Тема 5. Сортировка

5.9. Внешняя сортировка

Внешней называется сортировка, когда решается задача полной сортировки для случая такой большой таблицы, не умещающейся в оперативной памяти, что доступ к ней организован по частям, расположенным на внешних запоминающих устройствах. Внешняя сортировка файлов в корне отличается от внутренней, поскольку время доступа к файлам на внешних носителях накладывает существенные ограничения, кроме того, в каждый момент непосредственно доступна только одна запись (для последовательного файла) или блок записей (для файла с прямым доступом), но не весь файл целиком. Поэтому многие методы внутренней сортировки фактически бесполезны для внешней сортировки. Основой внешней сортировки является слияние. Сортировка слиянием применяется в основном для внешней, а не для внутренней сортировки, причем используется обычно многопутевое слияние.

При изучении основных идей внешней сортировки будем рассматривать только сортировку таблицы, организованной в виде последовательного файла как наихудшего с точки зрения доступа. Будем считать, что имеется $t + 1$ файлов, один из которых представляет собой таблицу имен $x_1, x_2, \ldots, x_n$. Остальные t файлов являются рабочими. Файл с исходной таблицей служит для ввода данных. Будем считать, что внутренняя память вместе с другими данными, программами и прочей информацией может содержать одновременно только m имен. Предполагается, что размер n входной таблицы значительно превышает m.

Общей стратегией внешней сортировки является использование внутренней памяти для сортировки имен из файла по частям, т. е. формирование множества упорядоченных подтаблиц (*исходных отрезков*) из исходной таблицы. По мере порождения эти отрезки распределяются по рабочим файлам, затем производится их слияние обратно в исходный файл так, что он будет содержать меньшее число более длинных отрезков. Далее полученные отрезки снова распределяются по рабочим файлам и снова производится их слияние и т. д. Процесс продолжается до тех пор, пока не получится единственный отрезок – отсортированная таблица. Таким образом, имеются две отдельные проблемы: как порождать исходные отрезки и как осуществлять слияние.

5.9.1. Порождение исходных отрезков

Самый простой метод порождения исходных отрезков заключается в том, что из исходной таблицы считываются *m* имен, производится их сортировка с использованием любого из методов внутренней сортировки и полученная отсортированная последовательность имен записывается в файл в виде отрезка; далее считываются следующие *m* имен, из которых формируется следующий отрезок, и так до тех пор, пока не будут исчерпаны все имена исходной таблицы. Все полученные таким образом отрезки содержат *m* имен, за исключением, возможно, последнего отрезка, в котором может оказаться менее *m* имен (когда *n* не кратно *m*).

Очевидно, что число исходных отрезков в конечном счёте определяет время слияния. Поэтому для повышения эффективности внешней сортировки необходимо стремиться к образованию меньшего количества более длинных исходных отрезков. Это можно сделать, если использовать технику *выбора с замещением*. Выбор с замещением основан не идее пирамидальной сортировки. При этом подходе *m* имен, которые умещаются в памяти, хранятся в виде такой пирамиды, что имена в сыновьях вершины больше имени в самой вершине (вместо того чтобы быть меньше, как это имеет место в пирамидальной сортировке).

Выбор с замещением порождает исходные отрезки следующим образом. Из входного файла считываются первые *m* имен, затем из них формируется пирамида (в соответствии с указанным выше принципом организации), в которой корнем является наименьшее имя. Наименьшее имя выводится как первое в первом отрезке и заменяется в пирамиде следующим именем из входного файла. Поскольку в результате такой замены может нарушиться свойство пирамидальности, далее производится восстановление пирамиды в соответствии с алгоритмом 5.7 (процедура *RESTORE*), модифицированным так, чтобы для восстановления пирамиды следить за наименьшим, а не за наибольшим именем. Процесс продолжается таким образом, что к текущему отрезку всегда добавляется наименьшее в пирамиде имя, большее или равное имени, которое последним добавлено к отрезку. Добавленное имя заменяется на следующее имя из входного файла, и восстанавливается пирамида. Когда в пирамиде нет имен, больших, чем последнее имя в текущем отрезке, отрезок обрывается и начинается новый отрезок. Этот процесс продолжается до тех пор, пока все имена не сформируются в отрезки.

При реализации процедуры выбора с замещением каждое имя x удобно рассматривать как пару (r, x), где r есть номер отрезка, в котором находится x. Таким образом, можно считать, что пирамида состоит из пар (r_1, x_1), (r_2, x_2), …, (r_m, x_m), причем сравнения между парами осуществляются лексикографически. Тогда, если на некотором шаге считывается имя, меньшее последнего имени в текущем отрезке, оно должно быть в следующем отрезке, и благодаря наличию номера отрезка это имя будет ниже всех имен пирамиды, которые входят в текущий отрезок, т. е. $(r_1, x_1) < (r_2, x_2)$, если $r_1 < r_2$ или если $r_1 = r_2$ и $x_1 < x_2$.

Очевидно, что выбор с замещением формирует исходные отрезки не хуже, чем простой метод, так как все отрезки (кроме, возможно, последнего) содержат не меньше m имен. В лучшем случае может быть сформирован единственный исходный отрезок, т. е. можно сразу получить отсортированную таблицу.

Исх. файл	Память (m = 3)	Раб. файл 1	Раб. файл 2	Комментарии
7, 3, 6, 5, 2, 1, 9, 4, 8				Исходная конфигурация
5, 2, 1, 9, 4, 8	(1,3), (1,7), (1,6)			Чтение m имен, построение пирамиды
5, 2, 1, 9, 4, 8	(1,3), (1,7), (1,6)	3		3 в раб. файл 1
2, 1, 9, 4, 8	(1,5), (1,7), (1,6)	3, 5		чтение 5, 5 > 3, поэтому r:=1, т.е. (1,5) замещает (1,3), восстановление пирамиды, 5 в раб. файл 1
1, 9, 4, 8	(1,6), (1,7), (2,2)	3, 5, 6		чтение 2, 2 < 5, поэтому r:=2, т.е. (2,2) замещает (1,5), восстановление пирамиды, 6 в раб. файл 1
9, 4, 8	(1,7), (2,1), (2,2)	3, 5, 6, 7		чтение 1, 1 < 6, поэтому r:=2, т.е. (2,1) замещает (1,6), восстановление пирамиды, 7 в раб. файл 1
4, 8	(1,9), (2,1), (2,2)	3, 5, 6, 7, 9		чтение 9, 9 > 6, поэтому r:=1, т.е. (1,9) замещает (1,7), восстановление пирамиды, 9 в раб. файл 1
8	(2,1), (2,4), (2,2)	3, 5, 6, 7, 9	1	чтение 4, 4 < 9, поэтому r:=2, т.е. (2,4) замещает (1,9), восстановление пирамиды, 1 в раб. файл 2
	(2,2), (2,4), (2,8)	3, 5, 6, 7, 9	1, 2	чтение 8, 8 > 1, поэтому r:=2, т.е. (2,8) замещает (2,1), восстановление пирамиды, 2 в раб. файл 2
	(2,4), (2,8)	3, 5, 6, 7, 9	1, 2, 4	4 в раб. файл 2
	(2,8)	3, 5, 6, 7, 9	1, 2, 4, 8	8 в раб. файл 2

5.9.2. Распределение и слияние отрезков

После формирования исходных отрезков возникает задача распределения их по рабочим файлам и слияния их до тех пор, пока не получится отсортированная таблица.

Простейший метод заключается в равномерном распределении отрезков по файлам 1, 2, …, t с последующим их слиянием в файл $t + 1$. Полученные в результате более длинные отрезки снова равномерно распределяются по файлам 1, 2, …, t и производится их слияние (образуются более длинные отрезки) в файл $t + 1$. Процесс продолжается до тех пор, пока в файле $t + 1$ не останется только один отрезок (отсортированная таблица). Очевидно, что каждый проход сокращает число отрезков в t раз (из t отрезков в результате слияния получается один отрезок). Следовательно, если имеется r исходных отрезков, то потребуется $\lceil \log_t r \rceil$ проходов, где каждый проход состоит из *фазы переписывания* (распределения отрезков по рабочим файлам), за которой следует *фаза слияния*. Таким образом, имеется всего $2\lceil \log_t r \rceil \approx (2/\log t)\lceil \log r \rceil$ проходов по именам, половина которых не уменьшает числа отрезков.

Более эффективные методы распределения и слияния отрезков основаны на исключении фазы переписывания, которая не сокращает числа отрезков. Одним из таких методов является *многофазное слияние*. Идея многофазного слияния состоит в том, чтобы организовать исходные отрезки так, что после каждого слияния, кроме последнего, остается ровно один пустой файл, который будет принимающим при следующем слиянии. В последнем слиянии должно участвовать только по одному отрезку из каждого из t непустых файлов. Распределение исходных отрезков с такими свойствами называется *совершенным распределением*. Пример многофазного слияния для 57 исходных отрезков, распределенных в соответствии с совершенным распределением по четырем файлам, показан на рис. 5.16. На этом рисунке запись вида $n * i$ означает n отрезков порядка i (отрезок порядка i есть результат слияния i исходных отрезков).

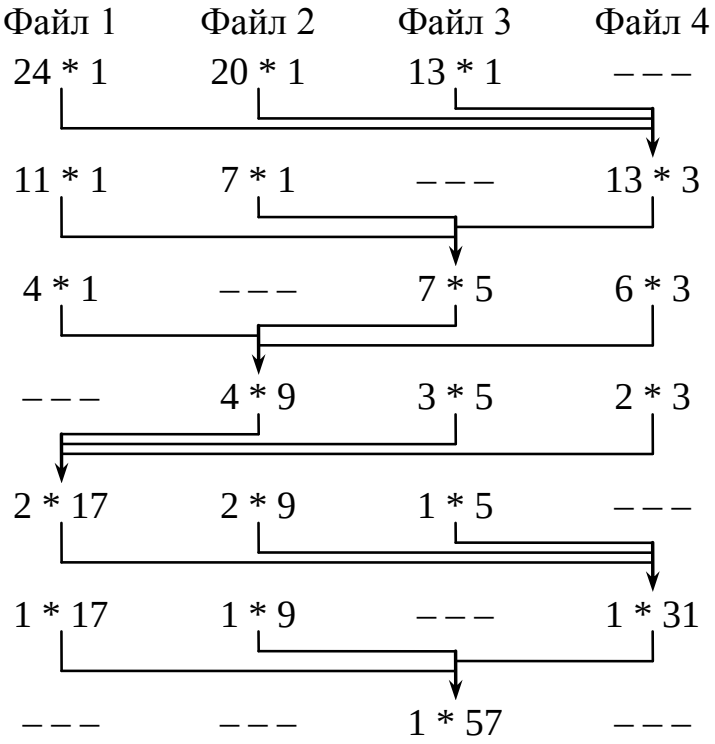

Рис. 5.16. Многофазное слияние 57 исходных отрезков

Можно легко определить формулу совершенного распределения исходных отрезков для общего случая, двигаясь в обратном направлении. Пусть $a_{k,j}$ – число отрезков в j-м файле, когда остается осуществить k фаз слияния. Будем считать, что файл $t + 1$ всегда является принимающим и что файл 1 содержит не меньше отрезков, чем файл 2, который содержит не меньше отрезков, чем файл 3, и т. д. Это можно сделать, используя логические номера файлов и переключая их соответствующим образом. Тогда в самом конце, когда больше проходов со слиянием не остается, имеем распределение

$$\begin{array}{rccccccc} j: & 1 & 2 & 3 & \ldots & t-1 & t & t+1 \\ a_{0,j}: & 1 & 0 & 0 & \ldots & 0 & 0 & 0 \end{array}$$

Когда остается k слияний, имеем распределение

$$\begin{array}{rccccccc} j: & 1 & 2 & 3 & \ldots & t-1 & t & t+1 \\ a_{k,j}: & a_{k,1} & a_{k,2} & a_{k,3} & \ldots & a_{k,t-1} & a_{k,t} & 0 \end{array}$$

Тогда следующее слияние приведет к распределению

$$\begin{array}{rccccccc} j: & 1 & 2 & 3 & \ldots & t-1 & t & t+1 \\ a_{k-1,j}: & a_{k,1}-a_{k,t} & a_{k,2}-a_{k,t} & a_{k,3}-a_{k,t} & \ldots & a_{k,t-1}-a_{k,t} & 0 & a_{k,t} \end{array}$$

Очевидно, что для совершенного распределения должно выполняться условие $2a_{k,t} \geq a_{k,1}$ при $k \geq 1$. Тогда переключение логических номеров файлов, при котором числа отрезков расположены в убывающем порядке, дает распределение

j:	1	2	3	...	$t-1$	t	$t+1$
$a_{k-1,j}$:	$a_{k,t}$	$a_{k,1}-a_{k,t}$	$a_{k,2}-a_{k,t}$	...	$a_{k,t-2}-a_{k,t}$	$a_{k,t-1}-a_{k,t}$	0

Поэтому

$$a_{k-1,1} = a_{k,t},$$
$$a_{k-1,j} = a_{k,j-1} - a_{k,t}, 2 \leq j \leq t,$$

или

$$a_{k+1,t} = a_{k,1},$$
$$a_{k+1,j} = a_{k,j+1} + a_{k,1}, 1 \leq j < t.$$

Следовательно, совершенные распределения для $t+1$ файлов можно представить следующим образом:

Файлы :	1	2	3	$\cdots$	$t-2$	$t-1$	t	$t+1$
$k=0$	1	0	0	$\cdots$	0	0	0	0
$k=1$	1	1	1	$\cdots$	1	1	1	0
$k=2$	2	2	2	$\cdots$	2	2	1	0
$k=3$	4	4	4	$\cdots$	4	3	2	0
$\vdots$				$\cdots$				
k	$a_{k,1}$	$a_{k,2}$	$a_{k,3}$	$\cdots$	$a_{k,t-2}$	$a_{k,t-1}$	$a_{k,t}$	0
$k+1$	$a_{k,2}+a_{k,1}$	$a_{k,3}+a_{k,1}$	$a_{k,4}+a_{k,1}$	$\cdots$	$a_{k,t-1}+a_{k,1}$	$a_{k,t}+a_{k,1}$	$a_{k,1}$	0

Очевидно, что число исходных отрезков не всегда удовлетворяет совершенному распределению. Выходом из такой ситуации является добавление фиктивных отрезков для получения требуемого распределения. Фиктивные отрезки прослеживаются алгоритмом многофазного слияния с помощью специальных счетчиков (обычно для каждого файла организуется свой счетчик), но не записываются физически в файлы. При этом фиктивные отрезки рекомендуется распределять как можно более равномерно по t файлам.

5.10. Порядковые статистики

Задачей вычисления *порядковых статистик* является *задача выбора k-го наименьшего имени* в таблице $T = \{x_1, x_2, \ldots, x_n\}$, т. е. имени, которое будет находиться в k-й позиции, если расположить имена в таблице в возрастающем порядке. Такое имя называется *k-й порядковой статистикой*. Для удобства будем считать, что таблица состоит из различных имен. Можно выделить специальные случаи: $k = 1$ (нахождение минимума), $k = n$ (нахождение максимума) и, если n нечетно, $k = (n + 1) / 2$ (нахождение *медианы*).

Задача, очевидно, симметрична: отыскание $(n - k + 1)$-го наименьшего (k-го наибольшего) имени можно осуществить, используя алгоритм отыскания k-го наименьшего имени, но меняя местами действия, предпринимаемые при результатах сравнений имен (больше и меньше). Таким образом, отыскание наименьшего имени ($k = 1$) эквивалентно отысканию наибольшего имени ($k = n$); отыскание второго наименьшего имени ($k = 2$) эквивалентно отысканию второго наибольшего ($k = n - 1$) и т. д. Поэтому можно считать, что $k \le \lceil n/2 \rceil$.

Для решения задачи выбора можно использовать любой из методов полной сортировки имен с последующим тривиальным обращением к k-му наименьшему имени. Такой подход требует порядка n^2 или $n \log n$ сравнений имен независимо от значения k. При таком подходе выполняется гораздо больше работы, чем требуется, поэтому необходимо искать более эффективные методы решения задачи.

Другой подход к решению задачи выбора основан на использовании одного из алгоритмов сортировки, основанных на выборе, т. е. либо простая сортировка выбором, либо пирамидальная сортировка. Очевидно, что для этого алгоритм простой сортировки выбором (алгоритм 5.6) необходимо модифицировать так, чтобы осуществлялся выбор не *i*-го наибольшего, а *i*-го наименьшего имени. Для пирамидальной сортировки (алгоритм 5.8) имена должны храниться в виде такой пирамиды, что имена в сыновьях вершины больше имени в самой вершине (корнем должно быть наименьшее имя). В каждом случае процесс сортировки можно остановить после выполнения первых *k* шагов. Для простой сортировки выбором это означает использование

$$(n-1)+(n-2)+\cdots+(n-k)=kn-\frac{k(k+1)}{2}$$

сравнений имен, а для пирамидальной сортировки – использование $n + k \log n$ сравнений имен. Здесь также выполняется избыточная работа, так как полностью определяется порядок *k* наименьших имен, но объем этой работы при данном подходе зависит от *k*.

Более эффективный подход к решению задачи выбора k-го наименьшего имени основан на идее быстрой сортировки. Таблица разбивается на две подтаблицы (меньше x_1 и больше x_1), а затем подходящая подтаблица исследуется рекурсивно. Предположим, что x_1 попадает в позицию j, т. е. x_1 является j-м наименьшим именем. Если $k = j$, то процедура заканчивается; если $k < j$, то поиск k-го наименьшего имени продолжается среди $x_1, x_2, \ldots, x_{j-1}$ (в левой подтаблице); если $k > j$, продолжается поиск $(k - j)$-го наименьшего имени среди $x_{j+1}, \ldots, x_n$ (в правой подтаблице). Анализ эффективности такого алгоритма подобен анализу алгоритма быстрой сортировки. Пусть $C_{\text{ave}}(n, k)$ – среднее число сравнений, требующееся для отыскания k-го наименьшего имени. Тогда

$$C_{\text{ave}}(n,k) = n + 1 + \frac{1}{n}\sum_{j=1}^{k-1} C_{\text{ave}}(n-j, k-j) + \frac{1}{n}\sum_{j=k+1}^{n} C_{\text{ave}}(j-1, k) .$$

Тривиальная индукция по k дает $C_{ave}(n, k) = O(n)$, т. е. данный метод требует в среднем только $O(n)$ сравнений имен независимо от значения k. К сожалению, число сравнений имен в худшем случае пропорционально n^2, поэтому в ряде случаев он может быть чрезвычайно неэффективным. Как и для алгоритма быстрой сортировки, причиной неэффективности является то, что расщепляющее имя может оказаться слишком близким к одному из двух краев таблицы (вместо того, чтобы быть ближе к середине). Для улучшения алгоритма необходимо применять способы эффективного нахождения расщепляющего имени, которое близко к середине таблицы. Вполне приемлемые результаты дает использование медианы малой случайной выборки, но наиболее близка к середине таблицы медиана медиан, что делает алгоритм пригодным даже для малых значений n.