

第7章 外部排序





学习目标

掌握外部排序的一般过程，熟练掌握适合外存特点的归并排序的相关技术。



7.1 磁盘文件的归并排序

• 外部排序的概念

- 是指在排序的过程中，数据的主要部分存放在外存储器上，借助内存储器(作为工作单元)，来调整外存储器上数据的位置。

• 外部归并排序重点研究的问题

- 如何进行多路归并以减少文件的归并遍数；
- 如何巧妙地运用内存的缓冲区使I/O和CPU尽可能并行工作
- 根据外存的特点选择较好的产生初始归并段的方法。

• 分类：

- 磁盘和磁带归并排序
- 磁盘是随机存储设备
- 磁带是顺序存储设备

7.1 磁盘文件的归并排序

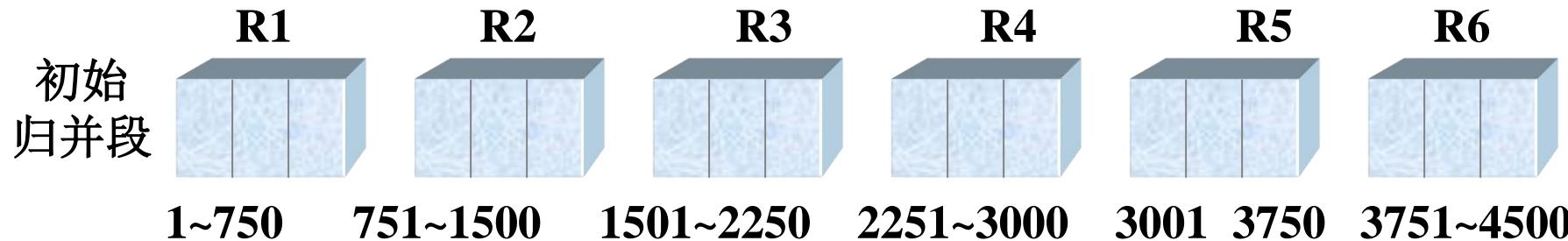
- 外部排序归并方法的一般过程：分两个阶段
 - 第一阶段：初始归并段形成
 - 首先，将文件中的数据分段输入到内存，在内存中采用内部排序方法对其进行排序（排序完的文件段，称为归并段 run），
 - 然后将有序段写回外存。
 - 整个文件经过在内存逐段排序又逐段写回外存，这样在外存中形成多个初始的归并段。
 - 第二阶段：多路归并
 - 对这些初始归并段采用某种归并排序方法，进行多遍归并，最后形成整个文件的单一归并段（整个文件有序）。

7.1 磁盘文件的归并分类

示例：设有一个含4500个记录的输入文件。现用一台其内存至多可容纳750个记录的计算机对该文件进行分类。输入文件放在磁盘上，磁盘每个页块可容纳250个记录，这样全部记录可存储在 $4500 / 250 = 18$ 个页块中。输出文件也放在磁盘上，用以存放归并结果。

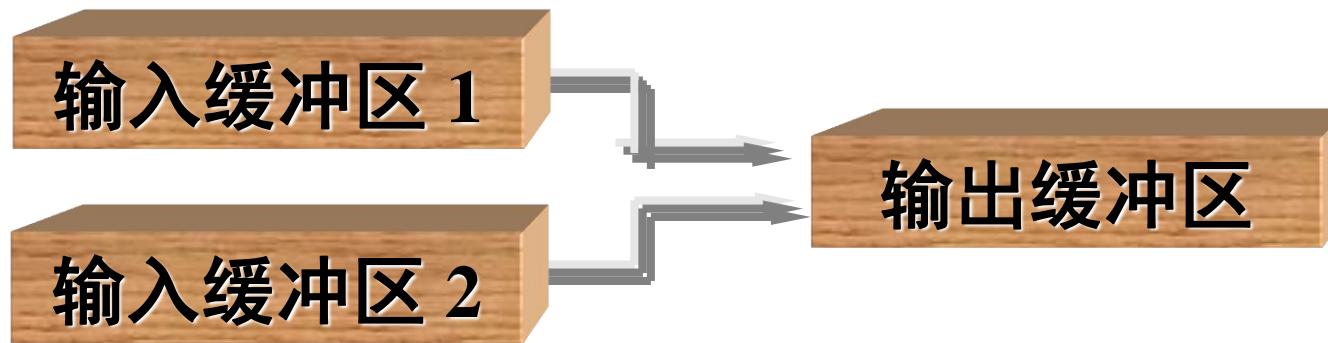
由于内存中可用于分类的存储区域能容纳750个记录，因此内存中恰好能存3个页块的记录。

在外分类一开始，把18块记录，每3块一组，读入内存。利用某种内分类方法进行内分类，形成初始归并段，再写回外存。总共可得到6个初始归并段。然后一趟一趟进行归并分类。



若把内存区域等份地分为 3 个缓冲区。其中的两个为输入缓冲区，一个为输出缓冲区，可以在内存中利用简单 2 路归并函数 `merge()` 实现 2 路归并。

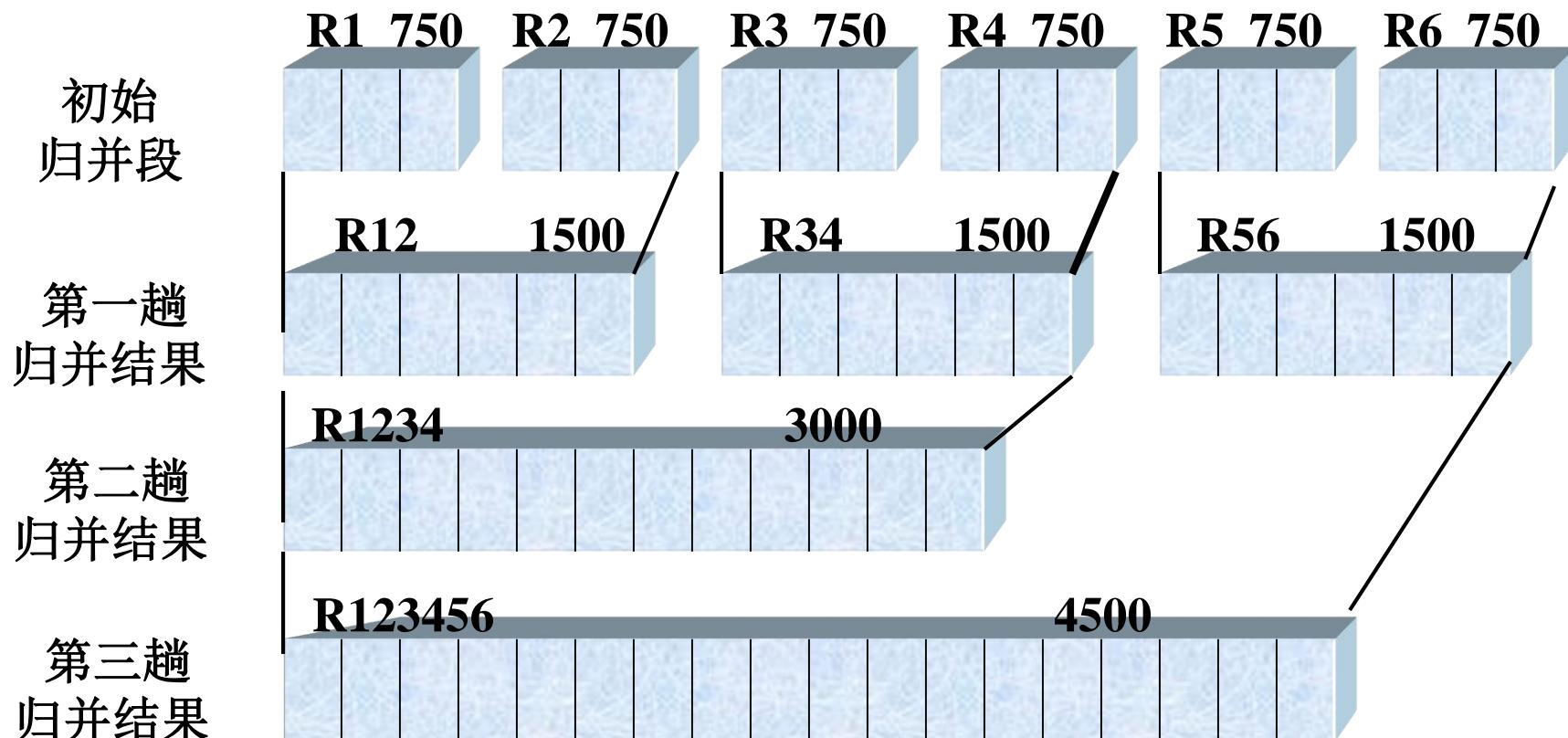
首先，从参加归并排序的两个输入归并段 R_1 和 R_2 中分别读入一块 250 个记录，放在输入缓冲区 1 和输入缓冲区 2 中。然后在内存中进行 2 路归并，归并结果顺序存放到输出缓冲区中。



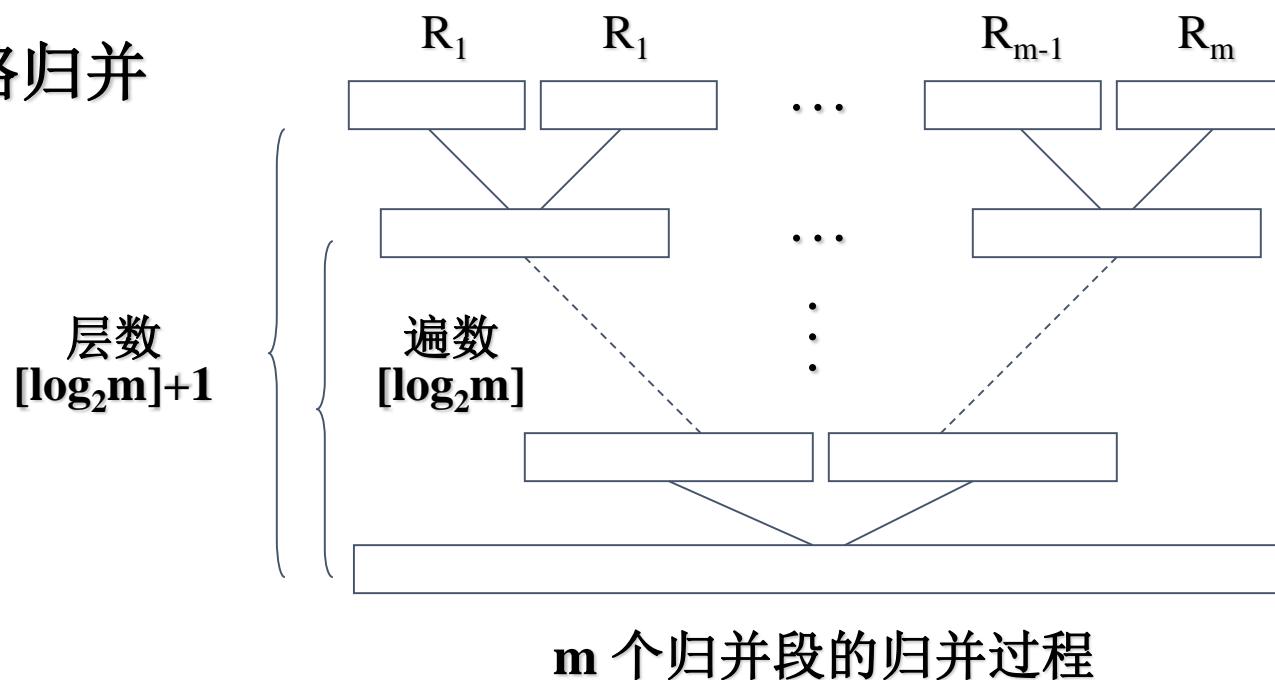
当输出缓冲区装满250个记录时，就输出到磁盘。

如果归并期间某个输入缓冲区空了，就立即向该缓冲区继续装入所对应归并段的一块记录信息，使之于另一个输入缓冲区的剩余记录归并，直到R1和R2归并为R12、R3和R4归并为R34、R5和R6归并为R56为止。

再把R12和R34归并为R1234，最后把R1234和R56归并为R123456



2路归并



讨论问题

- (1) 多路归并——减少归并遍数
- (2) 并行操作的缓冲区处理——使输入、输出和CPU处理尽可能重叠
- (3) 初始归并段的生成

7.1.1 多路归并——减少归并遍数

m 个初始段进行2路归并，需要 $\lceil \log_2 m \rceil$ 遍归并；

一般地， m 个初始段，采用 K 路归并，需要 $\lceil \log_K m \rceil$ 遍归并。

显然， K 越大，归并遍数越少，可提高归并的效率。

在 K 路归并时，从 K 个关键字中选择最小记录时，要比较 $K-1$ 次。若记录总数为 n ，每遍要比较的次数为：

$$(n-1) * (K-1) * [\log_K m] = (n-1) * (K-1) \lceil \log_2 m / \log_2 K \rceil$$

可以看出，随着 k 增大， $(K-1)/\log_2 K$ 也增大，当归并路数多时，CPU处理的时间也随之增多。**当 K 值增大到一定程度时，可能使CPU处理时间大于因 K 值增大而减少归并遍数所节省的时间。**

为此要选择好的分类方法，以减少分类中比较次数。

选择树 (Selection tree)

归并段或顺串 (run) ——————一个已排序的序列

假设有 k 个已经排序的序列，并且想要将其合并成一个单独的排序序列。每一个序列都由若干个记录组成并且按指定的域key（称为关键字）非降序排列。

合并工作可以通过重复地输出具有最小关键字值的记录来完成。

合并这 k 个归并段，最直接的方法是进行 $k-1$ 次比较，从而确定下一个要输出的记录。

对于 $k>2$ ，可以采用选择树的思想来减少寻找下一个最小元素所需要的比较次数。

一棵选择树 (selection tree) 是一棵二叉树，其中每一个结点都代表该结点两个儿子中的较小者。这样，树的根结点就表示树中最小元素的结点。

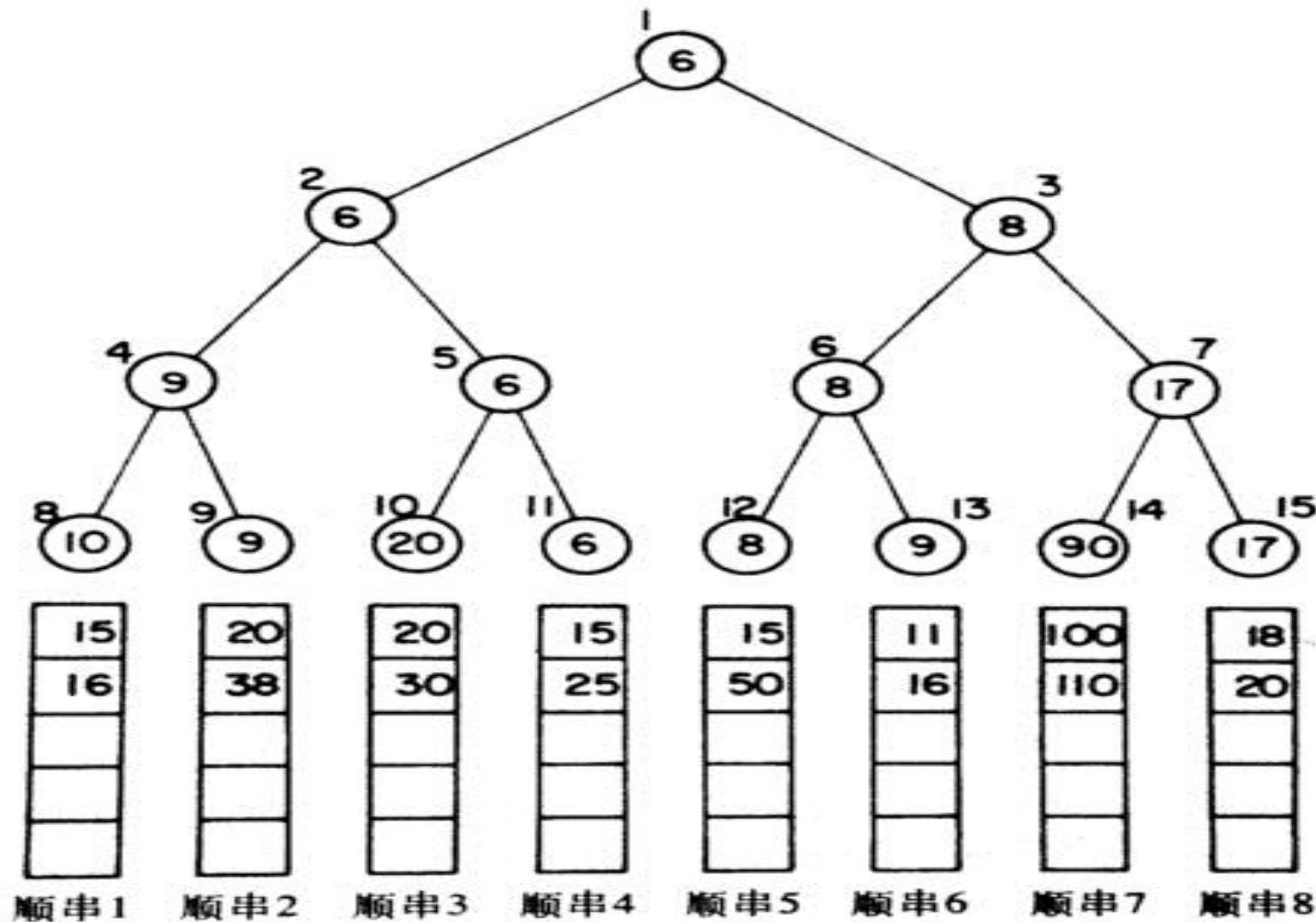
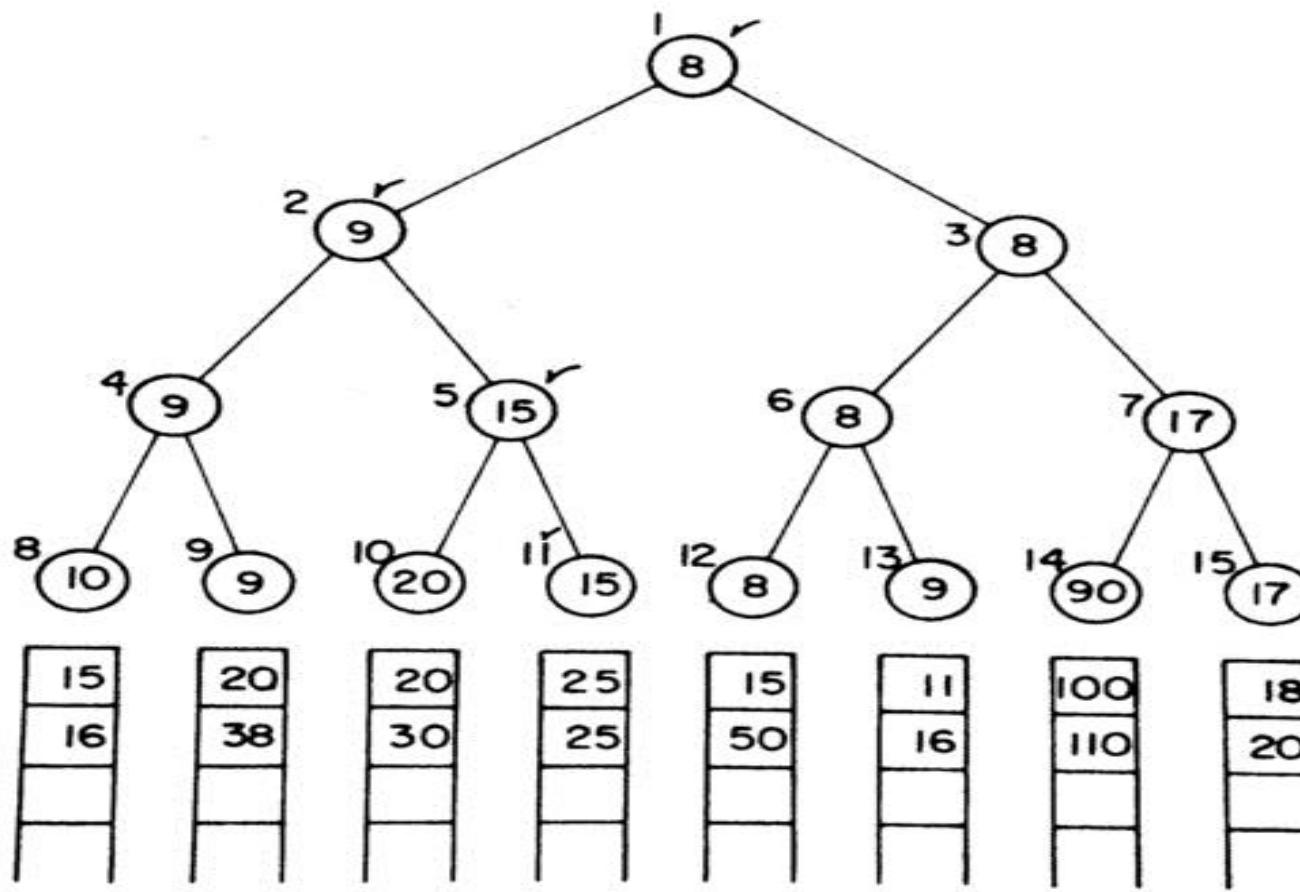
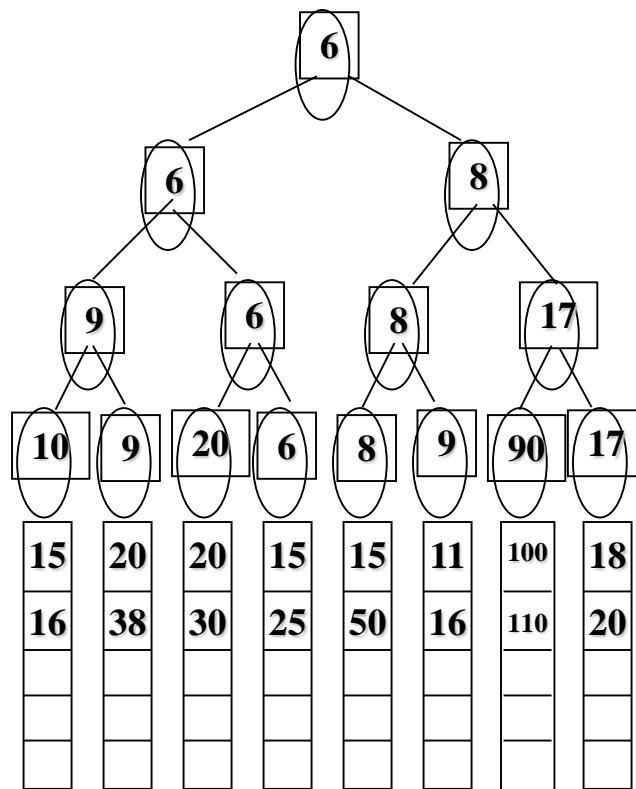


图7-16 $k=8$ 的选择树，并给出了每个归并段的前三个关键字



输出一个记录并重建的选择树（发生改变的结点用对勾作标记）

K 路平衡归并与选择树



选择树选最小元素过程

分析：

第一次建立选择树的比较所花时间为：

$$O(k-1) = O(k)$$

而后每次重新建造选择树所需时间为：

$$O(\log_2 k)$$

n 个记录处理时间为初始建立选择树的时间加上 $n-1$ 次重新选择树的时间：

$$O((n-1) \cdot \log_2 k) + O(k) = O(n \cdot \log_2 k)$$

这就是 k 路归并一遍所需的 CPU 处理时间。

归并遍数为 $\log_k m$, 总时间为：

$$O(n \cdot \log_2 k \cdot \log_k m) = O(n \cdot \log_2 m)$$

(k 路归并 CPU 时间与 k 无关)

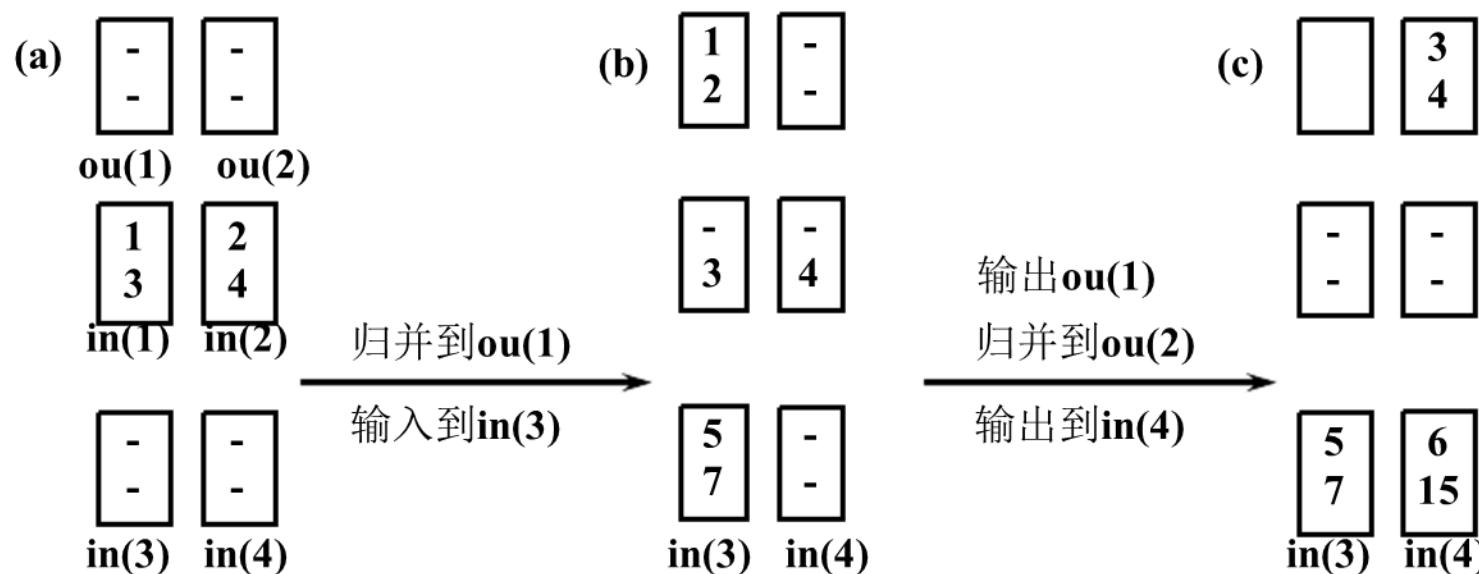
7.1.2 并行操作的缓冲区处理

——使输入、输出和 CPU 处理尽可能重叠

对 k 个归并段进行 k 路归并至少需要 k 个输入和 1 个输出缓冲区。

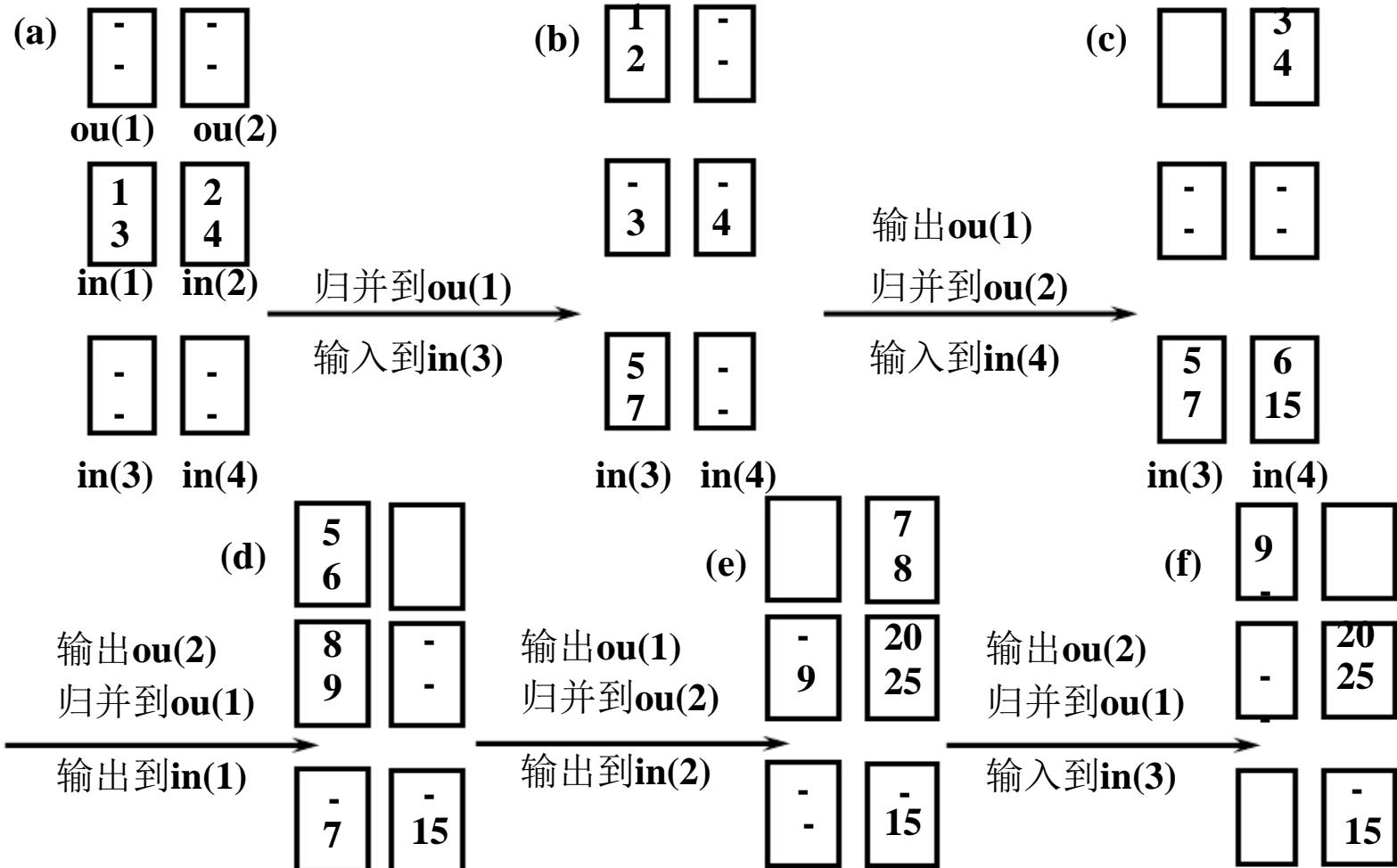
要使输入、输出和归并同时进行， $k+1$ 个缓冲区是不够的，需要 $2(k+1)$ 个缓冲区实现并行操作。

设有4个输入缓冲区，2个输出缓冲区，每个可容纳2个记录。归并段1数据为1, 3, 5, 7, 8, 9; 归并段2数据为2, 4, 6, 15, 20, 25。





并行操作的缓冲区处理——使I/O和 CPU 处理尽可能重叠



7.1.3 初始归并段的生成

- (a) 任何内部分类算法都可作为生成初始归并段的算法
- (b) 初始归并段的长度 \geq 缓冲区的长度?

使用选择树法

假设初始待排序文件为输入文件FI，初始归并段文件为输出文件FO，内存缓冲区为W，可容纳P个记录。FO,W初始为空，则置换-选择如下：

- (1) 从FI输入P个记录到缓冲区W;
- (2) 从W中选择出关键字最小的记录MIN；
- (3) 将MIN记录输出到FO中去；
- (4) 若FI不空，则从FI输入下一个记录到W;
- (5) 从W中所有关键字比MIN关键字大的记录中选出最小关键字记录，作为新的MIN;
- (6) 重复(3)~(5)，直到在W中选不出新的MIN为止。得到一个初始归并段，输出归并段结束标志到FO中
- (7) 重复(2)~(6)，直到W为空，由此得到全部初始归并段。

例如：缓冲区的长度为4，输入序列为：

新输入记录. key 小于当前记录. key, 等待下一个归并段

15 19 04 83 12 27 11 25 16 34 26 07 10 90 06 ...

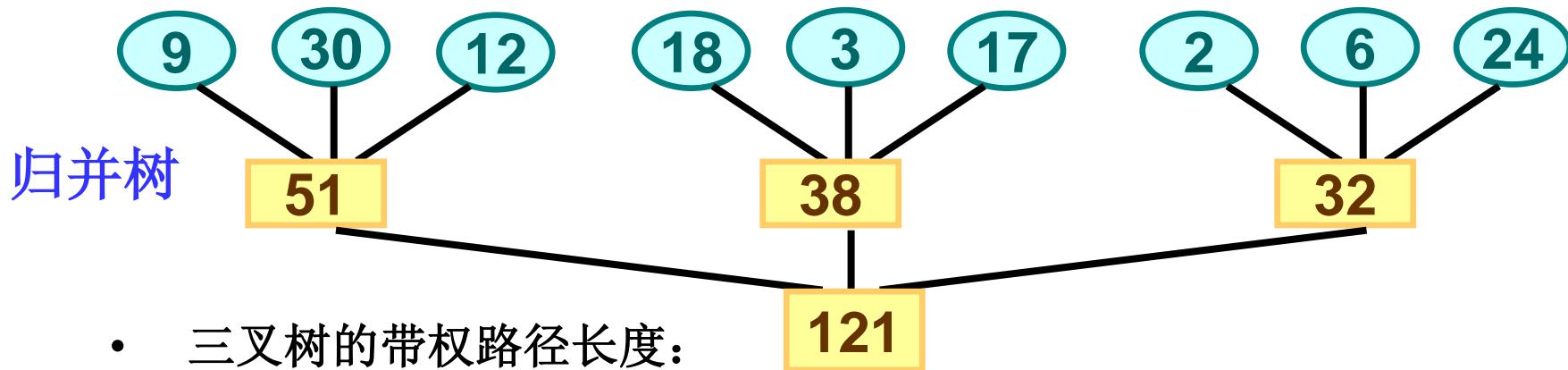
采用选择树法生成初始归并段的平均长度是缓冲区长度的两倍。

步	1	2	3	4	5	6	7	8	9	10	11	12	13	...
缓冲区内容	15	15	15	(11)	(11)	(11)	(11)	(11)	(11)	11	11	(06)
	19	19	19	19	25	(16)	(16)	(16)	(16)	16	16	16
	04	12	27	27	27	27	34	(26)	(26)	26	26	26
	83	83	83	83	83	83	83	83	(07)	10	90	90
输出结果	$\underbrace{04 \quad 12 \quad 15 \quad 19 \quad 25 \quad 27 \quad 34 \quad 83}_{R_1}$								$\underbrace{07 \quad 10 \quad 11 \quad 16 \quad \dots}_{R_2}$					

- 最佳归并树—使外存读写次数最少
 - 由置换-选择排序所得初始归并段的长度可能不等，这对于多路平衡归并将产生什么影响？
 - 例：假设经置换-选择排序先后得到的归并段长度分别为：

9, 30, 12, 18, 14, 25, 31, 7, 27

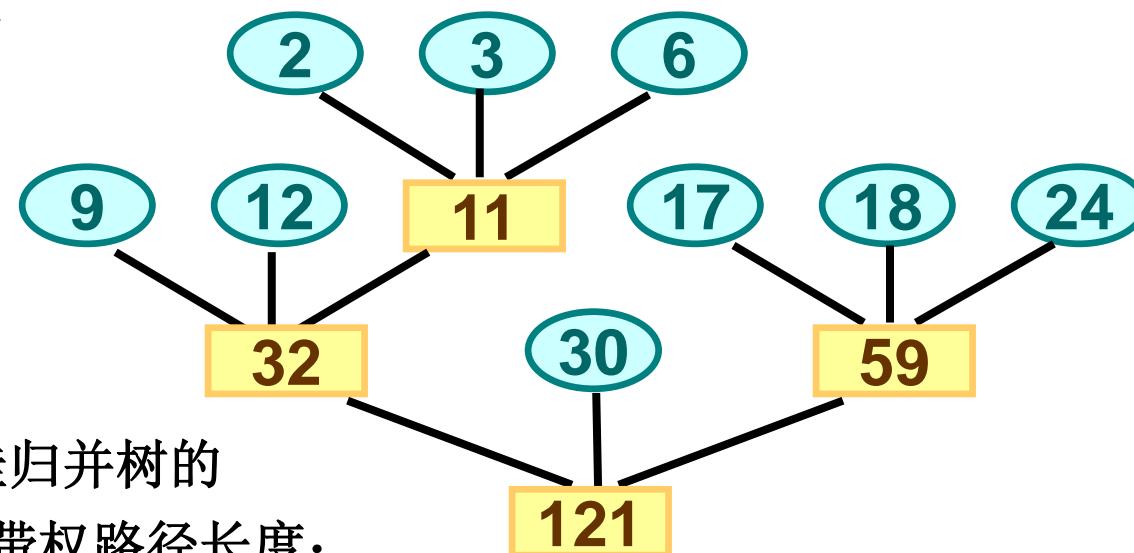
且每个记录占一个物理块，则进行3-路平衡归并排序时访问外存的次数是多少？



- 三叉树的带权路径长度：
$$WPL = (9+30+12+18+3+17+2+6+24) \times 2 = 242$$
 是否最佳？
- 访问外存次数： $242 \times 2 = 484$ (读/写各1次)



最佳归并树



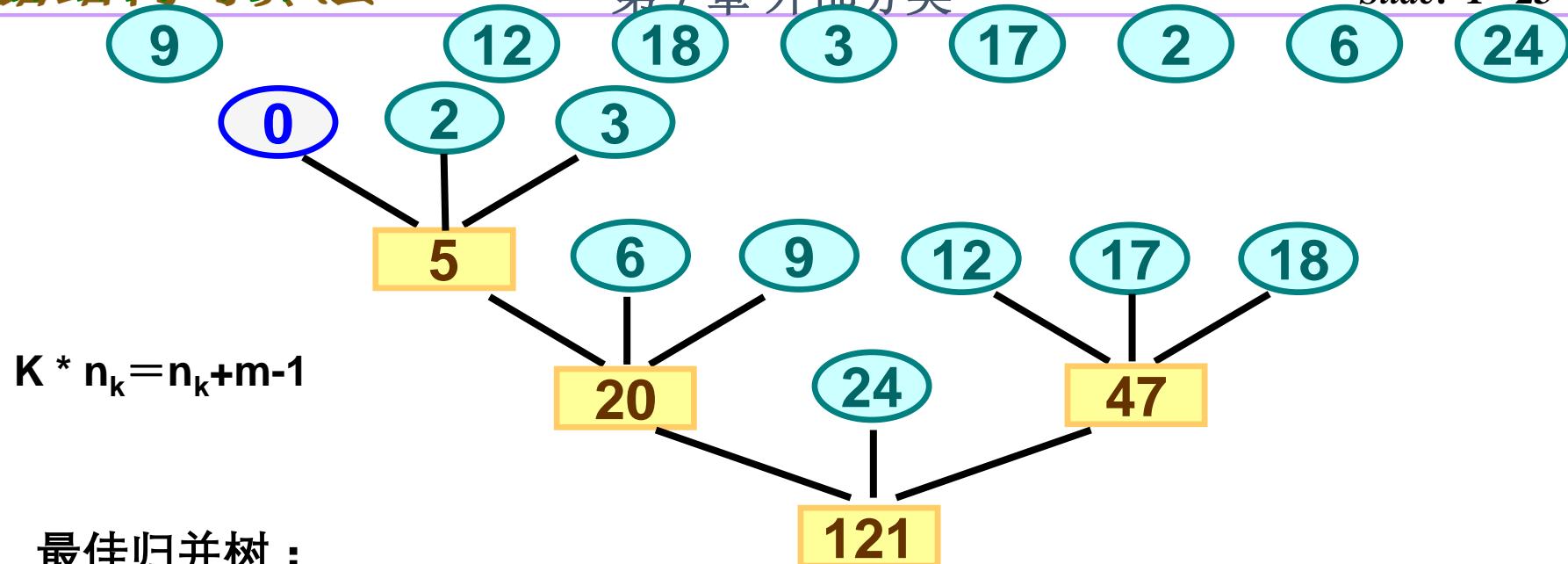
- 最佳归并树的
 - 带权路径长度:
$$WPL = (2+3+6) \times 3 + (9+12+17+18+24) \times 2 + 30 \times 1 = 223$$
 - 访问外存次数: $223 \times 2 = 446$ (读/写各1次)
- 如果去掉一个长度为30的初始归并段, 求最佳归并树?
 - $WPL = (2+3+5) \times 3 + (21+59) \times 2 = 193$
 - 访问外存次数: $193 \times 2 = 386$

是否有更少的?

Huffman树是一棵正则m叉树。若只有8个初始归并段，设上例中少了一个长度为30的归并段。如果在设计归并方案时，缺额的归并段留着最后，即除了最后一次作2-路归并外，其他各次归并仍都是3-路归并，此归并方案的外存读写次数为386。

显然不是最佳方案。

正确的做法是，若初始归并段不足构成一棵正则m叉树时，需添加长度为0的“虚段”，按照Huffman树的原则，权为0的叶子应离根最近。



◆ 最佳归并树：

- $WPL = (2+3) \times 3 + (15+47) \times 2 + 24 \times 1 = 163$

- 访问外存次数： $163 \times 2 = 326$

◆ 问题：当初始归并段的数目不足时，怎样求最佳归并树？

- 最佳归并树应该是一棵“正则树”

- 对 K 路归并而言，设初始归并段为 m，若：

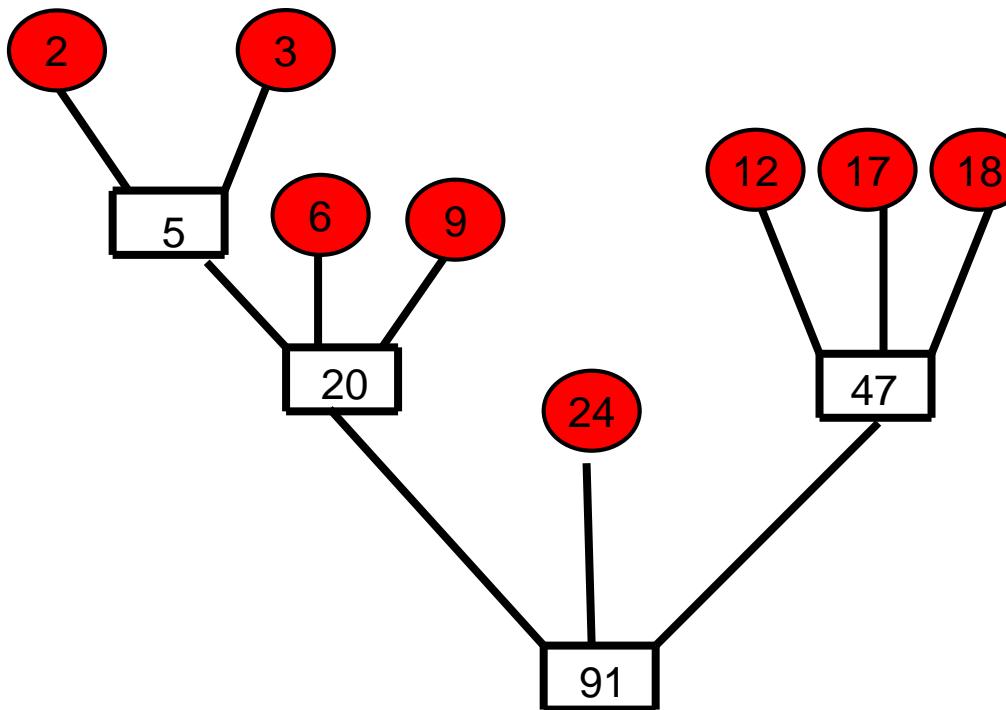
$$(m - 1) \% (K - 1) = 0$$

则不需要加虚段，否则需要加虚段的个数为：

$$K - (m - 1) \% (K - 1) - 1$$

- 按照 **HUFFMAN** 树的思想，记录少的段最先合并。不够时增加虚段。设归并段长度为 $(2, 3, 6, 9, 12, 17, 18, 24)$ ，最佳归并树如下例所示。

C.



从外存读 5 个记录

写入外存 5 个记录

从外存读 67 个记录

写入外存 67 个记录

从外存读 91 个记录

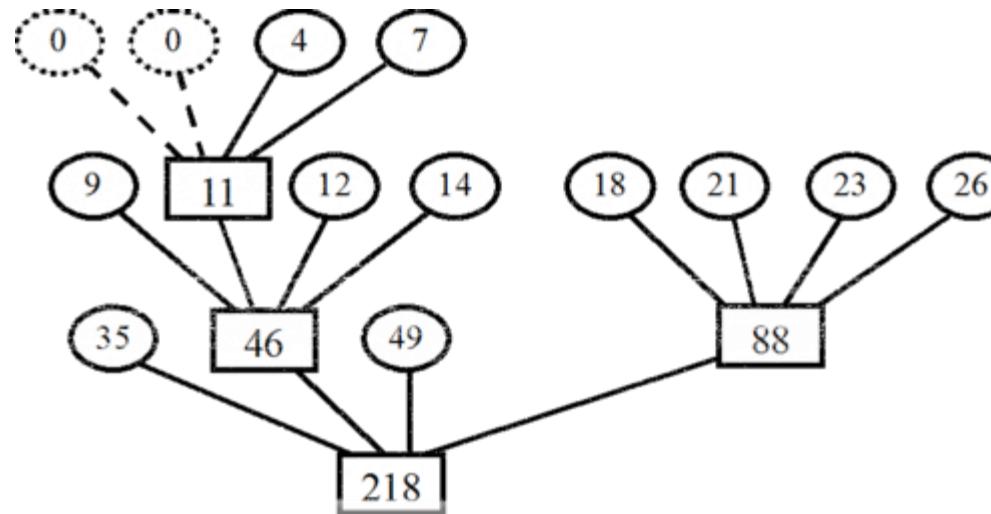
写入外存 91 个记录

总共读写外存 326 个记录

例 设文件经预处理后，得到长度为{47,9,39,18,4,12,23,7,21,16,26}的11个初始归并段，试为4路归并设计一个读写文件次数最少的归并方案。

初始归并段的个数n=11，归并路数k=4，由于 $(n-1)%(k-1)=1$ ，不为0，因此需附加长度为0的虚段个数： $(k-1)-(n-1)%(k-1)=2$ 。

根据集合：{49,9,35,18,4,12,23,7,21,14,26,0,0}构造4阶哈夫曼树，如下图所示（四路最佳归并树）。



若每个记录占用一个物理页块，则此方案对外存的读写次数为：

$$\begin{aligned} & 2 \times [(4+7) \times 3 + (9+12+14+18+21+23+26) \times 2 + (35+49) \times 1] \\ & = 726 \text{ 次。} \end{aligned}$$

所有叶结点加权外通路长度的2倍

本章小结

- 内部排序过程中不涉及数据的内、外存交换，待排序的记录全部存放在内存中。
- 若待排序的文件很大，就无法将整个文件的所有记录同时调入内存进行排序，则采用外排序算法。
- 外部排序的实现，主要是依靠数据的内、外存交换和“内部归并”。
- 外部排序基本上包括相对独立的两个阶段：初始归并段的形成和多路归并。
- 外部排序主要研究的技术问题是：
 - 如何进行多路归并以减少文件的归并遍数；
 - 如何运用内存的缓冲区使I/O和CPU尽可能并行工作；
 - 根据外存的特点选择较好的产生初始归并段的方法。

本章小结

- 初始归并段的形成：
 - 目标：尽量增加初始归并段的长度，从而减少个数
 - 任何内部排序算法都可作为生成初始归并段的算法
 - 初始归并段的长度 \geq 缓冲区的长度?
采用选择树法(置换-选择排序)生成初始归并段的平均长度是缓冲区长度的两倍
- I/O和CPU尽可能并行工作
 - 进行 K 路归并至少需要 K 个输入和 1 个输出缓冲区。
 - 要使输入、输出和归并同时进行， $K+1$ 个缓冲区是不够的
 - 缓冲区备份法：需要 $2(K+1)$ 个缓冲区实现并行操作

本章小结

- 多路归并: (假设有 n 个记录, m 个初始段)
 - 采用2路归并, 需要 $\lceil \log_2 m \rceil$ 趟归并;
 - 采用 K 路归并, 需要 $\lceil \log_K m \rceil$ 趟归并。
 - 显然, K 越大, 归并趟数越少, 可提高归并的效率
 - 但可能增加整个归并过程的比较次数:
 - $n * (K-1) \lceil \log_K m \rceil = n * (K-1) \lceil \log_2 m / \log_2 K \rceil$
- 多路平衡归并---选择树法
 - 建立选择树的时间: $O(K-1) = O(K)$
 - 重构选择树的时间: $O(\log_2 K)$
 - 一趟归并的时间: $O((n-1) \cdot \log_2 K) + O(K) = O(n \cdot \log_2 K)$
 - 总的归并时间: $O(n \log_2 K \log_K m) = O(n \log_2 m)$

本章小结

- 最佳归并树:

- 由置换-选择排序所得初始归并段的长度可能不等
- 目标：最小化多路平衡归并的外存读写次数
- 当初始归并段的数目不足时，怎样进行最佳归并？
 - 最佳归并树应该是一棵“正则树”
 - 对 K 路归并而言，设初始归并段为 m，
 - 若 $(m-1) \% (K-1) = 0$ ，则不需要加虚段；
 - 否则，需要加虚段的个数为：
 - $K - (m-1) \% (K-1) - 1$

何谓海量数据处理？

所谓海量数据处理，无非就是基于海量数据上的存储、处理、操作。何谓海量，就是数据量太大，所以导致要么是无法在较短时间内迅速解决，要么是数据太大，导致无法一次性装入内存。

那解决办法呢？针对时间，可以采用巧妙的算法搭配合适的数据结构，如Bloom filter/Hash/bit-map/堆/数据库或倒排索引/trie树；

针对空间，无非就一个办法：大而化小，分而治之（hash映射），规模太大就把规模大化为规模小的，各个击破。

处理海量数据问题，无非就是：

分而治之/hash映射 + hash统计 + 堆/快速/归并排序；

双层桶划分

Bloom filter/Bitmap；

Trie树/数据库/倒排索引；

外排序；

分布式处理之Hadoop/Mapreduce。

什么是Bloom Filter

Bloom Filter是一种空间效率很高的随机数据结构，它的原理是，当一个元素被加入集合时，通过K个Hash函数将这个元素映射成一个位阵列（Bit array）中的K个点，把它们置为1。检索时，我们只要看看这些点是不是都是1就（大约）知道集合中有没有它了：如果这些点有任何一个0，则被检索元素一定不在；如果都是1，则被检索元素很可能在。这就是布隆过滤器的基本思想。

但Bloom Filter的这种高效是具有一定代价的：在判断一个元素是否属于某个集合时，有可能会把不属于这个集合的元素误认为属于这个集合（false positive）。因此，Bloom Filter不适合那些“零错误”的应用场合。而在能容忍低错误率的应用场合下，Bloom Filter通过极少的错误换取了存储空间的极大节省。

1、海量日志数据，提取出某日访问百度次数最多的那个IP。

算法思想：分而治之+Hash

1. IP地址最多有 $2^{32}=4G$ 种取值情况，所以不能完全加载到内存中处理；
2. 可以考虑采用“分而治之”的思想，按照IP地址的 $\text{Hash(IP)} \% 1024$ 值，把海量IP日志分别存储到1024个小文件中。这样，每个小文件最多包含4MB个IP地址；
3. 对于每一个小文件，可以构建一个IP为key，出现次数为value的Hash map，同时记录当前出现次数最多的那个IP地址；
4. 可以得到1024个小文件中的出现次数最多的IP，再依据常规的排序算法得到总体上出现次数最多的IP；

2、有一个1G大小的一个文件，里面每一行是一个词，词的大小不超过16字节，内存限制大小是1M。返回频数最高的100个词。

1、分而治之/hash映射：顺序读文件中，对于每个词x，取 $\text{hash}(x) \% 5000$ ，然后按照该值存到5000个小文件（记为 $x_0, x_1, \dots, x_{4999}$ ）中。这样每个文件大概是200k左右。

如果其中的有的文件超过了1M大小，还可以按照类似的方法继续往下分，直到分解得到的小文件的大小都不超过1M。

2、hash_map统计：对每个小文件，统计每个文件中出现的词以及相应的频率（可以采用trie树/hash_map等）

3、堆/归并排序：取出出现频率最大的100个词（可以用含100个结点的最小堆），并把100个词及相应的频率存入文件，这样又得到了5000个文件。下一步就是把这5000个文件进行归并（类似与归并排序）的过程了。

3、在2.5亿个整数中找出不重复的整数，注，内存不足以容纳这2.5亿个整数。

第一种方法：采用2-Bitmap（每个数分配2bit，00表示不存在，01表示出现一次，10表示多次，11无意义）进行，共需内存 $2^{32} * 2 \text{ bit} = 1 \text{ GB}$ 内存，还可以接受。然后扫描这2.5亿个整数，查看Bitmap中相对应位，如果是00变01，01变10，10保持不变。扫描完事后，查看bitmap，把对应位是01的整数输出即可。

第二种方法：也可采用与第1题类似的方法，进行划分小文件的方法。然后在小文件中找出不重复的整数，并排序。然后再进行归并，注意去除重复的元素。

4、100w个数中找出最大的100个数。

方案1：在前面的题中，我们已经提到了，用一个含100个元素的最小堆完成。复杂度为 $O(100w * \lg 100)$ 。

方案2：采用快速排序的思想，每次分割之后只考虑比轴大的一部分，知道比轴大的一部分在比100多的时候，采用传统排序算法排序，取前100个。复杂度为 $O(100w * 100)$ 。

方案3：采用局部淘汰法。选取前100个元素，并排序，记为序列L。然后一次扫描剩余的元素x，与排好序的100个元素中最小的元素比，如果比这个最小的要大，那么把这个最小的元素删除，并把x利用插入排序的思想，插入到序列L中。依次循环，知道扫描了所有的元素。复杂度为 $O(100w * 100)$ 。

5、给40亿个不重复的unsigned int的整数，没排过序的，然后再给一个数，如何快速判断这个数是否在那40亿个数当中？

与第3题类似，第一反应时快速排序+二分查找。以下是其它更好的方法：

方案1：申请512M的内存，一个bit位代表一个unsigned int值。读入40亿个数，设置相应的bit位，读入要查询的数，查看相应bit位是否为1，为1表示存在，为0表示不存在。

方案2：这个问题在《编程珠玑》里有很好的描述，大家可以参考下面的思路，探讨一下：

又因为 2^{32} 为40亿多，所以给定一个数可能在，也可能不在其中；

这里我们把40亿个数中的每一个用32位的二进制来表示

假设这40亿个数开始放在一个文件中。

然后将这40亿个数分成两类：

- 1.最高位为0
- 2.最高位为1

并将这两类分别写入到两个文件中，其中一个文件中数的个数 ≤ 20 亿，而另一个 ≥ 20 亿（这相当于折半了）；

与要查找的数的最高位比较并接着进入相应的文件再查找

再然后把这个文件又分成两类：

- 1.次最高位为0
- 2.次最高位为1

并将这两类分别写入到两个文件中，其中一个文件中数的个数 ≤ 10 亿，而另一个 ≥ 10 亿（这相当于折半了）；

与要查找的数的次最高位比较并接着进入相应的文件再查找。