная – вспомогательной) и процесс анализа, слияния и переключения повторяется до тех пор, пока не будет сформирован единственный отрезок – отсортированная таблица.

В общем случае в середине таблицы возникает перекрытие отрезков, когда при движении с обоих концов считывается одно и то же имя. Такая ситуация должна обнаруживаться алгоритмом прежде, чем она может привести к осложнениям.

Таким образом, для реализации естественного двухпутевого слияния необходимы две таблицы размера n (исходная и вспомогательная). Причем результат сортировки может сформироваться в любой из них. Если определено требование, что результат должен быть в исходной таблице, то придется выполнить копирование данных из вспомогательной таблицы в исходную.

Чтобы упростить переключение таблиц, для хранения второй таблицы обычно отводят область памяти, располагающуюся вслед за исходной таблицей, т. е. $x_{n+1}, \ldots, x_{2n}$. Тогда переключение таблиц реализуется соответствующей установкой значений индексов.

Для переключения направления вывода (по возрастанию индекса – в левой части таблицы и по убыванию – в правой) достаточно использовать переменную для хранения шага приращения индекса и изменять ее знак при переключении.

Естественное двухпутевое слияние обладает тем преимуществом, что исходные таблицы с преобладанием возрастающего или убывающего расположения имен обрабатываются очень быстро. Но при этом приходится постоянно проверять, достигнут конец отрезка или нет (их длина заранее не известна и определяется случайным расположением имен), что приводит к замедлению процесса сортировки в общем случае. Чтобы этого избежать, можно использовать на каждом шаге фиксированные длины отрезков. В исходной таблице все отрезки имеют длину 1, после первого шага все отрезки (кроме, возможно, последнего) имеют длину 2, ... , после k-го шага – длину 2^k (кроме, возможно, последнего). Такой способ называется *простым двухпутевым слиянием*. В остальном общая схема слияния аналогична естественному слиянию. Пример простого двухпутевого слияния показан на рис. 5.15.

Поскольку длина отрезков фиксирована для каждого шага, можно заранее предсказать положение отсортированной таблицы (в исходной или в дополнительной): если значение $\lceil \log n \rceil$ нечетно, то результат окажется в дополнительной таблице, если четно – в исходной.

Время работы алгоритмов слияния (как естественного, так и простого) составляет $O(n \log n)$. Однако в среднем простое слияние несколько лучше естественного.

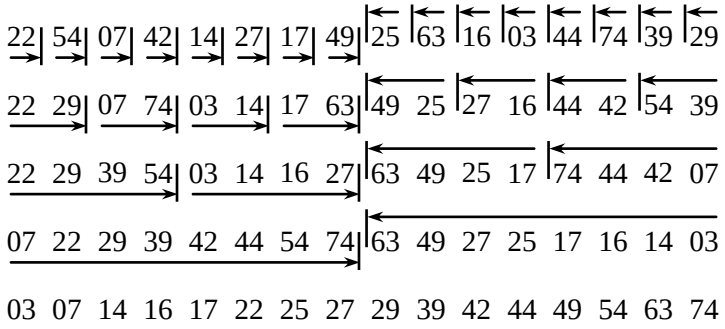

Рис. 5.15. Процесс сортировки простым двухпутевым слиянием

Рассмотренные ранее методы сортировки слиянием требуют памяти для хранения $2n$ имен. При этом в любой момент времени половина памяти не используется, а остается зарезервированной для последующего заполнения. Более логичной структурой данных является связный список. Тогда все необходимые операции перемещения имен можно заменить операциями со связями. Как правило, добавление n полей связи выгоднее добавления пространства памяти еще для n имен, так как отказавшись от перемещения имен, можно выиграть во времени. В процессе сортировки поля связи устанавливаются таким образом, что имена оказываются связанными в порядке возрастания. Такая сортировка называется *сортировкой слиянием списков*. Для реализации такой сортировки связные списки удобнее представлять с помощью массивов, используя индекс массива в качестве указателя на элемент массива, следующего за текущим элементом.

Слияние списков можно реализовать как для естественного, так и для простого слияния. Причем при использовании связного распределения естественное слияние предпочтительнее простого.

Хорошо известен и широко используется рекурсивный вариант сортировки слиянием, представленный алгоритмом 5.12. Процедура *MERGE* выполняет слияние двух отсортированных подтаблиц (необходима такая модификация алгоритма 5.10, чтобы результат слияния получался в исходной таблице). Процедура выполняет сортировку подтаблицы $x_i, x_{i+1}, \ldots, x_j$, состоящей из k имен ($k = j - i + 1$). Если $i \geq j$, подтаблица содержит не более одного имени, т. е. она отсортирована. В противном случае производится разбиение на две подтаблицы: $x_i, \ldots, x_m$ с $\lceil k/2 \rceil$ именами и $x_{m+1}, \ldots, x_j$ с $\lfloor k/2 \rfloor$ именами. Для сортировки исходной таблицы из n имен первым обращением к процедуре является *MERGESORT* (1, n). В ходе работы алгоритма производится попарное слияние одноэлементных отрезков в отрезок длиной 2, затем – двухэлементных отрезков в отрезок длиной 4 и так до тех пор, пока не будет получен один отрезок длиной n, т. е. отсортированная таблица. Таким образом, процедура реализует, по сути, простое двухпутевое слияние.

procedure *MERGESORT*(i, j)
  if $i < j$
    then $\begin{cases} m \leftarrow \lfloor (i+j)/2 \rfloor \\ MERGE(MERGESORT(i, m), MERGESORT(m+1, j)) \end{cases}$
return

Алгоритм 5.12. Рекурсивный вариант сортировки слиянием

Время работы сортировки слиянием в худшем случае определяется рекуррентным соотношением

$$T(1) = O(1),$$
$$T(n) = 2T(n/2) + n - 1,$$

решением которого является $T(n) = O(n \log n)$ (см. п. 1.4).

Исследование процесса работы данного алгоритма предлагается в качестве упражнения.

5.8. Гибридный алгоритм сортировки Timsort

Большое распространение получила сортировка *Timsort*, опубликованная в 2002 году Тимом Петерсом. В настоящее время она является стандартным алгоритмом сортировки в Python, OpenJDK 7 и реализован в Android JDK 1.5. *Timsort* представляет собой гибридный алгоритм сортировки, сочетающий сортировку вставками и сортировку слиянием. Идея алгоритма заключается в следующем. По специальному алгоритму сортируемая таблица разделяется на подтаблицы. Каждая подтаблица сортируется простой сортировкой вставками. Отсортированные подтаблицы (отрезки) собираются в единую таблицу с помощью модифицированной сортировки слиянием.

Минимальный размер подтаблиц k определяется исходя из следующих принципов: k не должно быть слишком большим (к подтаблице будет применена сортировка вставками, а она эффективна только на небольших массивах) и не должно быть слишком маленьким (чем больше число сформированных отрезков, тем больше операций слияния придётся выполнить). Оптимальное значение для n / k – это степень числа 2 (или близкая к нему). Это требование обусловлено тем, что алгоритм слияния отрезков наиболее эффективно работает на отрезках примерно равного размера. Автор алгоритма ссылается на собственные эксперименты, показавшие, что наиболее эффективно использовать значения k из диапазона (32;65). Исключение – если $n < 64$, тогда $k = n$ и *Timsort* превращается в простую сортировку вставками.

Процесс формирования подтаблиц начинается с первого имени таблицы. Выполняется поиск упорядоченной подпоследовательности (отрезка). Если получившаяся подпоследовательность упорядочена по убыванию, имена переставляются так, чтобы они шли по возрастанию. Если размер текущего отрезка меньше k, подтаблица дополняется следующими за найденным отрезком именами до размера k. В результате будет получена подтаблица размером k или больше, к которой применяется сортировка вставками. Начиная со следующего за сформированным отрезком имени, аналогично формируется новый отрезок и так до тех пор, пока вся таблица не будет разбита на отрезки.

Завершающим этапом алгоритма является слияние полученных отрезков. Следует объединять отрезки примерно равного размера. Для этого создается пустой стек для хранения пар (b, s), где b – номер позиции начала отрезка, s – размер отрезка. В стек добавляется (b, s) первого отрезка. Пусть на некотором этапе в верхней части стека находятся данные о трех отрезках X, Y и Z с размерами s_X, s_Y и s_Z соответственно. Для определения, необходимо ли выполнять слияние отрезков, проверяется выполнение двух правил: $s_X > s_Y + s_Z$ и $s_Y > s_Z$. Если одно из правил нарушается, отрезок Y сливается с меньшим из отрезков X и Z. Процесс повторяется до выполнения обоих правил или полного упорядочивания данных. Если еще остались нерассмотренные отрезки, берётся следующий отрезок и его данные заносятся в стек и процесс повторяется. В противном случае процесс завершается.

Время работы алгоритма $O(n)$ в лучшем случае и $O(n \log n)$ в среднем и худшем случаях. Особенно хорошо алгоритм работает с частично отсортированными таблицами. Детали реализации алгоритма предлагаются в качестве упражнения.