第十章

- 外排序

外排序

当待排序的对象数目特别多时，在内存中不能一次处理。必须把它们以文件的形式存放于外存，排序时再把它们一部分一部分调入内存进行处理。这样，在排序过程中必须不断地在内存与外存之间传送数据。这种基于外部存储设备（或文件）的排序技术就是外排序。

外排序的基本过程

- 当对象以文件形式存放于磁盘上的时候，通常是按物理块存储的。

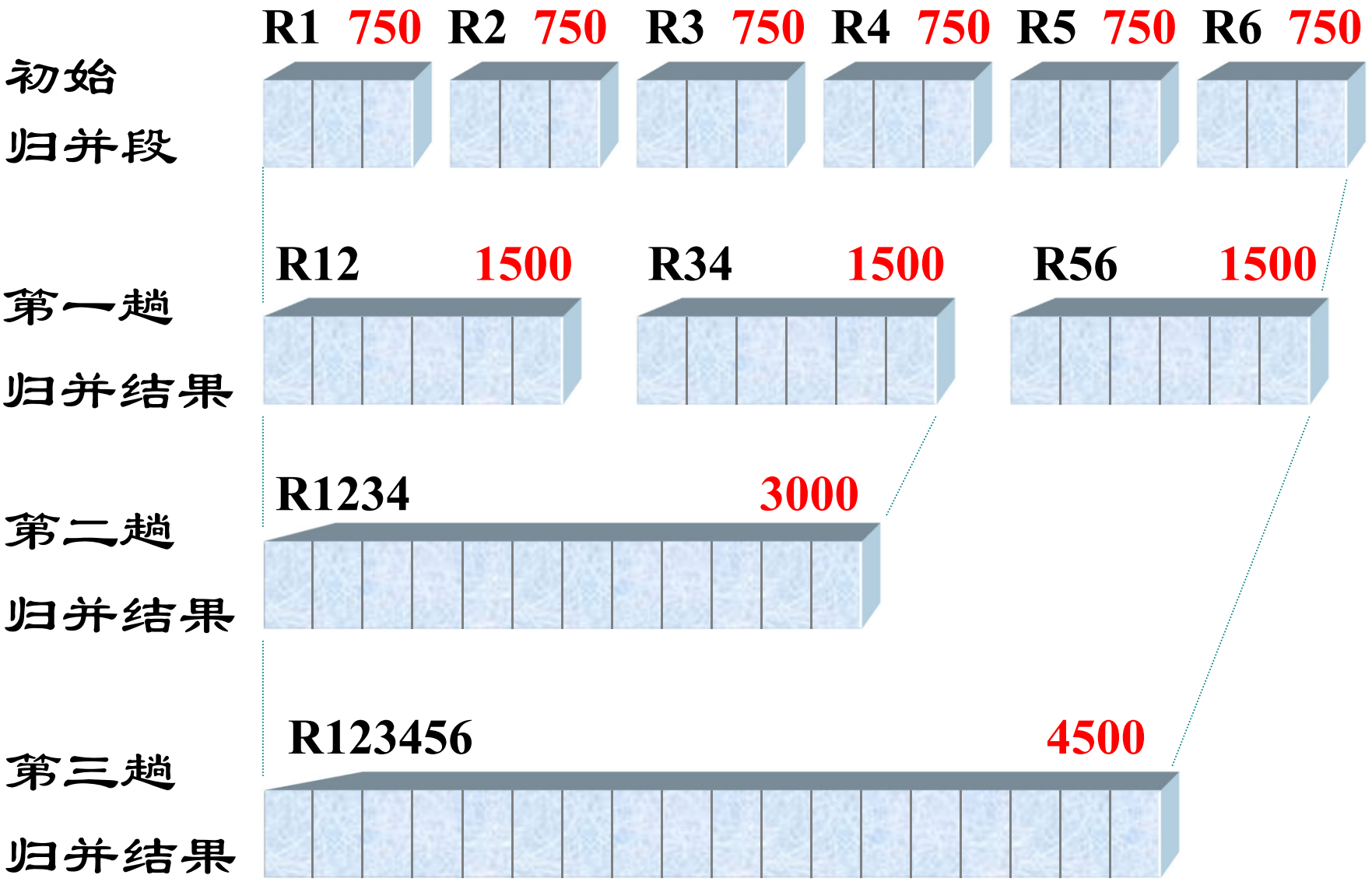
- 物理块也叫做页块。每个页块可存放几个对象。操作系统按页块对磁盘信息进行读写。
- 磁盘通常是指由若干片磁盘组成的磁盘组, 各个盘片安装在同一主轴上高速旋转。各个盘面上半径相同的磁道构成了柱面。各盘面设置一个读写磁头，它们装在同一动臂上，可以径向从一个柱面移到另一个柱面上。
- 为了访问某一页块, 先寻找柱面: 移动动臂使读写磁头移到指定柱面上: 寻查(seek)。再根据磁道号(盘面号)选择相应读写磁头, 等待指定页块转到读写磁头下: 等待(latency)。 在磁盘组上存取一个页块的时间:

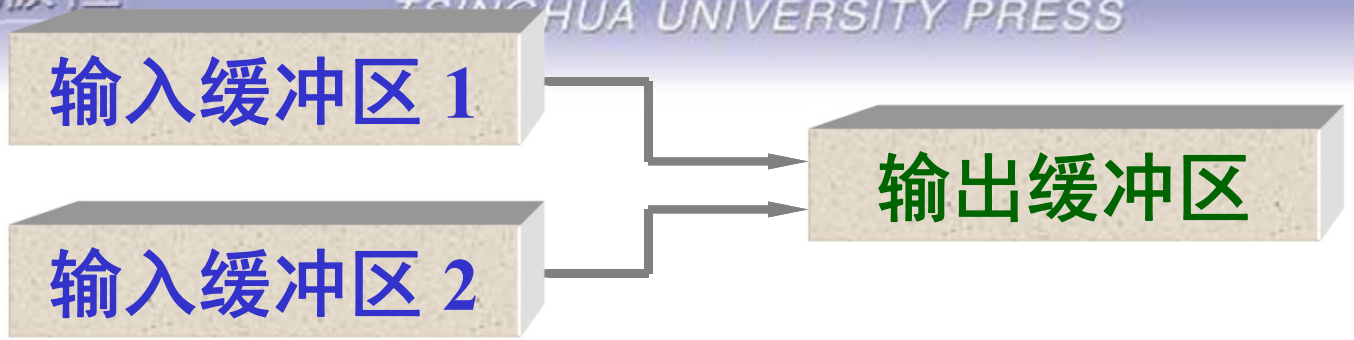
$$t_{io} = t_{seek} + t_{latency} + t_{rw}$$

- 基于磁盘进行的排序多使用归并排序方法。其排序过程主要分为两个阶段：
  ① 建立用于外排序的内存缓冲区。根据它们的大小将输入文件划分为若干段，用某种内排序方法对各段进行排序。经过排序的段叫做初始归并段或初始顺串 (Run)。当它们生成后就被写到外存中去。
  ② 仿照归并树模式，把 ① 生成的初始归并段加以归并，一趟趟地扩大归并段和减少归并段个数，直到最后归并成一个大归并段(有序文件)为止。

- 示例：设有一个包含4500个对象的输入文件。现用一台其内存至多可容纳750个对象的计算机对该文件进行排序。输入文件放在磁盘上,磁盘每个页块可容纳250个对象，这样全部对象可存储在 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。
- 由于内存中可用于排序的存储区域能容纳750 个对象, 因此内存中恰好能存3个页块的对象。
- 在外排序一开始, 把18块对象, 每3块一组, 读入内存。利用某种内排序方法进行内排序, 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排序。

两路归并排序的归并树


R1 750
R2 750
R3 750
R4 750
R5 750
R6 750
初始
归并段
R12 1500
R34 1500
R56 1500
第一趟
归并结果
R1234 3000
第二趟
归并结果
R123456 4500
第三趟
归并结果



- 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, 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。

- 若总对象个数为 n，磁盘上每个页块可容纳 b 个对象，内存缓冲区可容纳 i 个页块，则每个初始归并段长度为 $len = i * b$，可生成 $m = \lceil n / len \rceil$ 个等长的初始归并段。
- 在做 2 路归并排序时, 第一趟从 m 个初始归并段得到 $\lceil m/2 \rceil$ 个归并段，以后各趟将从 l (l >1) 个归并段得到 $\lceil l/2 \rceil$ 个归并段。总归并趟数等于归并树的高度 $\lceil \log_2 m \rceil$。
- 估计 2 路归并排序时间 t_{ES} 的上界为：

$$t_{ES} = m*t_{IS} + d*t_{IO} + S*u*t_{mg}$$

- 对 4500 个对象进行排序的例子, 各种操作的计算时间如下:
  - 读 18 个输入块, 内部排序6段, 写18个输出块 $= 6\,t_{IS} + 36\,t_{IO}$
  - 成对归并初始归并段 $R_1 \sim R_6$
    $= 36\,t_{IO} + 4500\,t_{mg}$
  - 归并两个具有 1500 个对象的归并段 R_{12} 和 R_{34}
    $= 24\,t_{IO} + 3000\,t_{mg}$
  - 最后将 R_{1234} 和 R_{56} 归并成一个归并段
    $= 36\,t_{IO} + 4500\,t_{mg}$
- 合计 $t_{ES} = 6\,t_{IS} + 132\,t_{IO} + 12000\,t_{mg}$

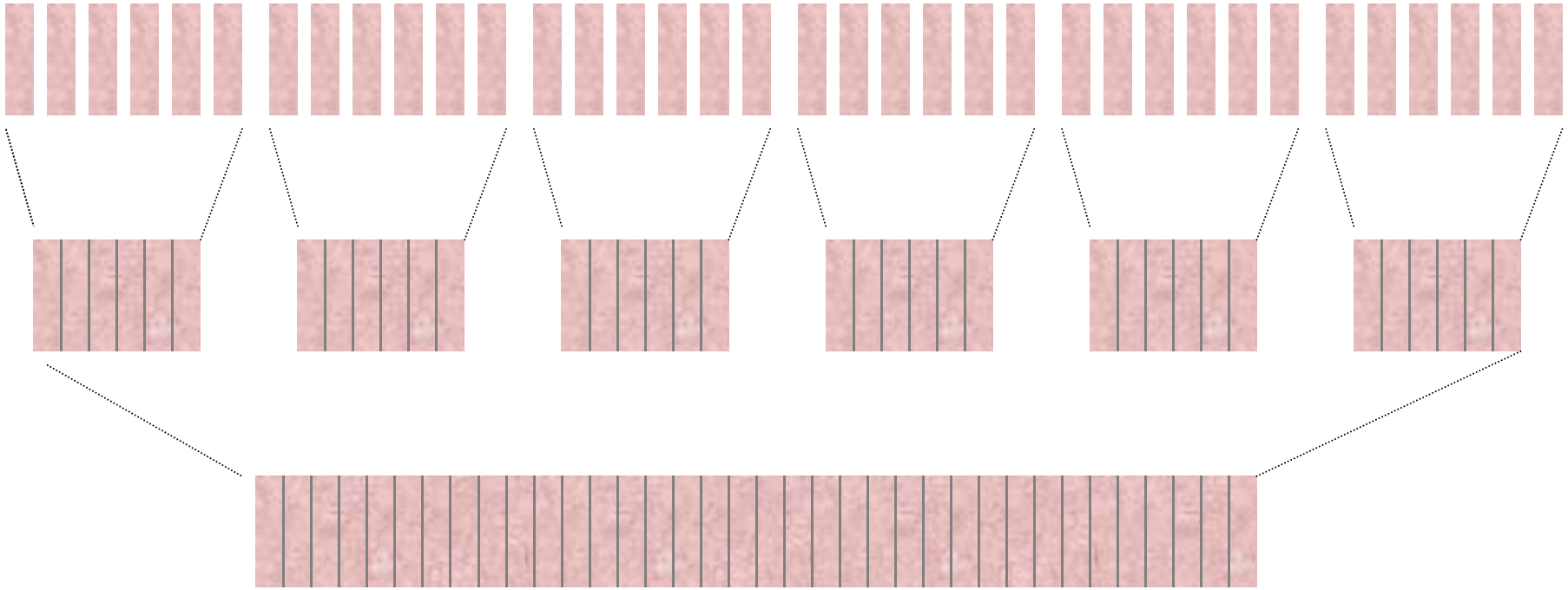
- 由于 $t_{IO}=t_{seek}+t_{latency}+t_{rw}$，其中，$t_{seek}$和$t_{latency}$是机械动作，而$t_{rw}$、$t_{IS}$、$t_{mg}$是电子线路的动作，所以$t_{IO}$远远大于$t_{IS}$和$t_{mg}$。想要提高外排序的速度，应着眼于减少 d。
- 若对相同数目的对象，在同样页块大小的情况下做 3 路归并或做 6 路归并(当然，内存缓冲区的数目也要变化)，则可做大致比较：

归并路数 k	总读写磁盘次数 d	归并趟数 S
2	132	3
3	108	2
6	72	1

- 增大归并路数, 可减少归并趟数, 从而减少总读写磁盘次数d。
- 对 m 个初始归并段, 做 k 路平衡归并, 归并树可用**正则 k 叉树** (即只有度为 k 与度为0的结点的 k 叉树) 来表示。
- 第一趟可将 m 个初始归并段归并为 $l=\lceil m/k \rceil$ 个归并段，以后每一趟归并将 l 个归并段归并成 $l=\lceil l/k \rceil$ 个归并段，直到最后形成一个大的归并段为止。树的高度 = $\lceil \log_k m \rceil$ = 归并趟数S。

k路平衡归并 (k-way Balanced merging)

- 做 k 路平衡归并时, 如果有 m 个初始归并段, 则相应的归并树有 $\lceil \log_k m \rceil + 1$ 层，需要归并 $\lceil \log_k m \rceil$ 趟。下图给出对有 36 个初始归并段的文件做 6 路平衡归并时的归并树。



- 做内部 k 路归并时，在 k 个对象中选择最小者，需要顺序比较 k-1 次。每趟归并 n 个对象需要做(n-1)*(k-1)次比较，S 趟归并总共需要的比较次数为：

$$S*(n-1)*(k-1) = \lceil \log_k m \rceil * (n-1) * (k-1)$$
$$= \lceil \log_2 m \rceil * (n-1) * (k-1) / \lceil \log_2 k \rceil$$

- 在初始归并段个数 m 与对象个数 n 一定时，$\lceil \log_2 m \rceil * (n-1) = \text{const}$，而 $(k-1) / \lceil \log_2 k \rceil$ 在 k 增大时趋于无穷大。因此，增大归并路数 k，会使得内部归并的时间增大。

- 使用“败者树”从 k 个归并段中选最小者，当 k 较大时 ($k \geq 6$)，选出排序码最小的对象只需比较$\lceil \log_2 k \rceil$次。

$$S*(n\text{-}1)*\lceil \log_2 k \rceil = \lceil \log_k m \rceil * (n\text{-}1) * \lceil \log_2 k \rceil$$
$$= \lceil \log_2 m \rceil * (n\text{-}1) * \lceil \log_2 k \rceil / \lceil \log_2 k \rceil$$
$$= \lceil \log_2 m \rceil * (n\text{-}1)$$

- 排序码比较次数与 k 无关，总的内部归并时间不会随 k 的增大而增大。
- 下面讨论利用败者树在 k 个输入归并段中选择最小者，实现归并排序的方法。

- 败者树是一棵正则的完全二叉树。其中
  - 每个叶结点存放各归并段在归并过程中当前参加比较的对象；
  - 每个非叶结点记忆它两个子女结点中对象排序码大的结点(即败者)；
- 因此，根结点中记忆树中当前对象排序码最小的结点 (最小对象)。
- 败者树与胜者树的区别在于一个选择了败者(排序码大者), 一个选择了胜者(排序码小者)。
- 示例：设有 5 个初始归并段, 它们中各对象的排序码分别是：

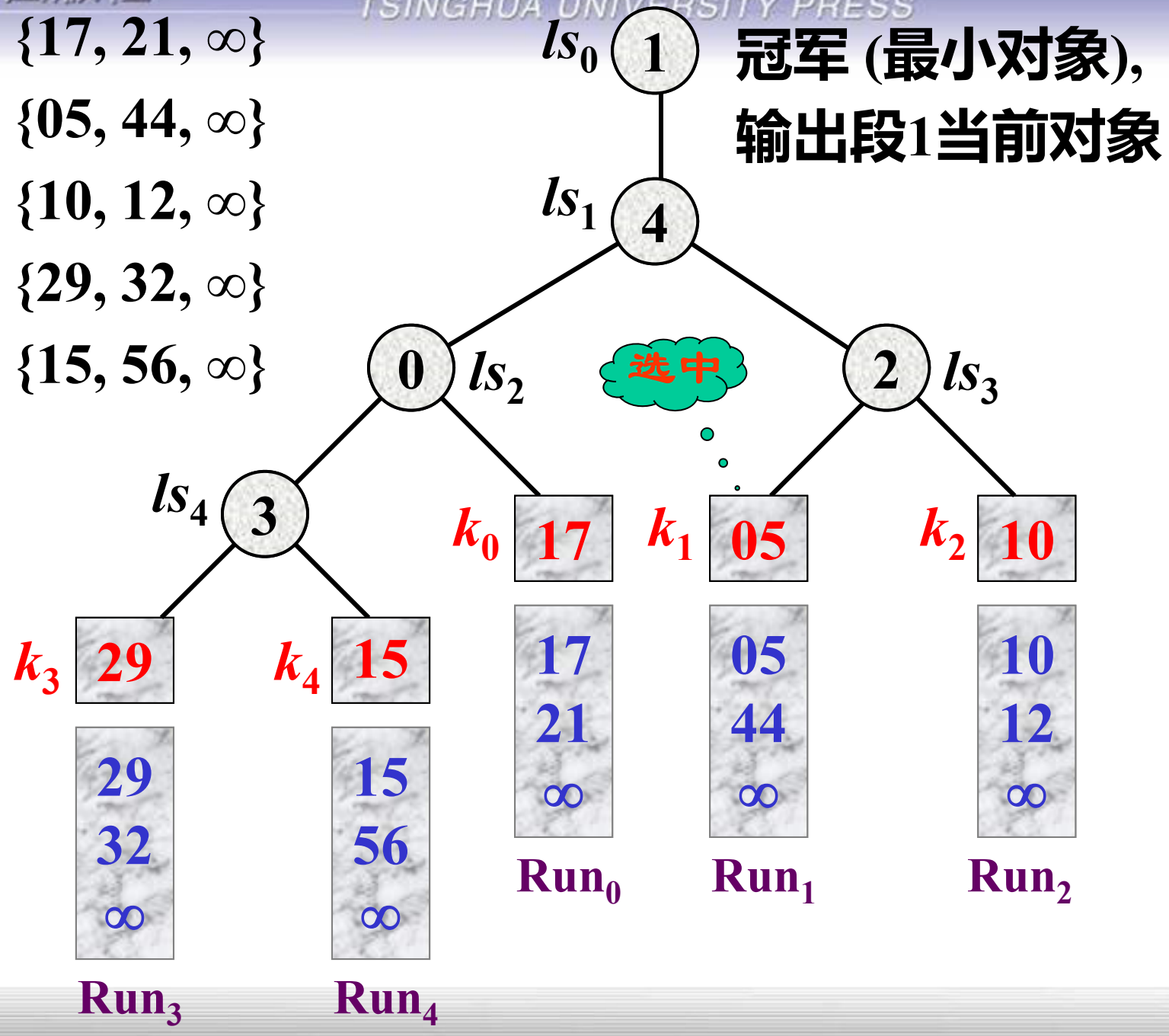
Run0: {17, 21, ∞}

Run1: {05, 44, ∞}

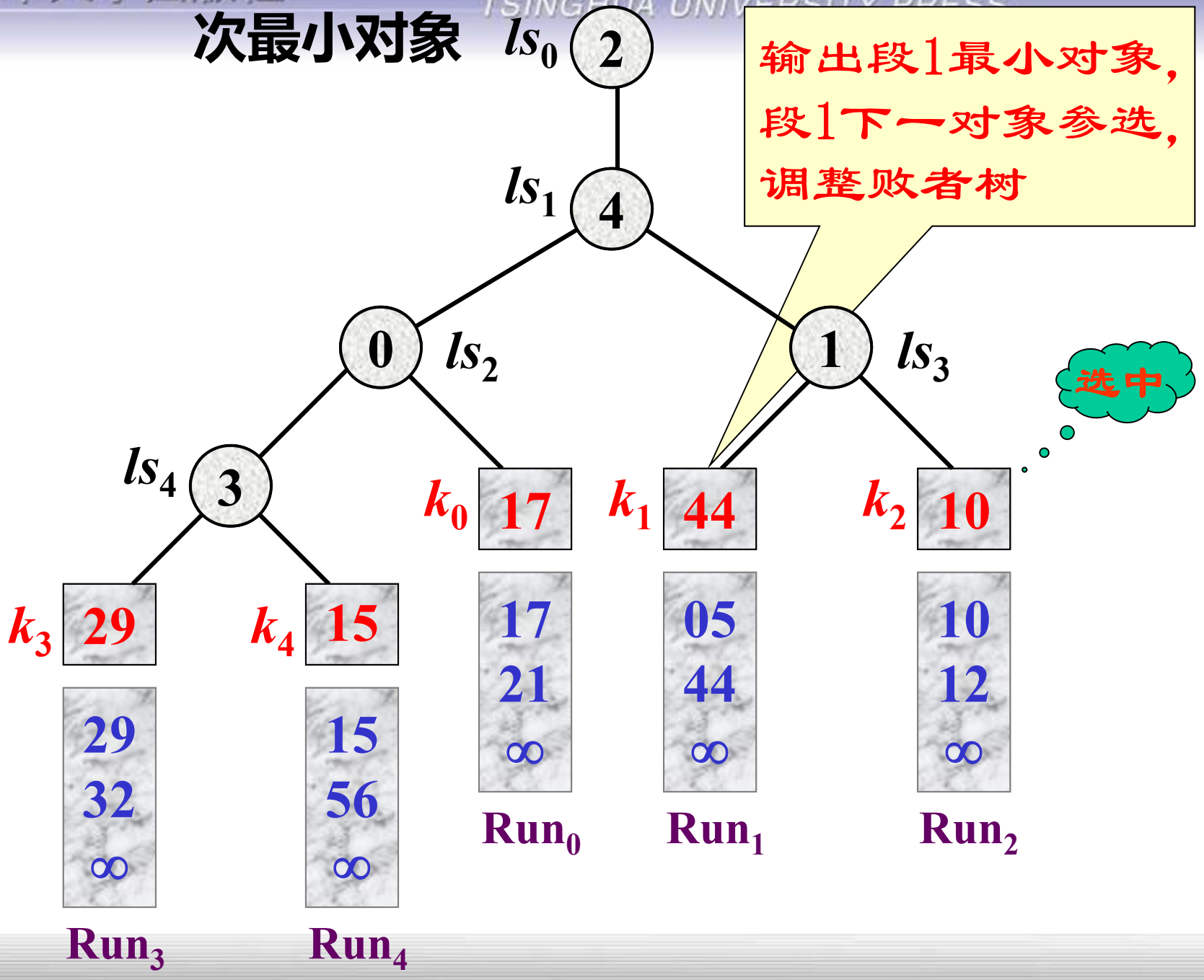
Run2: {10, 12, ∞}

Run3: {29, 32, ∞}

Run4: {15, 56, ∞}



次最小对象


ls_0 2
ls_1 4
0 ls_2
1 ls_3
ls_4 3
输出段1最小对象，
段1下一对象参选，
调整败者树
选中
k_0 17
k_1 44
k_2 10
k_3 29
k_4 15
17
21
∞
05
44
∞
10
12
∞
29
32
∞
15
56
∞
Run_0
Run_1
Run_2
Run_3
Run_4

- 败者树的高度为$\lceil \log_2 k \rceil$，在每次调整，找下一个具有最小排序码对象时，最多做$\lceil \log_2 k \rceil$次排序码比较。
- 在内存中应为每一个归并段分配一个输入缓冲区，其大小应能容纳一个页块的对象，编号与归并段号一致。每个输入缓冲区应有一个指针，指示当前参加归并的对象。
- 在内存中还应设立一个输出缓冲区，其大小相当于一个页块大小。它也有一个缓冲区指针，指示当前可存放结果对象的位置。每当一个对象 i 被选出，就执行OutputRecord(i)操作，将对象存放到输出缓冲区中。

- 在实现利用败者树进行多路平衡归并算法时，把败者树的叶结点和非叶结点分开定义。
- 败者树叶结点key[ ]有k+1个，key[0]到key[k-1]存放各归并段当前参加归并的对象的排序码，key[k]是辅助工作单元，在初始建立败者树时使用：存放一个最小的在各归并段中不可能出现的排序码：-MaxNum。
- 败者树非叶结点loser[ ]有k个，其中loser[1]到loser[k-1]存放各次比较的败者的归并段号，loser[0]中是最后胜者所在归并段号。另外还有一个存放各归并段参加归并对象的数组$r[k]$。

k 路平衡归并排序算法

```
void kwaymerge ( Element *r ) {
    r = new Element[k];                  //创建对象数组
    int *key = new int[k+1];             //创建外结点数组
    int *loser = new int[k];             //创建败者树数组
    for ( int i = 0; i < k; i++ )        //传送参选排序码
      { InputRecord ( r[i] );  key[i] = r[i].key; }
    for ( i = 0; i < k; i++) loser[i] = k;
    key[k] = -MaxNum;                    //初始化
    for ( i = k-1; i; i-- )              //调整形成败者树
        adjust ( key, loser, k, i );
```

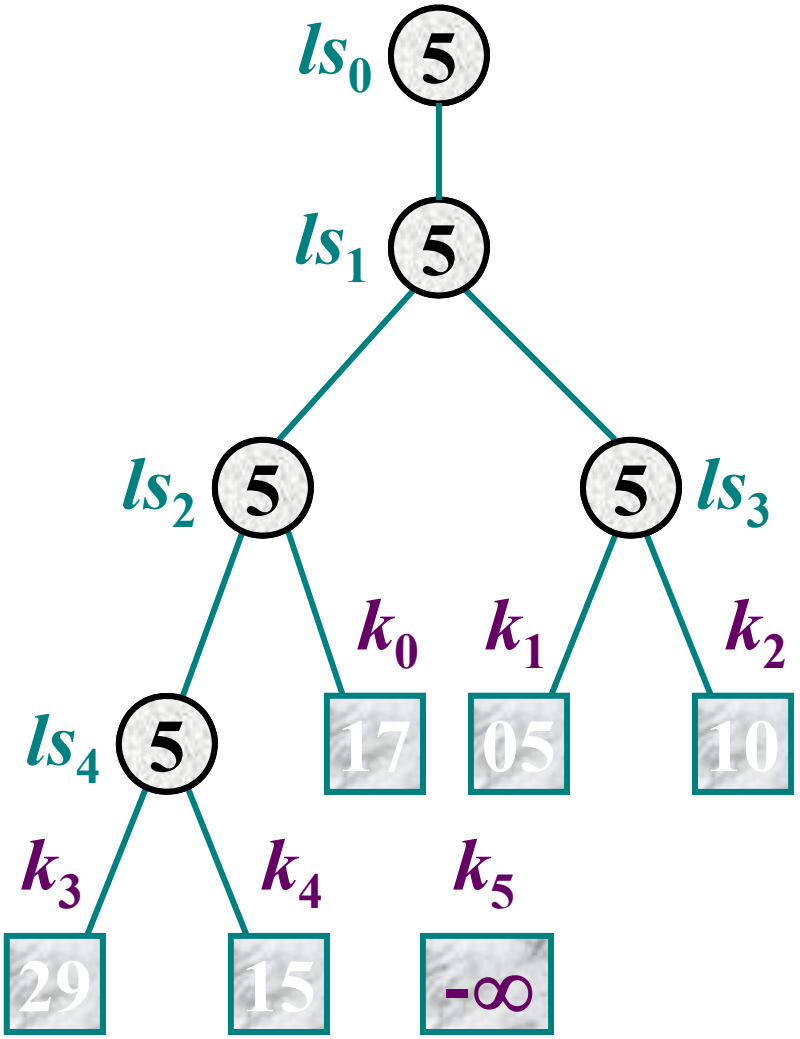
```
    while ( key[loser[0]] != MaxNum ) {   //选冠军
        q = loser[0];                     //最小对象的段号
        OutputRecord ( r[q] );            //输出
        InputRecord ( r[q] );             //从该段补入对象
        key[q] = r[q].key;
        adjust ( key, loser, k, q );      //调整
    }
    Output end of run marker; //输出段结束标志
    delete [ ] r;  delete [ ] key;  delete [ ] loser;
}
```

自某叶结点*key*[*q*]到败者树根结点的调整算法

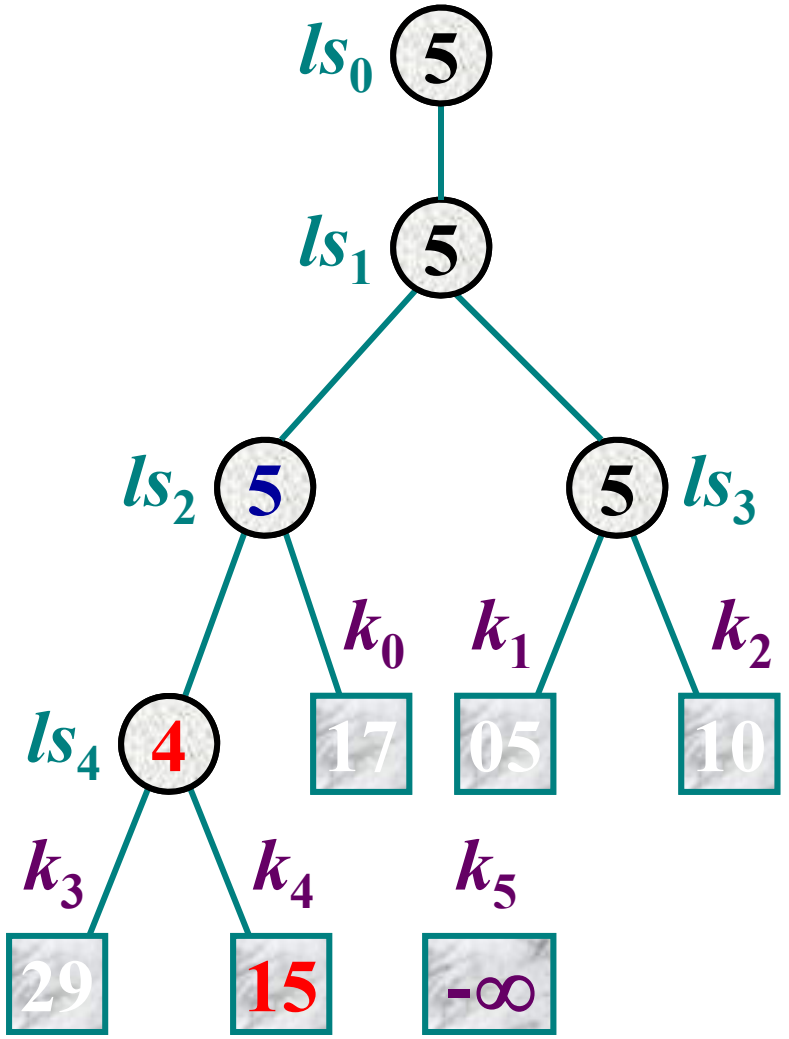
```
void adjust ( int key[ ]; int loser[ ]; const int k;
        const int q ) {
//q指示败者树某外结点key[q], 从该结点起到
//根进行比较, 将最小 key 对象所在归并段的
//段号记入loser[0]。 k是外结点key[ ]个数。
    for ( int t = (k+q) / 2; t > 0; t /= 2 )  // t是q双亲
        if ( key[loser[t]] < key[q]) {
                        //败者记入loser[t], 胜者记入q
            int temp = q;   q = loser[t];  loser[t] = temp;
        }               //q与loser[t]交换
    loser[0] = q;
}
```

- 每选出一个当前排序码最小的对象，就需要在将它送入输出缓冲区之后，从相应归并段的输入缓冲区中取出下一个参加归并的对象,替换已经取走的最小对象，再从叶结点到根结点，沿某一特定路径进行调整，将下一个排序码最小对象的归并段号调整到loser[0]中。
- 段结束标志MaxNum升入loser[0]，排序完成。
- 归并路数 k 不是越大越好。归并路数 k 增大，相应需增加输入缓冲区个数。如果可供使用的内存空间不变，势必要减少每个输入缓冲区的容量，使内外存交换数据的次数增大。

利用败者树进行5路平衡归并的过程

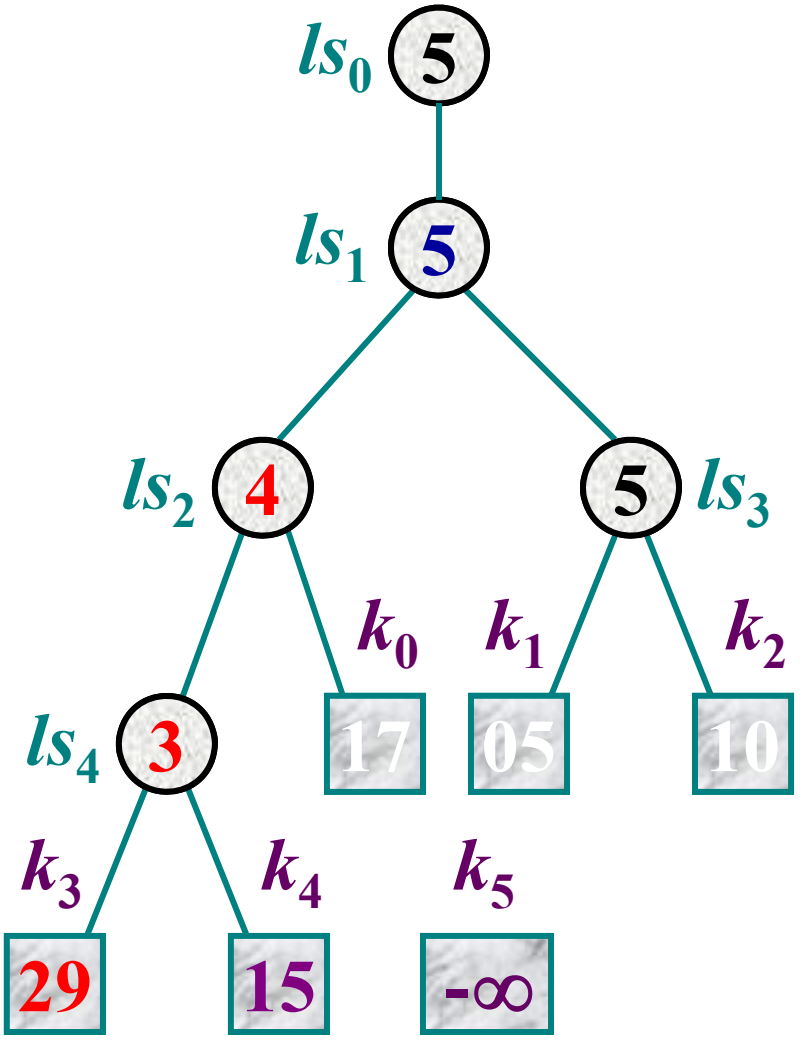


(1) 初始状态

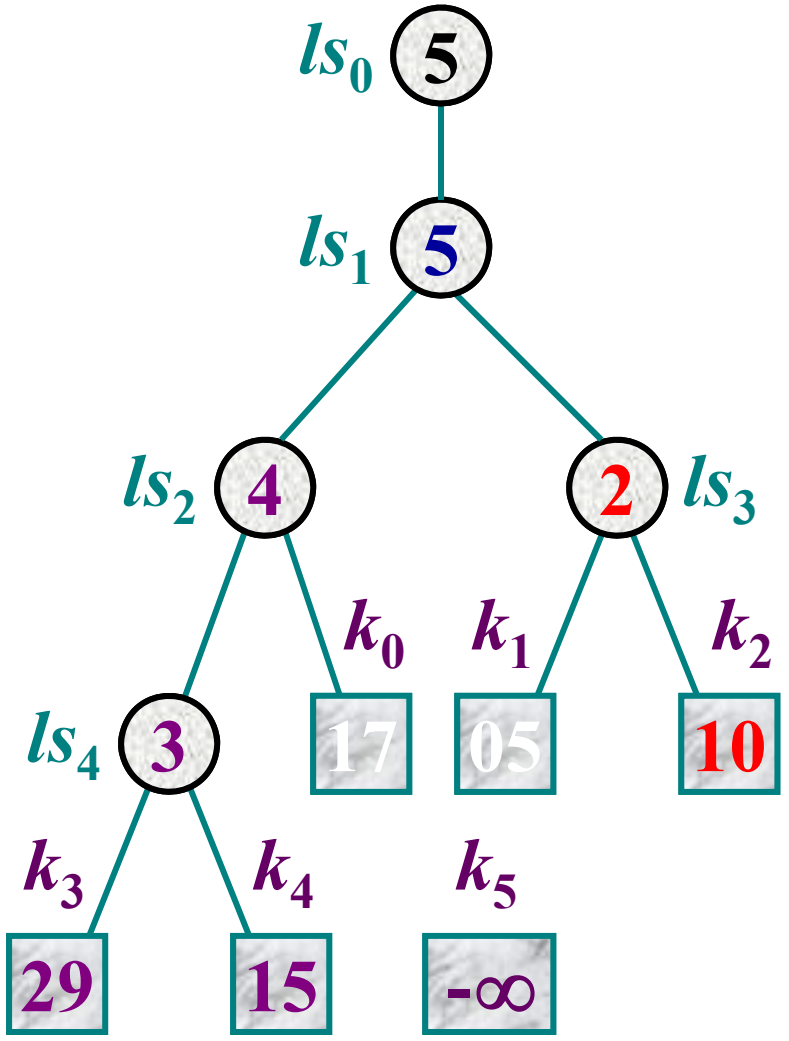


(2) 加入15, 调整

利用败者树进行5路平衡归并的过程

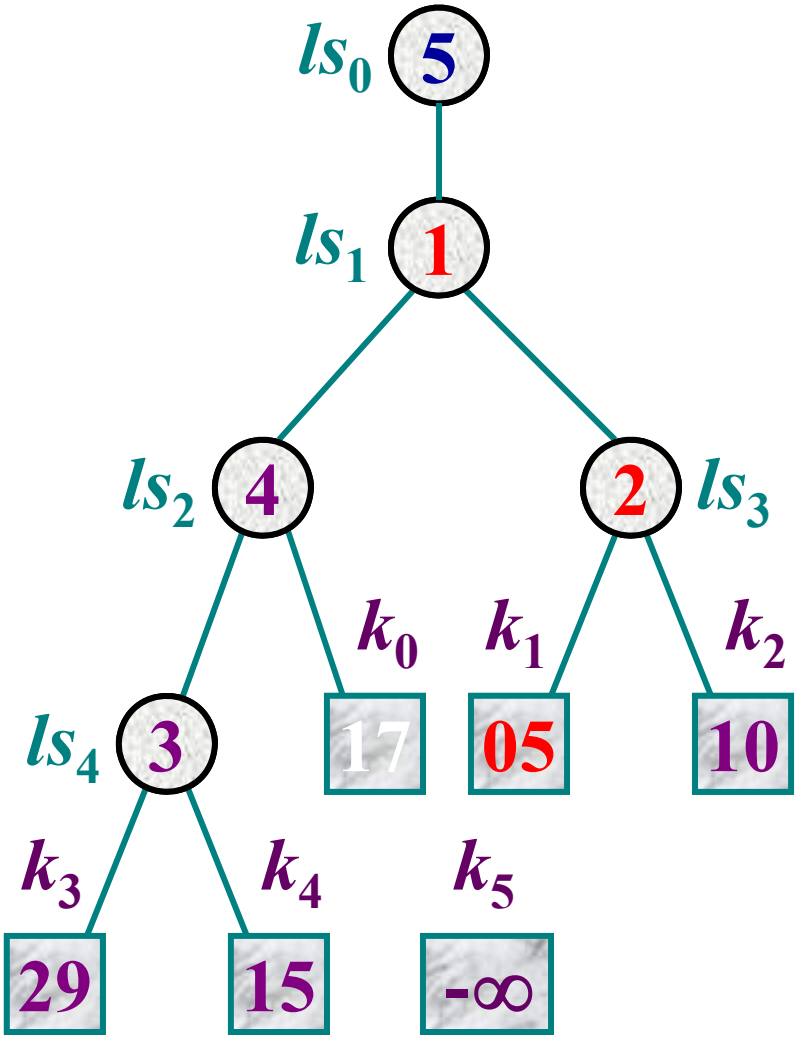


(3)加入29, 调整

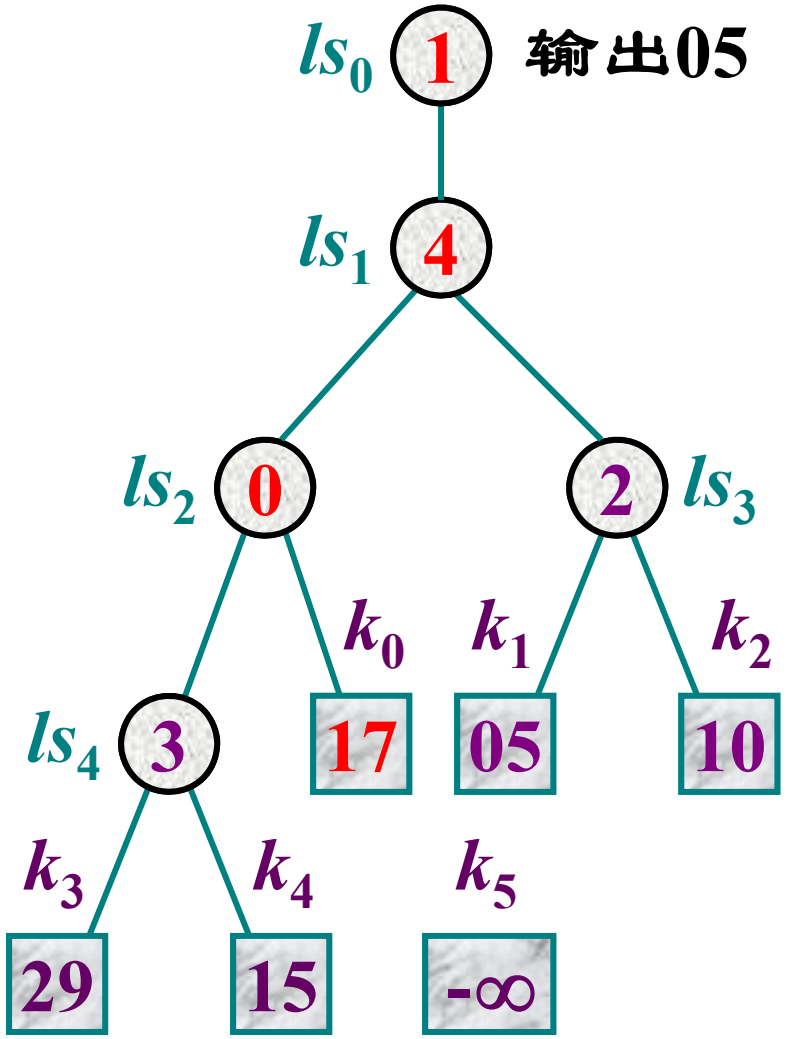


(4) 加入10, 调整

利用败者树进行5路平衡归并的过程

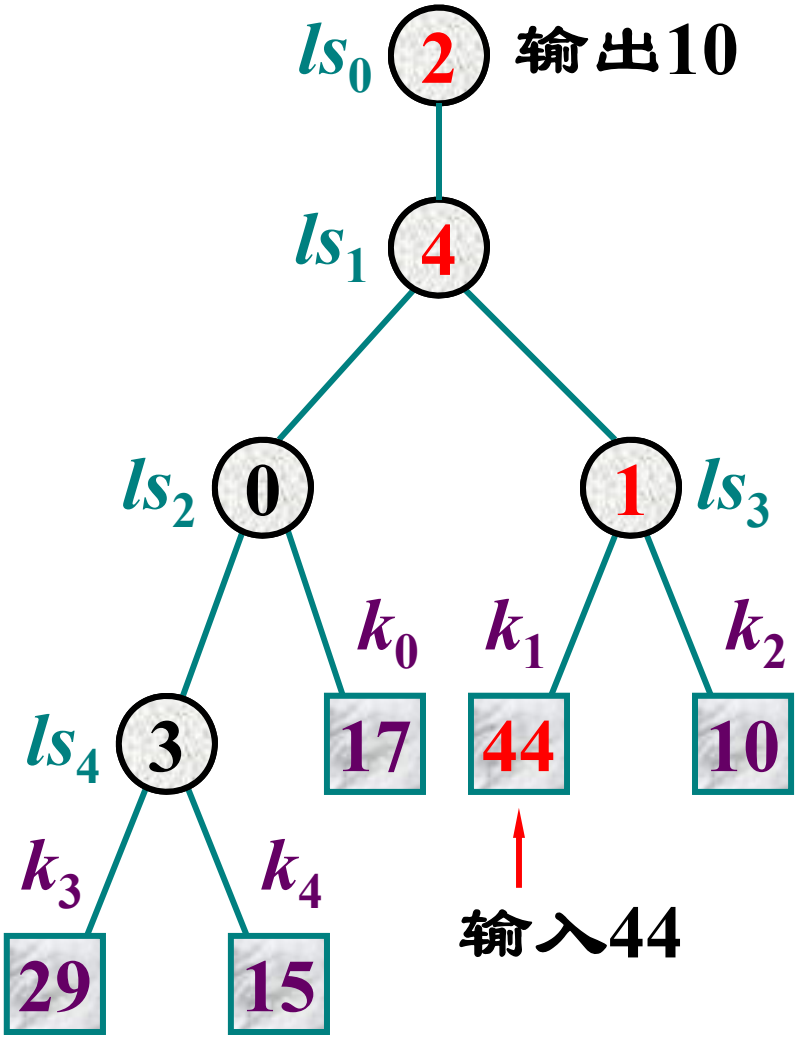


(5) 加入05, 调整

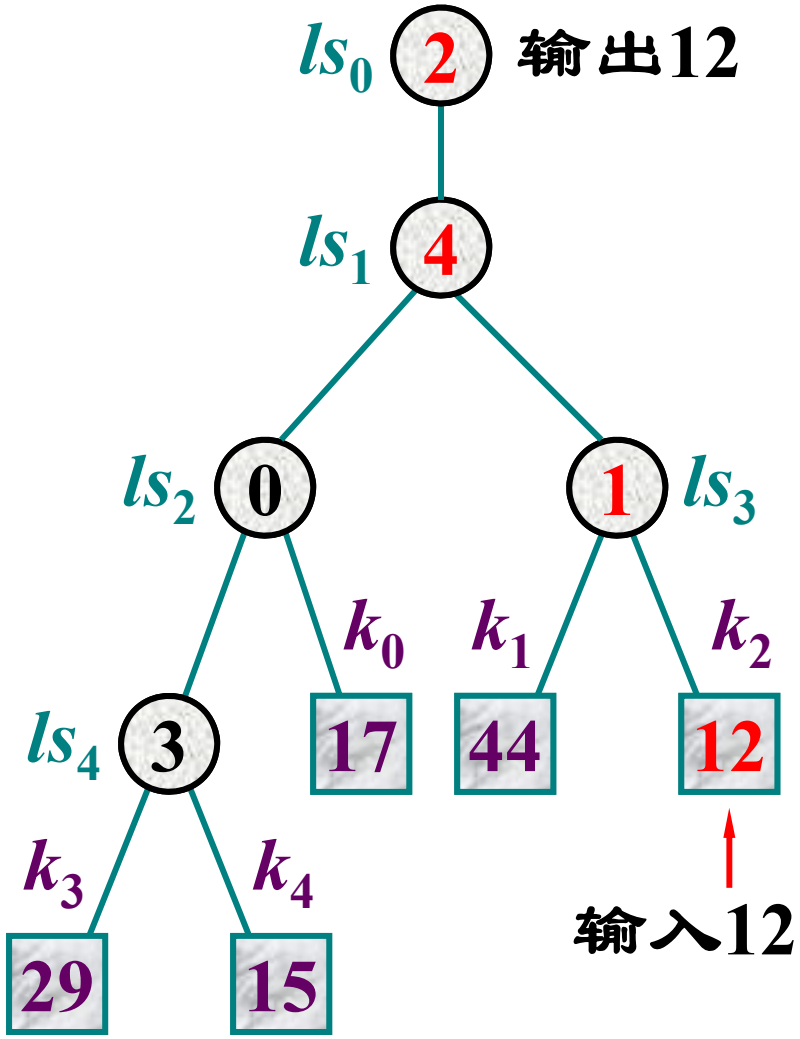


(6) 加入17, 调整

利用败者树进行5路平衡归并的过程

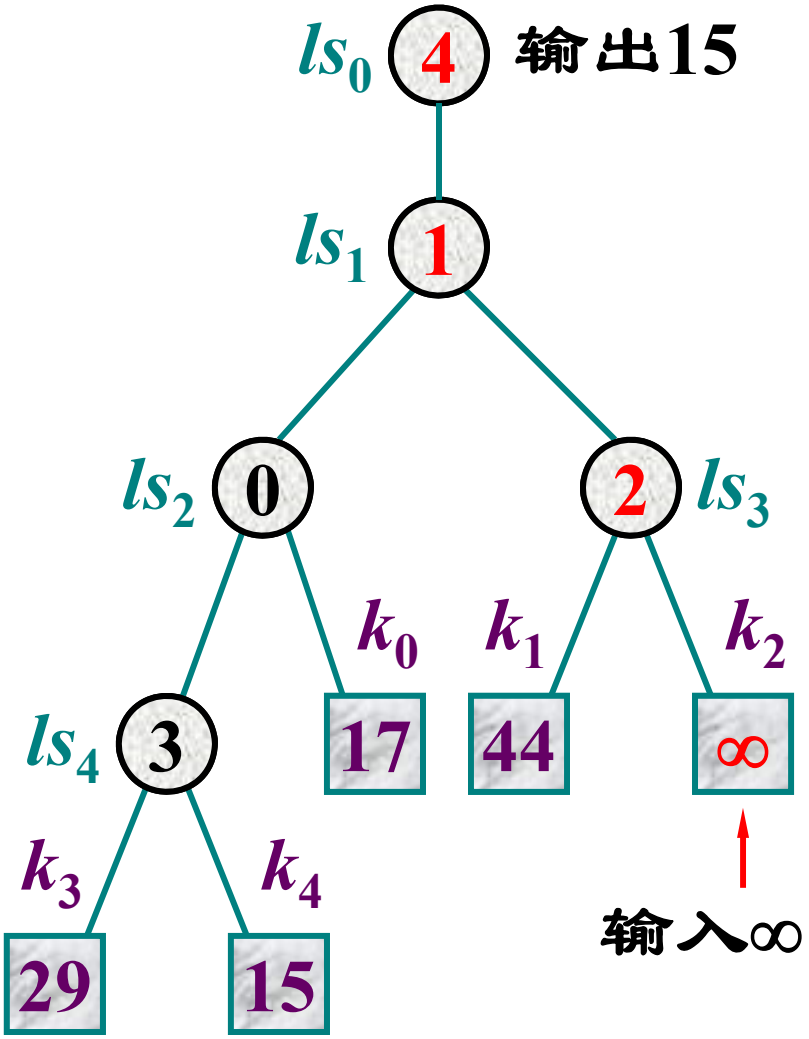


(7) 输出05后调整

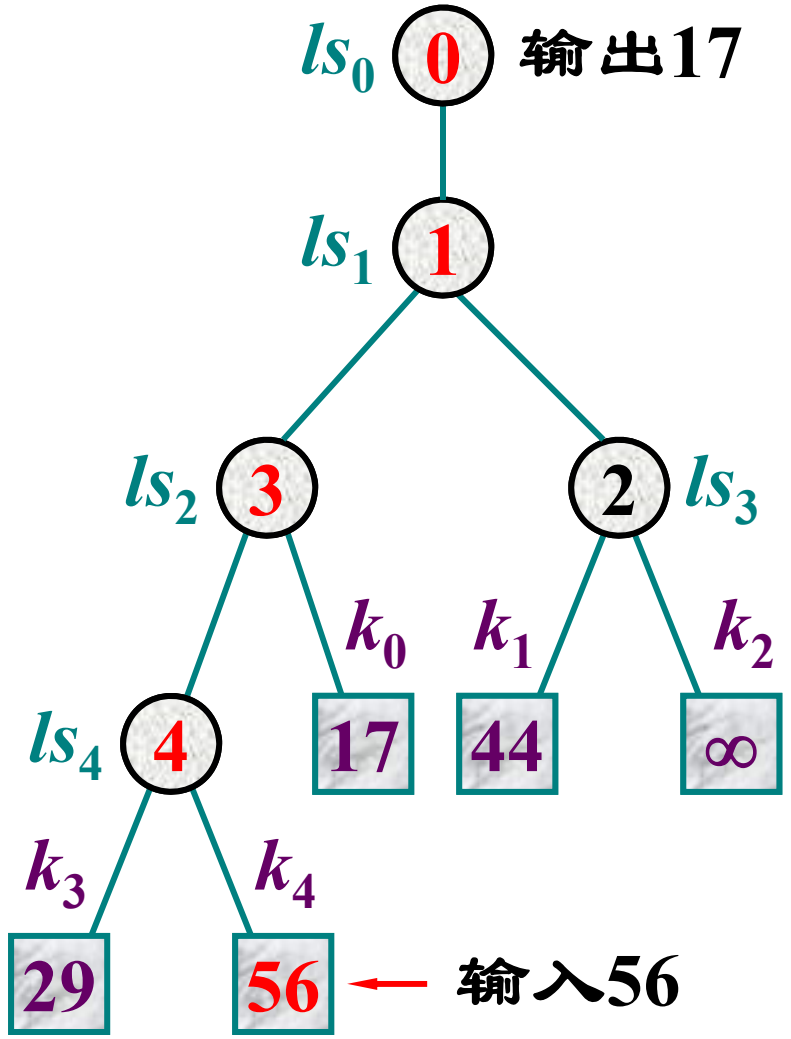


(8) 输出10后调整

利用败者树进行5路平衡归并的过程

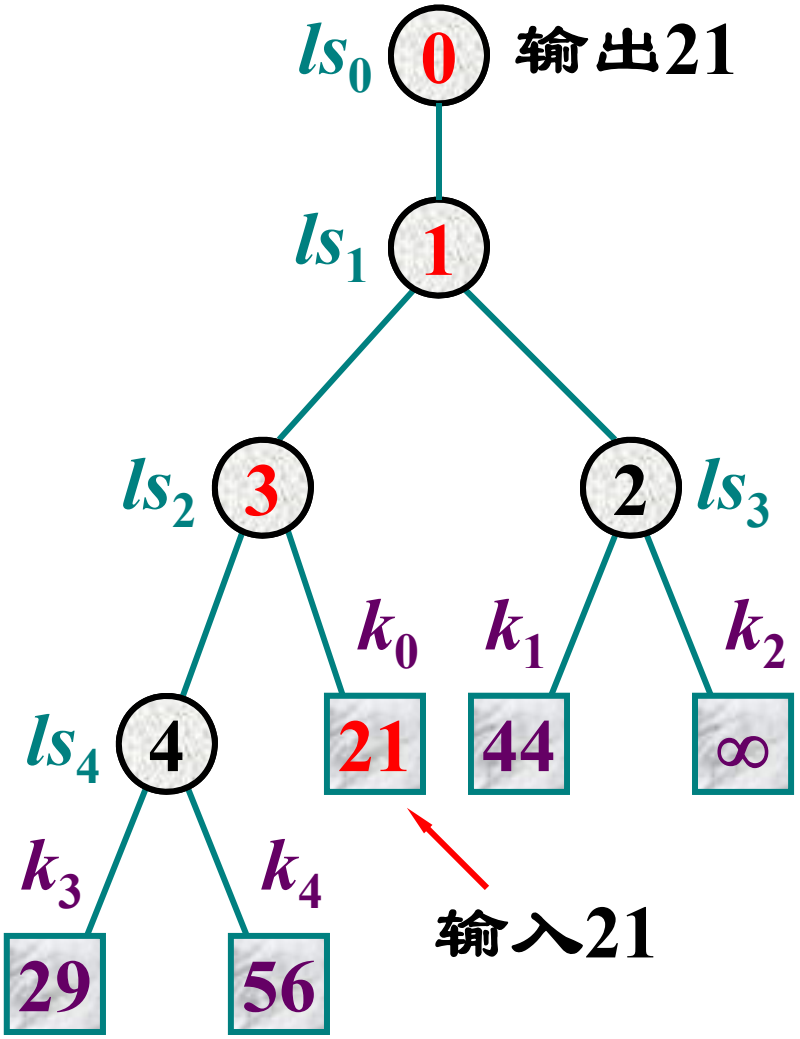


(9) 输出12后调整

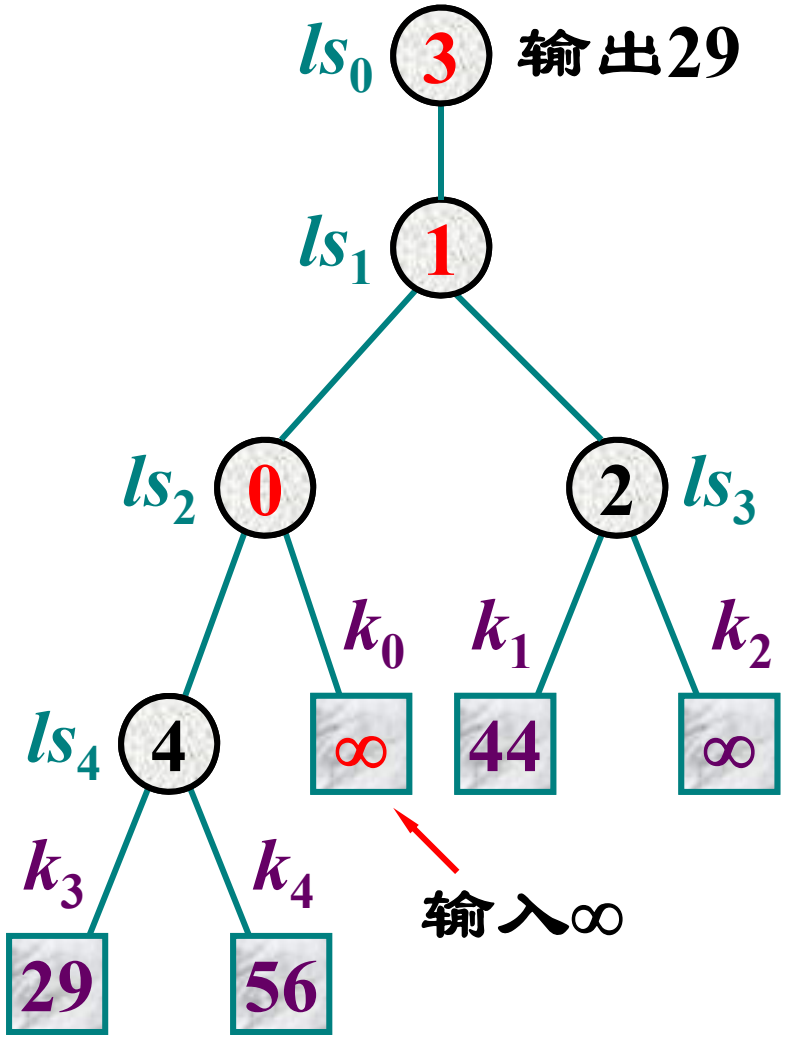


(10) 输出15后调整

利用败者树进行5路平衡归并的过程

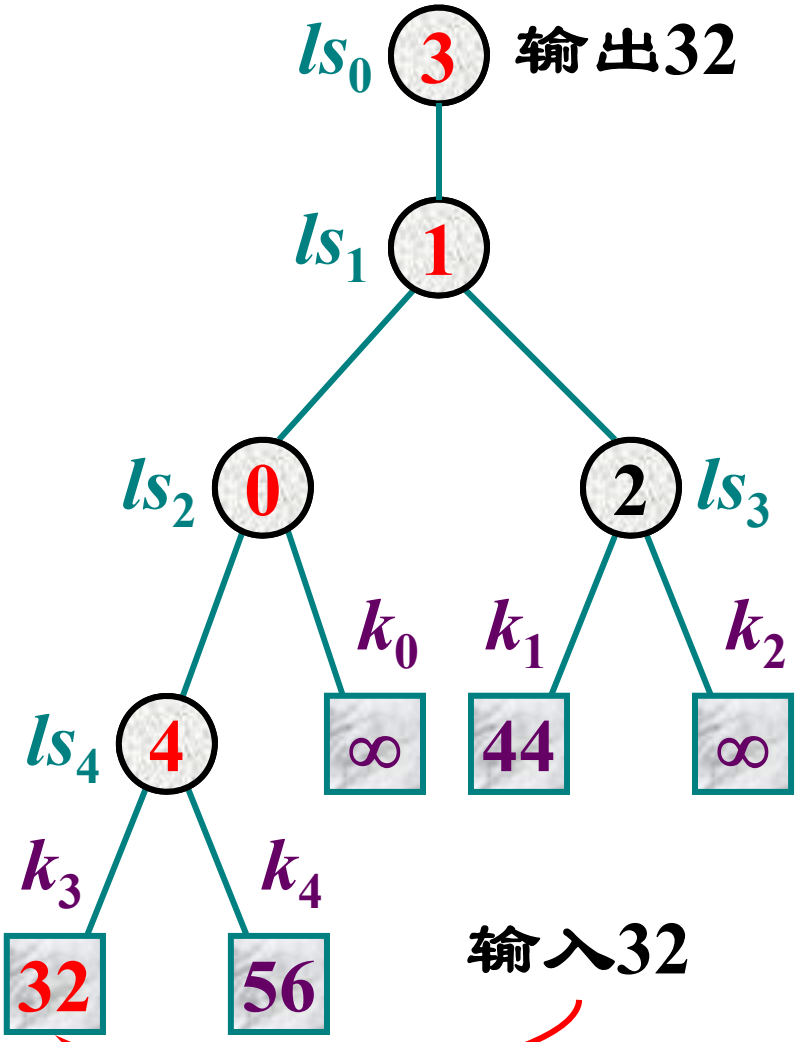


(11) 输出17后调整

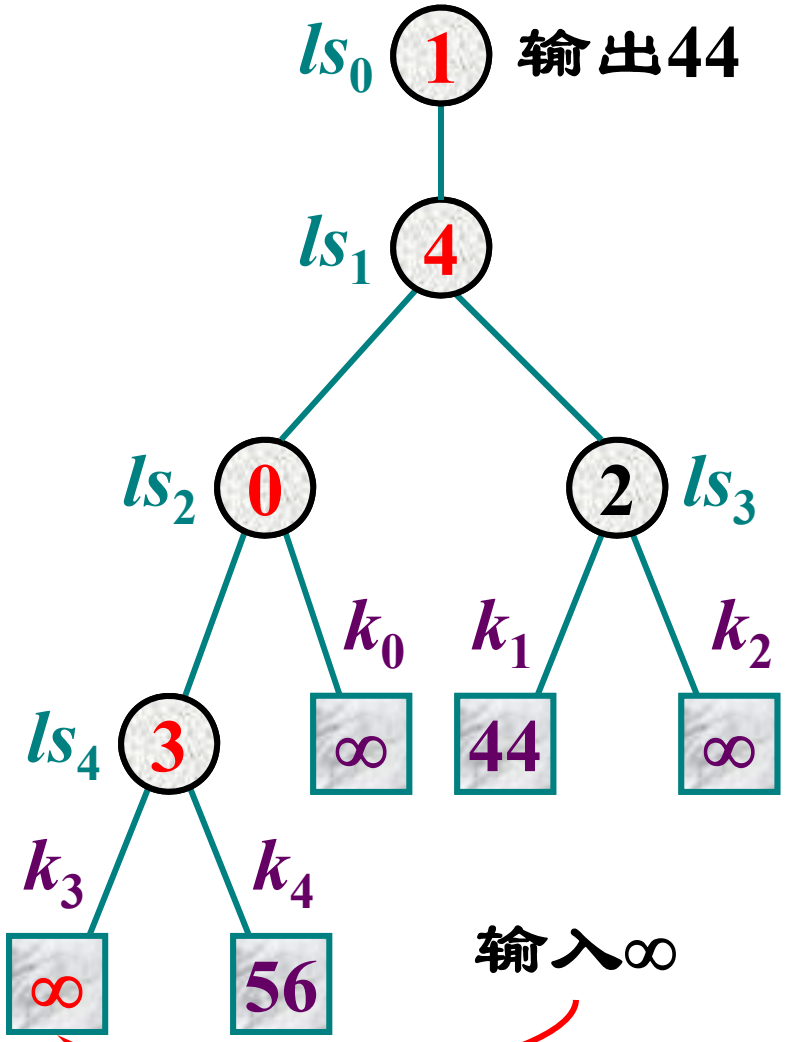


(12) 输出21后调整

利用败者树进行5路平衡归并的过程

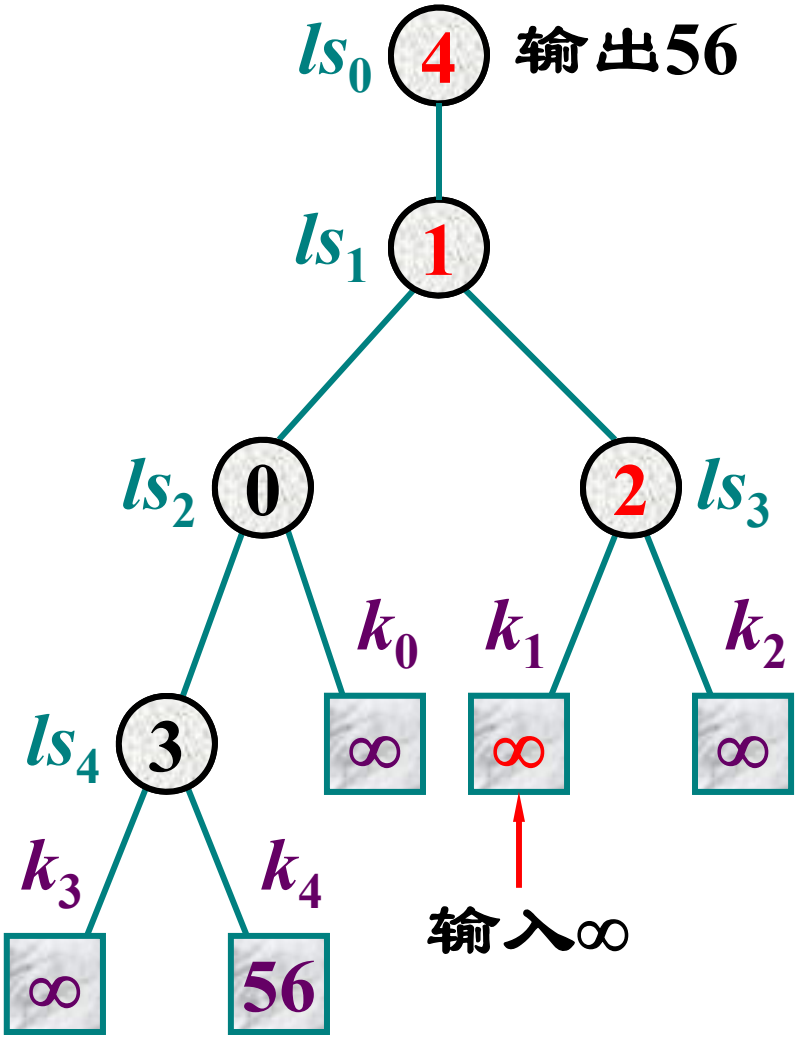


(13) 输出29后调整

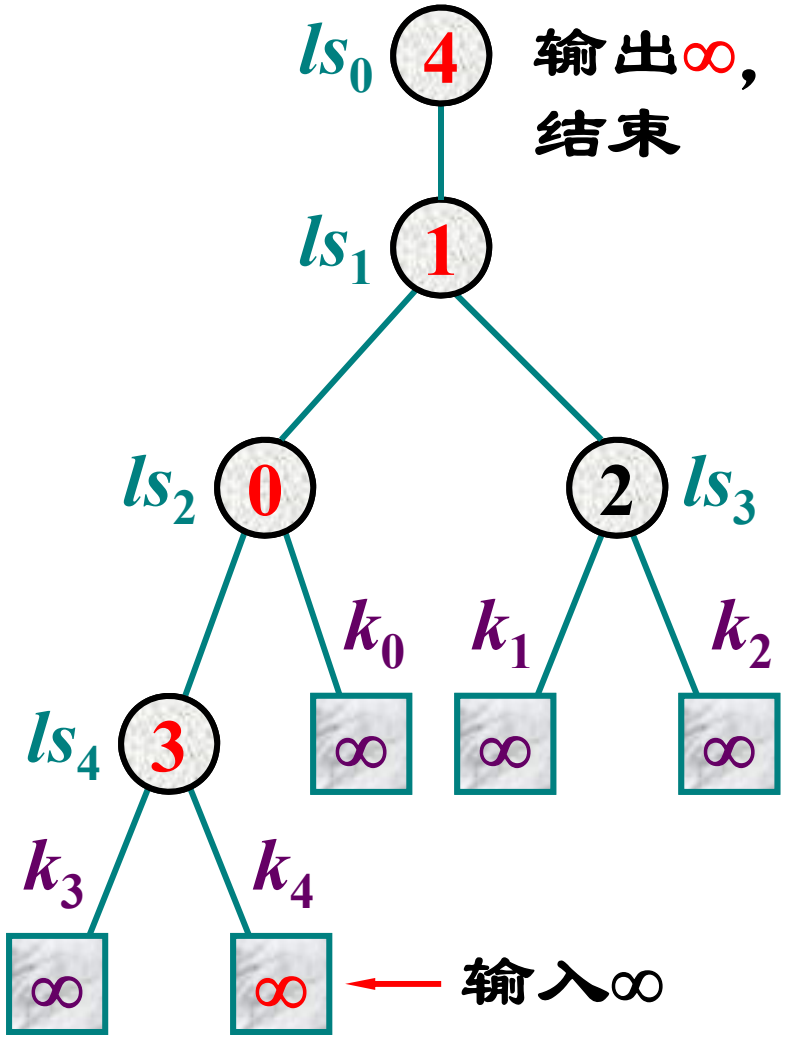


(14) 输出32后调整

利用败者树进行5路平衡归并的过程



(15) 输出44后调整



(16) 输出56后调整

初始归并段的生成 (Run Generation)

- 为减少读写磁盘次数, 除增加归并路数 k 外, 还可减少初始归并段个数m。 在总对象数n一定时, 要减少m, 必须增大初始归并段长度。
- 如果规定每个初始归并段等长, 则此长度应根据生成它的内存工作区空间大小而定, 因而m的减少也就受到了限制。
- 为了突破这个限制, 可采用败者树来生成初始归并段。 在使用同样大内存工作区的情况下, 可以生成平均比原来等长情况下大一倍的初始归并段, 从而减少初始归并段个数。

- 设输入文件*FI*中各对象的排序码序列为 { 17, 21, 05, 44, 10, 12, 56, 32, 29 }。
- 选择和置换过程的步骤如下：

① 从输入文件*FI*中把 *k* 个对象读入内存中，并构造败者树。(内存中存放对象的数组*r*可容纳的对象个数为 *k*)。

② 利用败者树在 *r* 中选择一个排序码最小的对象*r*[*q*]，其排序码存入LastKey作为门槛。以后再选出的排序码比它大的对象归入本归并段，比它小的归入下一归并段。

③ 将此*r*[*q*]对象写到输出文件*FO*中。(*q* 是叶结点序号)。

④ 若*FI*未读完，则从*FI*读入下一个对象，置换*r*[*q*]及败者树中的*key*[*q*]。

⑤ 调整败者树，从所有排序码比LastKey大的对象中选择一个排序码最小的对象*r*[*q*]作为门槛，其排序码存入LastKey。

⑥ 重复③～⑤，直到在败者树中选不出排序码比LastKey大的对象为止。此时，在输出文件*FO*中得到一个初始归并段，在它最后加一个归并段结束标志。

⑦ 重复②～⑥，重新开始选择和置换，产生新的初始归并段，直到输入文件*FI*中所有对象选完为止。

输入文件 *FI*	内存数组 *r*	输出文件 *FO*
21 05 44 10 12 56 32 29		
44 10 12 56 32 29	17 21 05	
10 12 56 32 29	17 21 44	05
12 56 32 29	10 21 44	05 17
56 32 29	10 12 44	05 17 21
32 29	10 12 56	05 17 21 44
29	10 12 32	05 17 21 44 56
29	10 12 32	05 17 21 44 56 ∞
29	10 12 32	
	29 12 32	10
	29 — 32	10 12
	— — 32	10 12 29
	— — —	10 12 29 32
	— — —	10 12 29 32 ∞

- 若按在 k 路平衡归并排序中所讲的，每个初始归并段的长度与内存工作区的长度一致，则上述9个对象可分成3个初始归并段：

  Run0 { 05, 17, 21 }

  Run1 { 10, 12, 44 }

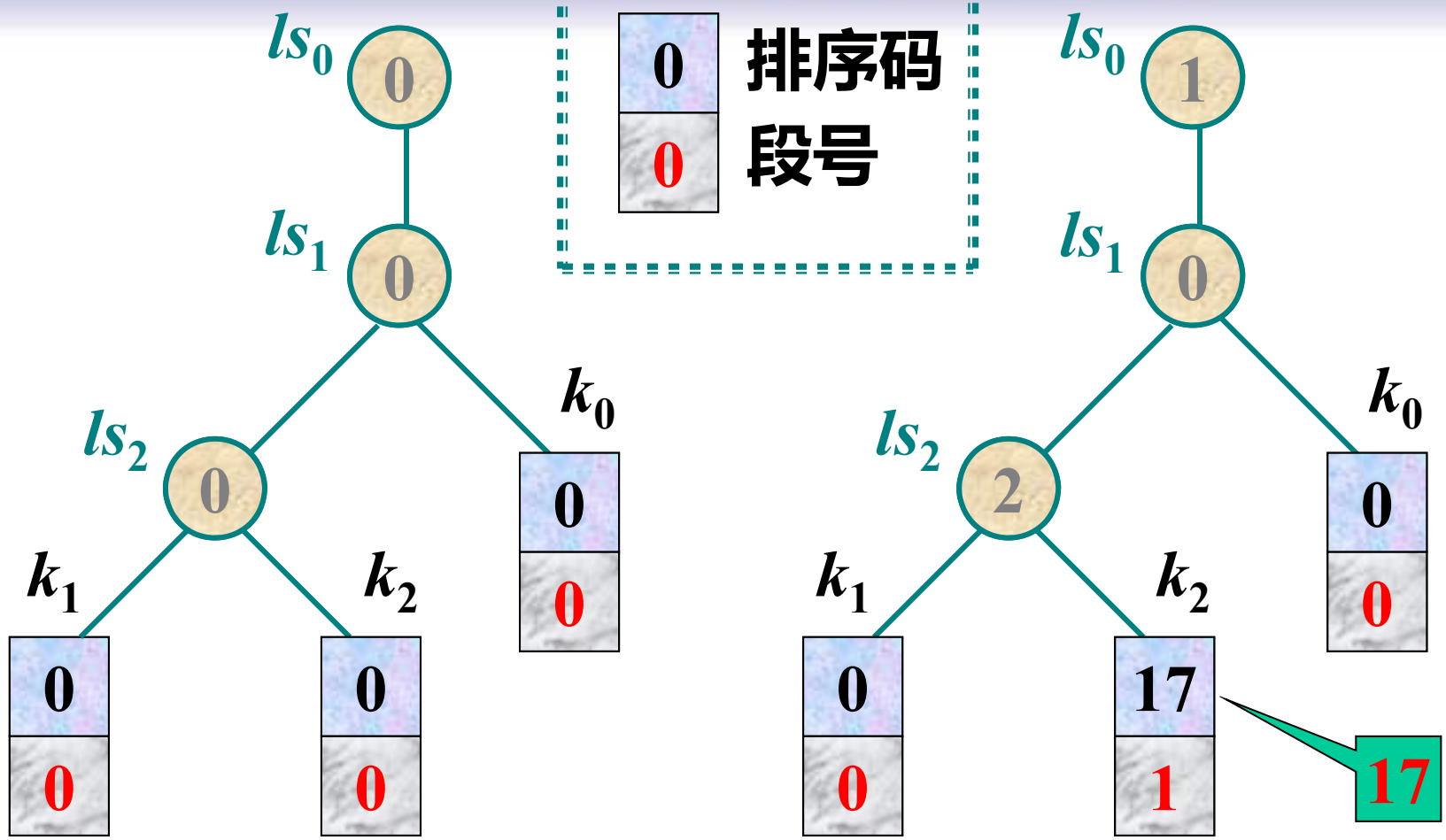
  Run2 { 29, 32, 56 }

- 但采用上述选择与置换的方法，可生成2个长度不等的初始归并段：

  Run0 { 05, 17, 21, 44, 56 }

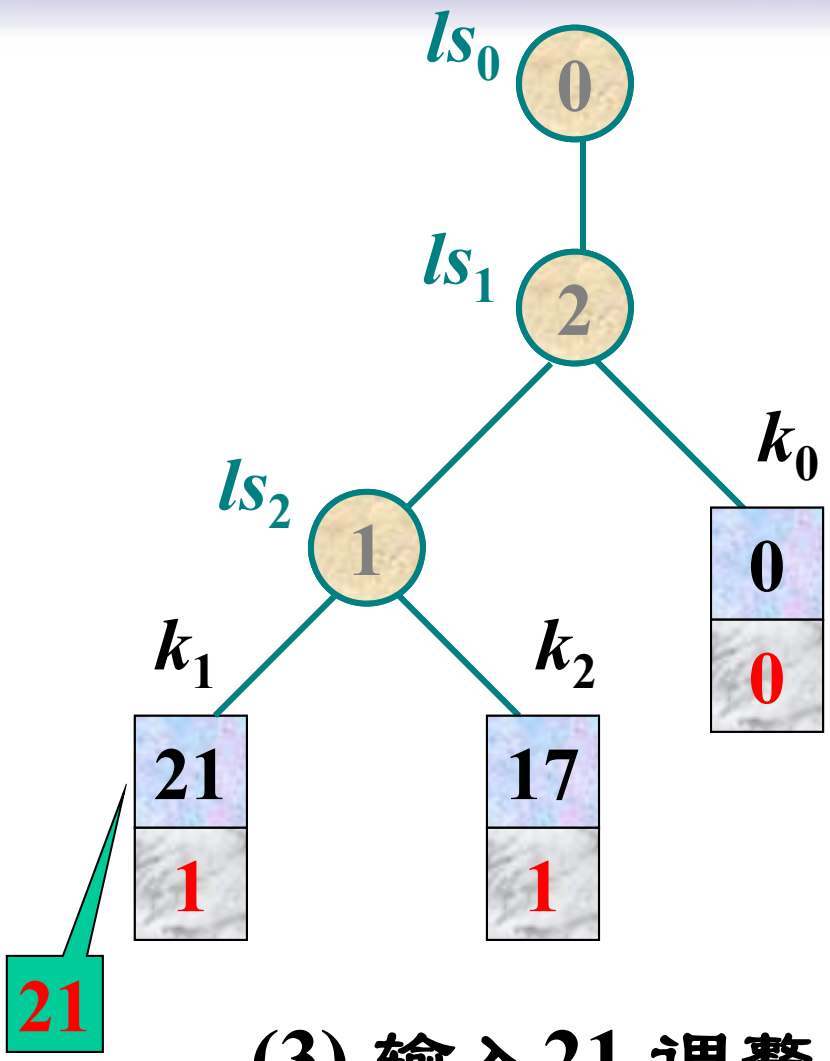
  Run1 { 10, 12, 29, 32 }

- 在利用败者树生成**不等长初始归并段**的算法和调整败者树并选出最小对象的算法中，用两个条件来决定谁为败者，谁为胜者。
  - 首先比较两个对象所在归并段的段号，**段号小者为胜者，段号大者为败者**；
  - 在归并段的段号相同时，**排序码小者为胜者，排序码大者为败者**。
- 比较后把败者对象在对象数组 *r* 中的序号记入它的双亲结点中，把胜者对象在对象数组 *r* 中的序号记入工作单元 *s* 中，向更上一层进行比较，最后的胜者记入 loser[0]中。

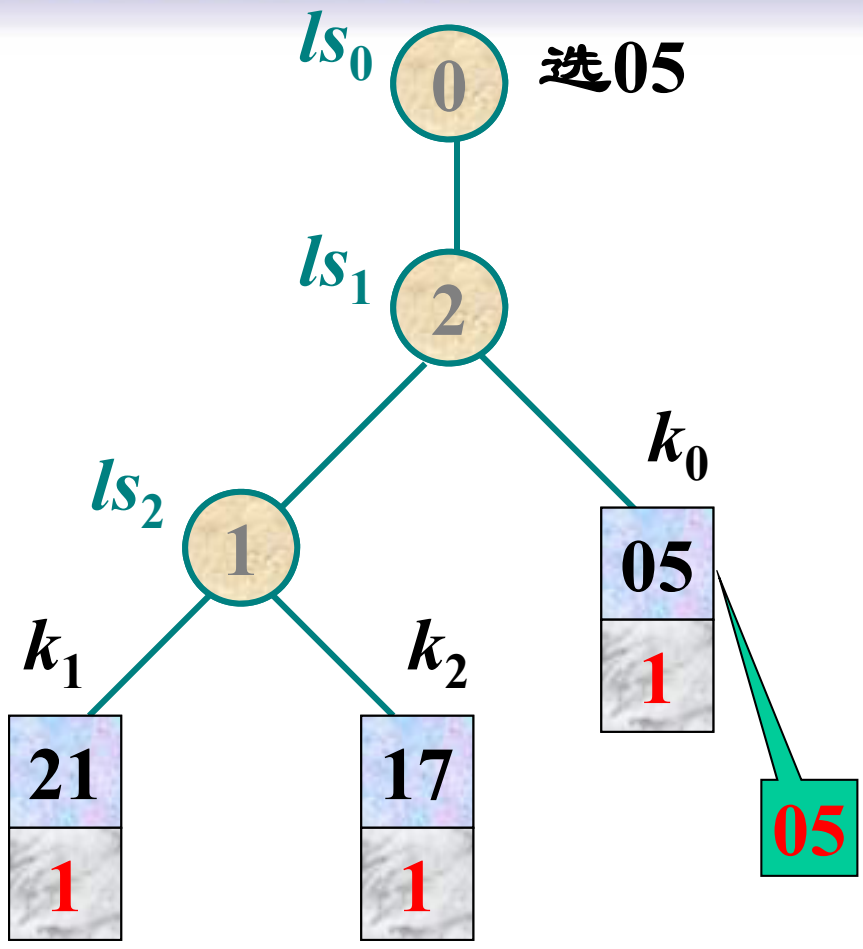


(1) 初始化　　(2) 输入17, 调整

{ 17, 21, 05, 44, 10, 12, 56, 32, 29 }

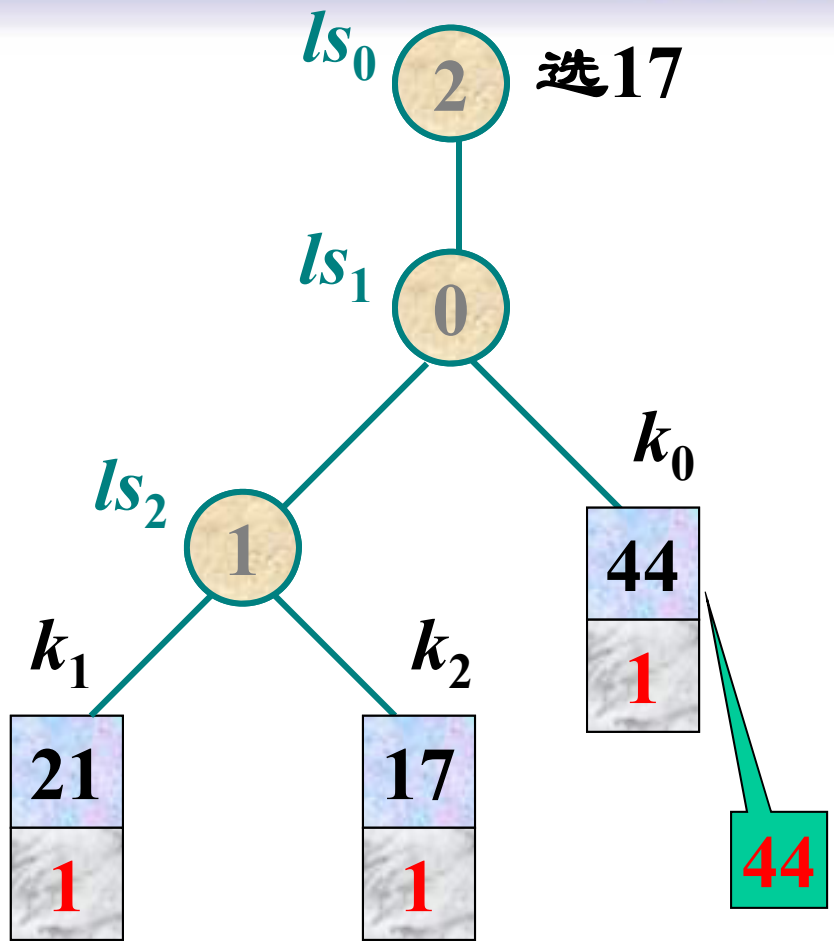


(3) 输入21,调整

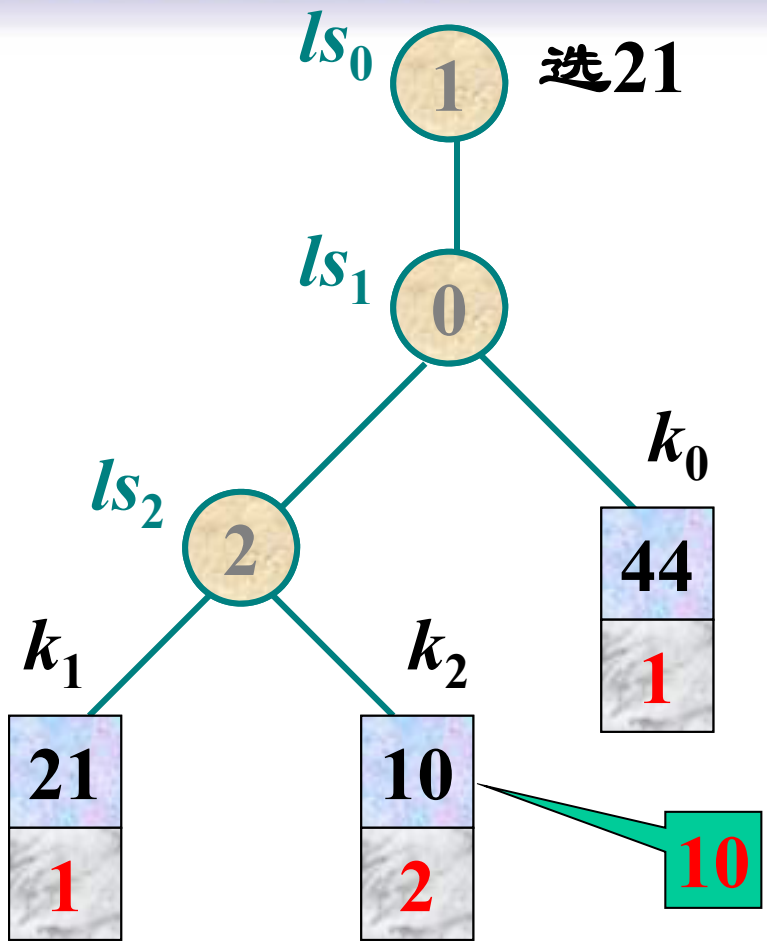


(4) 输入05, 建败者树

{ 17, 21, 05, 44, 10, 12, 56, 32, 29 }

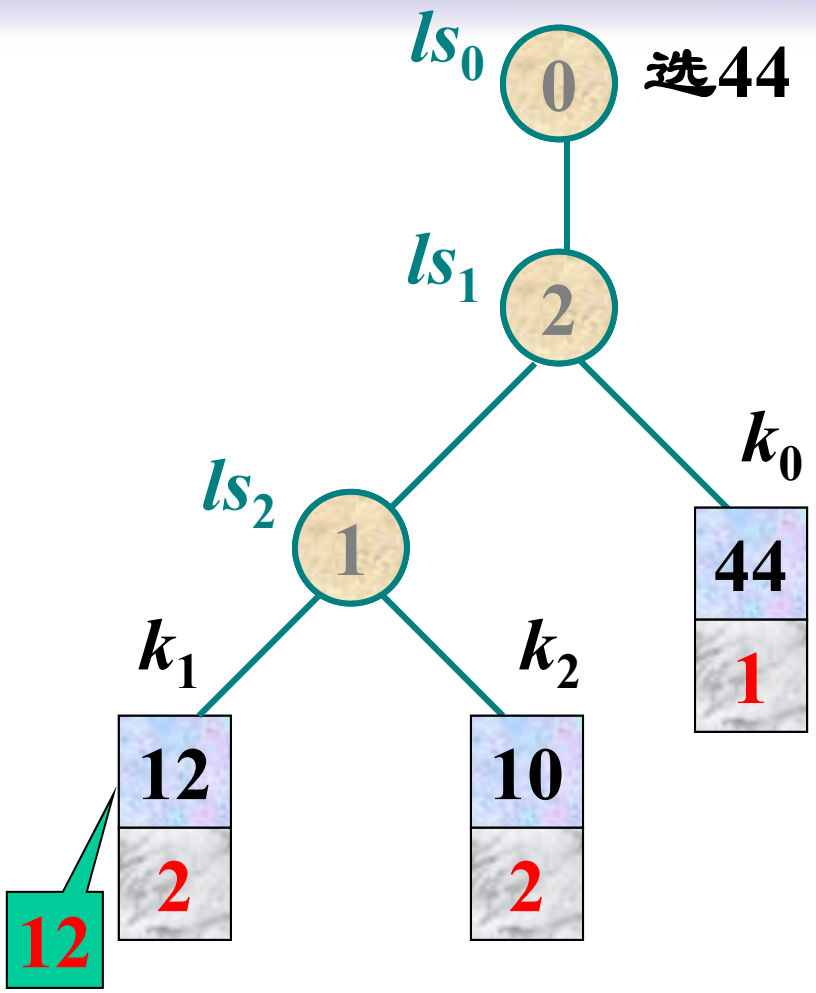


(5) lastKey=05, 置换 k_0, 选择17

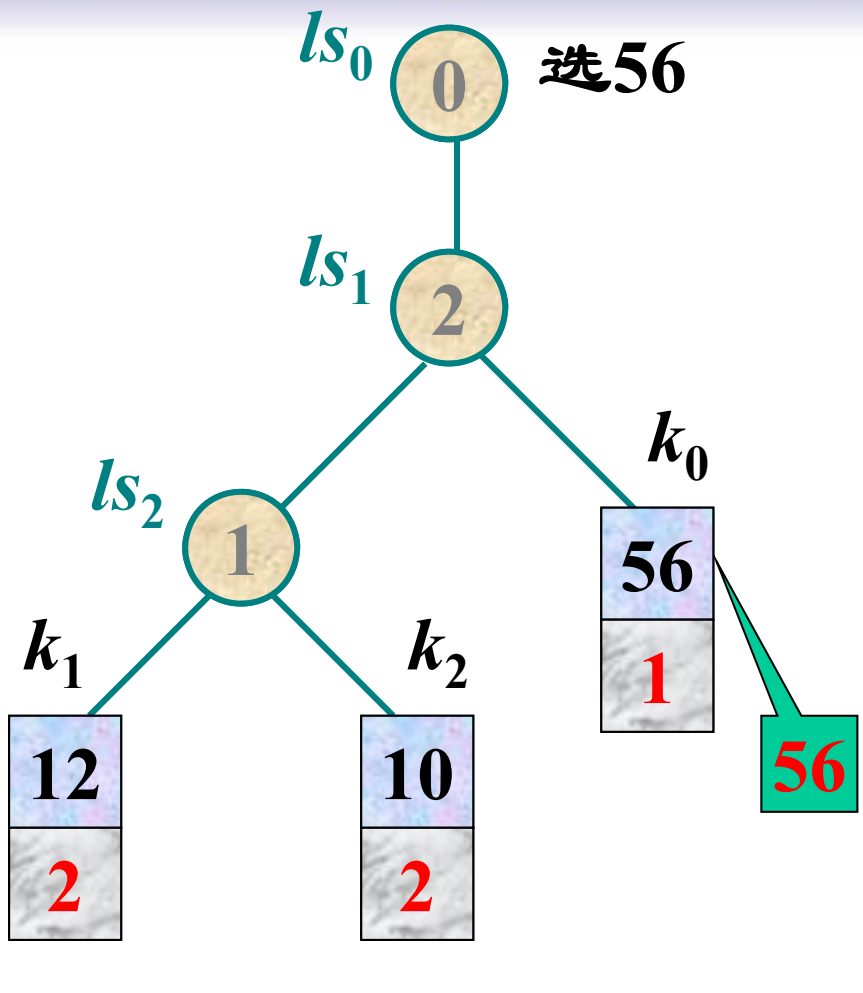


(6) lastKey=17, 置换 k_2, 段号加1, 选择21

{ 17, 21, 05, 44, 10, 12, 56, 32, 29 }

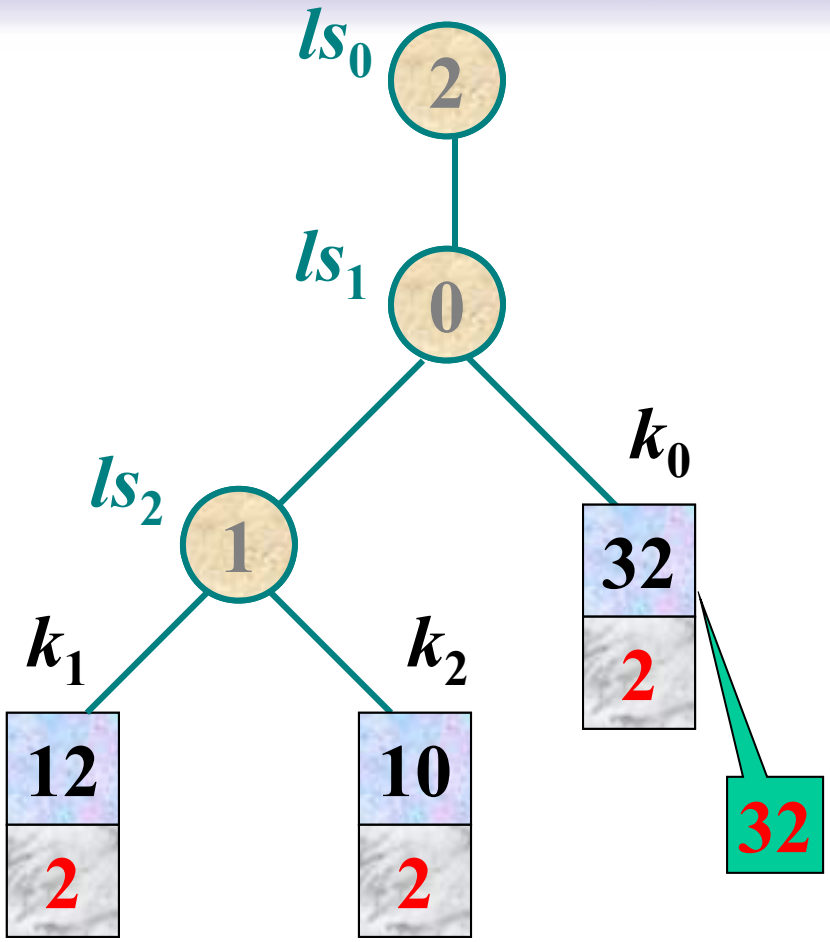


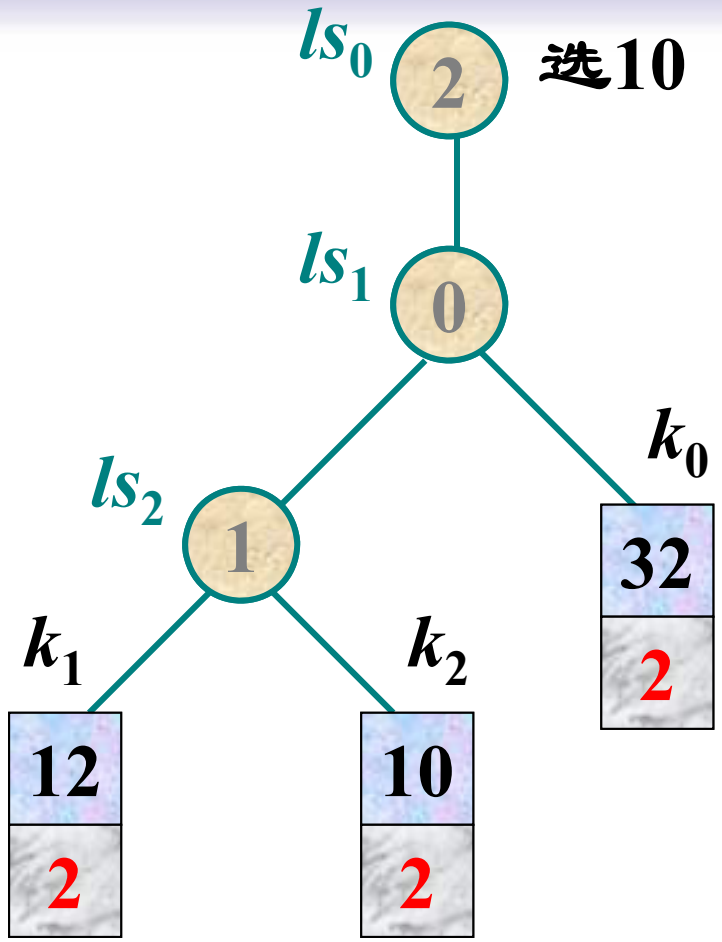
(7) lastKey=21, 置换 k_1, 段号加1, 选择44



(8) lastKey=44, 置换 k_0, 选择56

{ 17, 21, 05, 44, 10, 12, 56, 32, 29 }

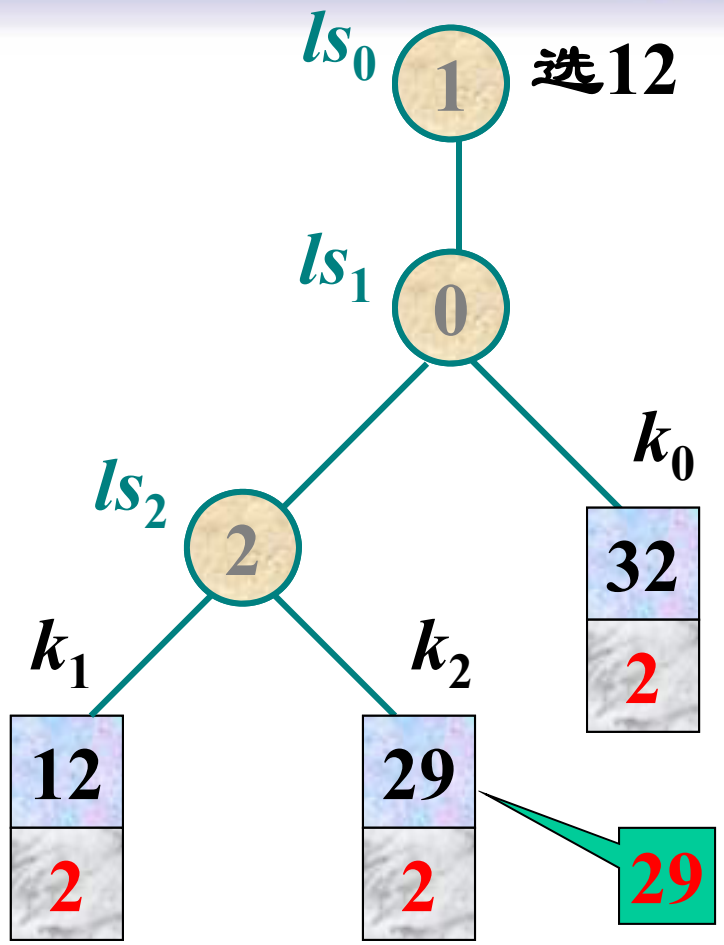




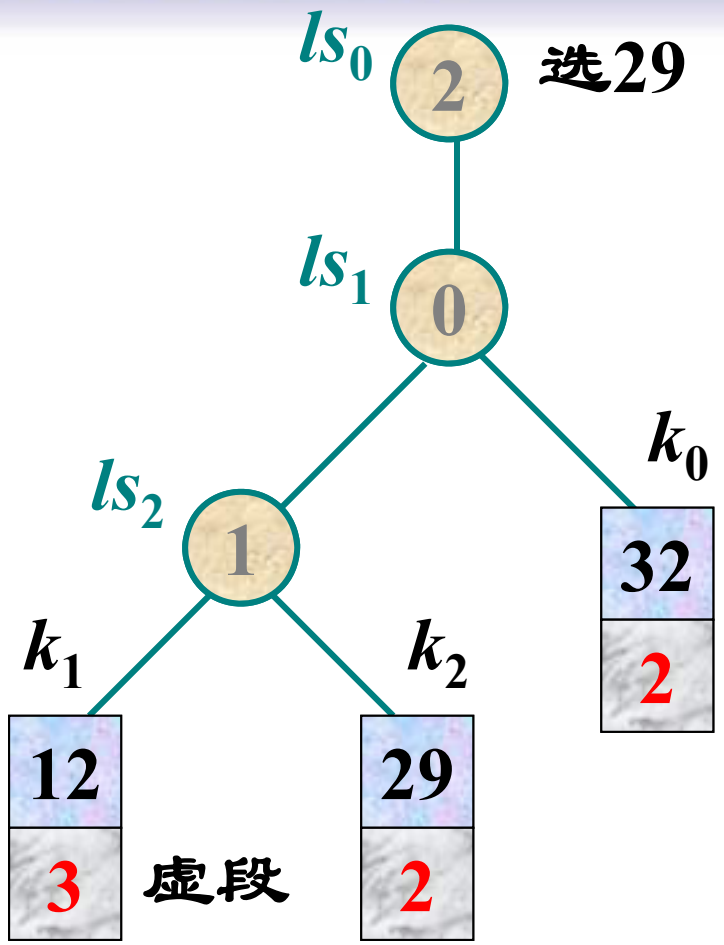
(9) lastKey=56, 置换 k_0, 段号加1, 本段结束

(10) 输出段结束标志, 选择10

{ 17, 21, 05, 44, 10, 12, 56, 32, 29 }

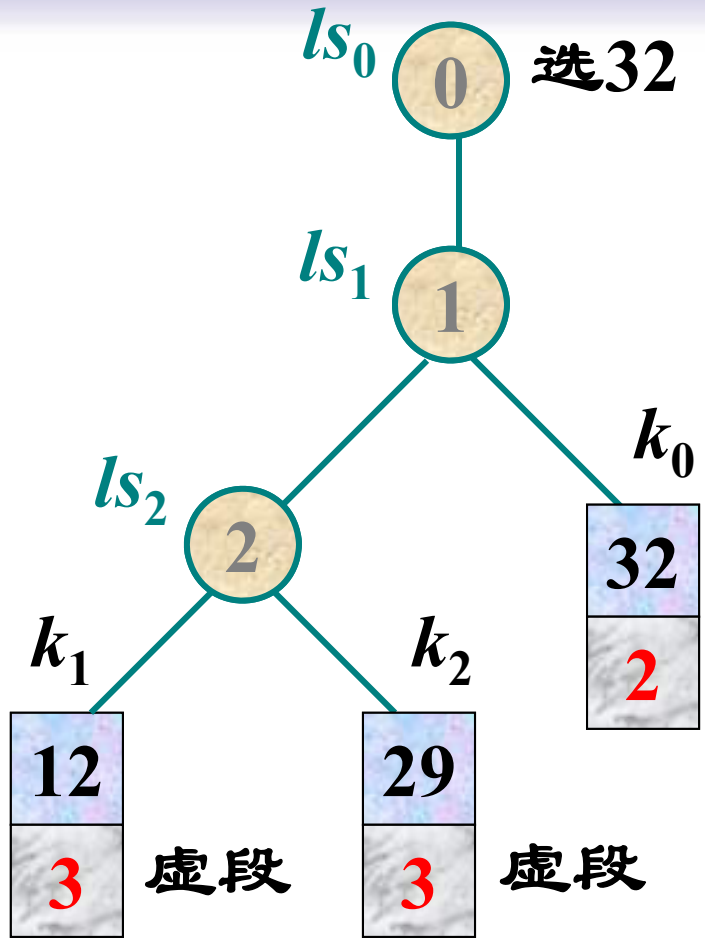


(11) lastKey=10, 置换 k_2, 选择12

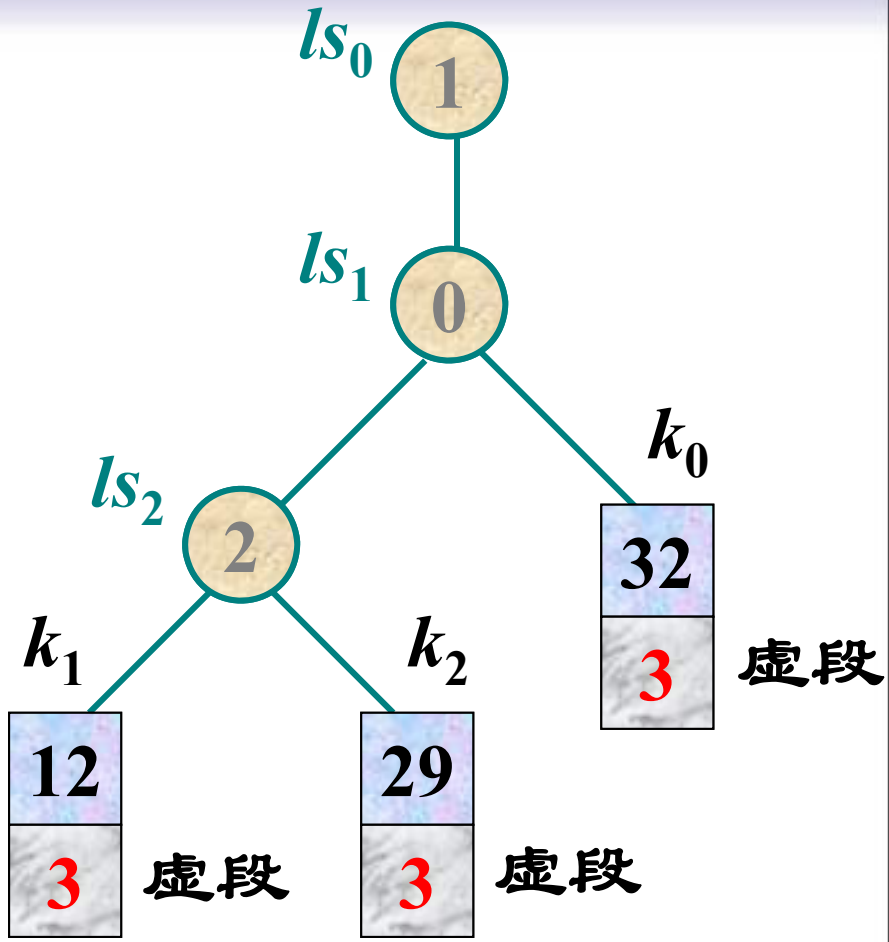


(12) lastKey=12, k_1 置虚段, 选择29

{ 17, 21, 05, 44, 10, 12, 56, 32, 29 }

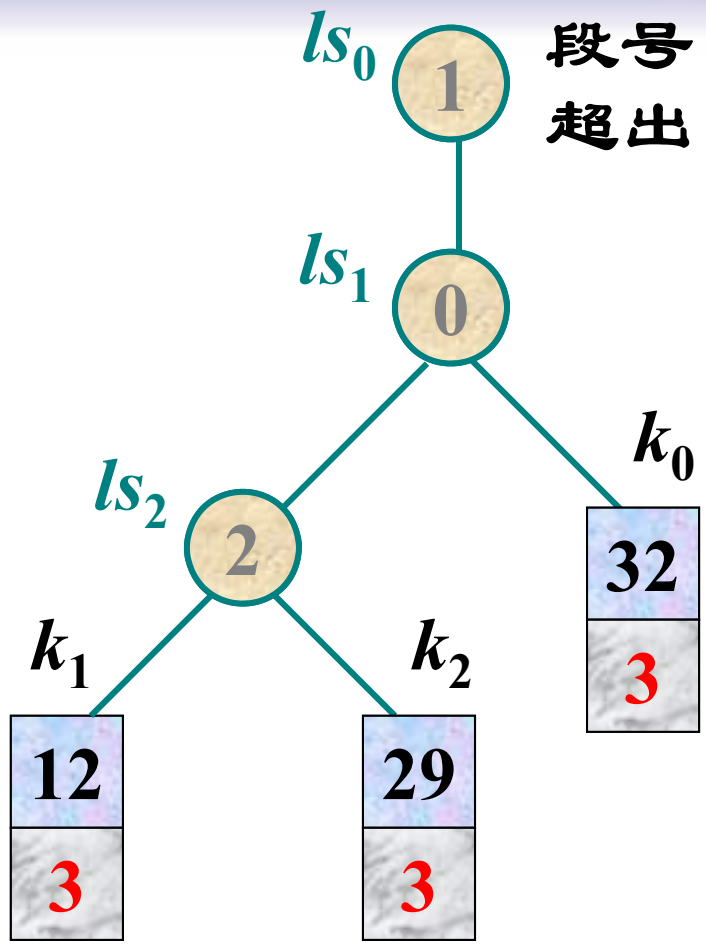


(13) lastKey=29, k_2 置虚段, 选择32



(14) lastKey=32, k_0 置虚段, 本段结束

{ 17, 21, 05, 44, 10, 12, 56, 32, 29 }



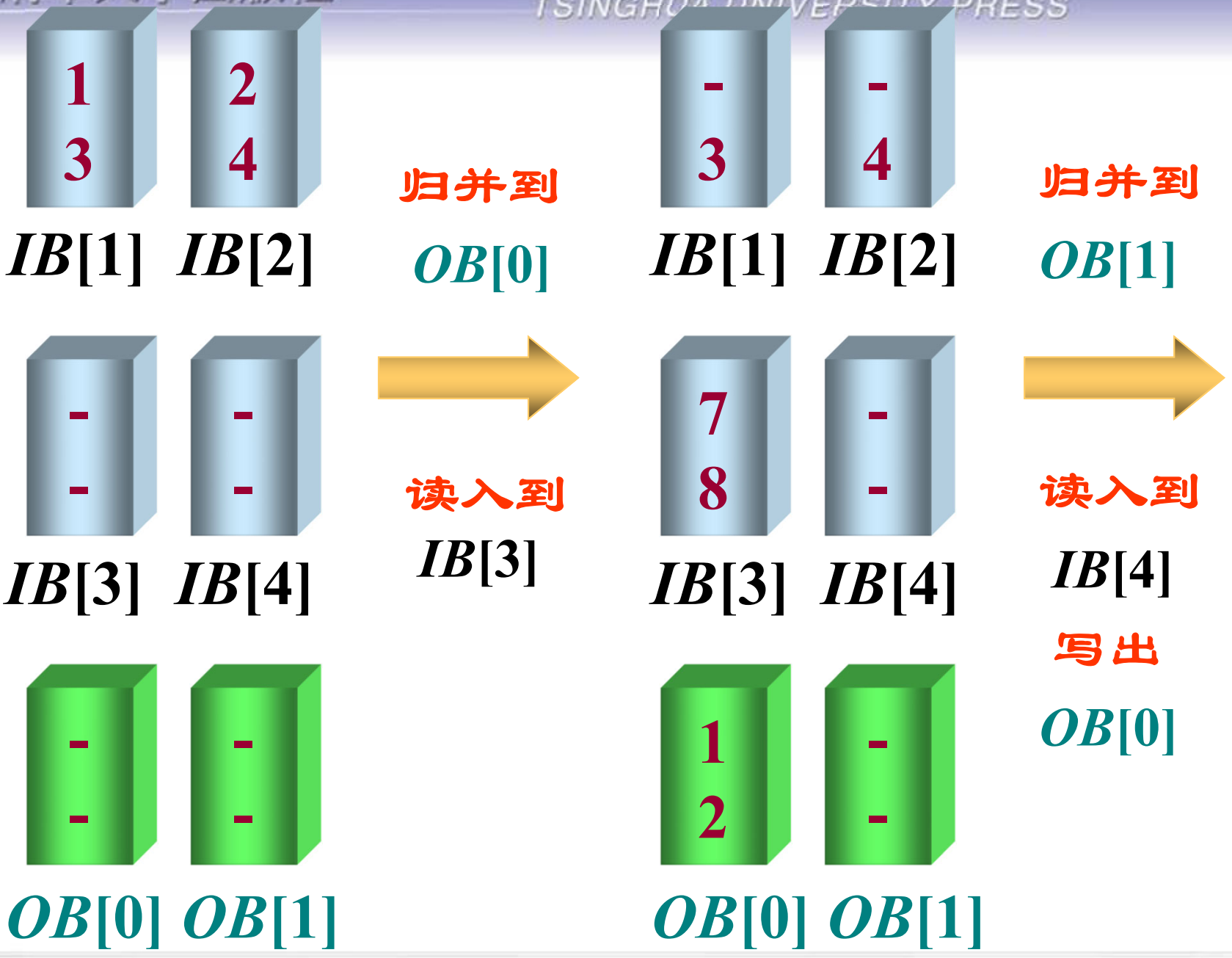
(15) 输出段结束标志，结束

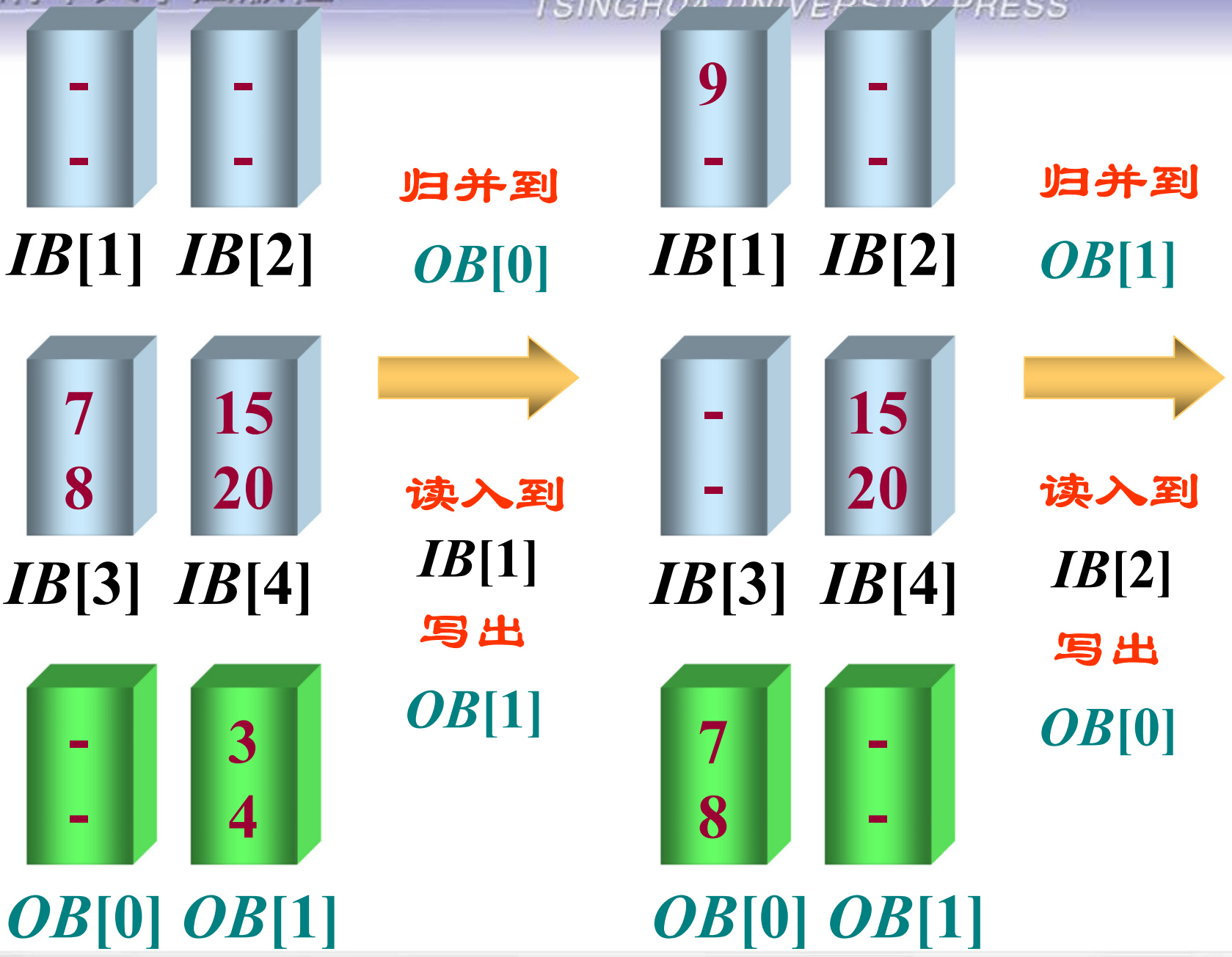
- 当输入的对象序列已经按排序码大小排好序时，只生成一个初始归并段。
- 在一般情况下，若输入文件有 n 个对象，生成初始归并段的时间开销是$O(n\log_2 k)$，因为每输出一个对象，对败者树进行调整需要时间为$O(\log_2 k)$。

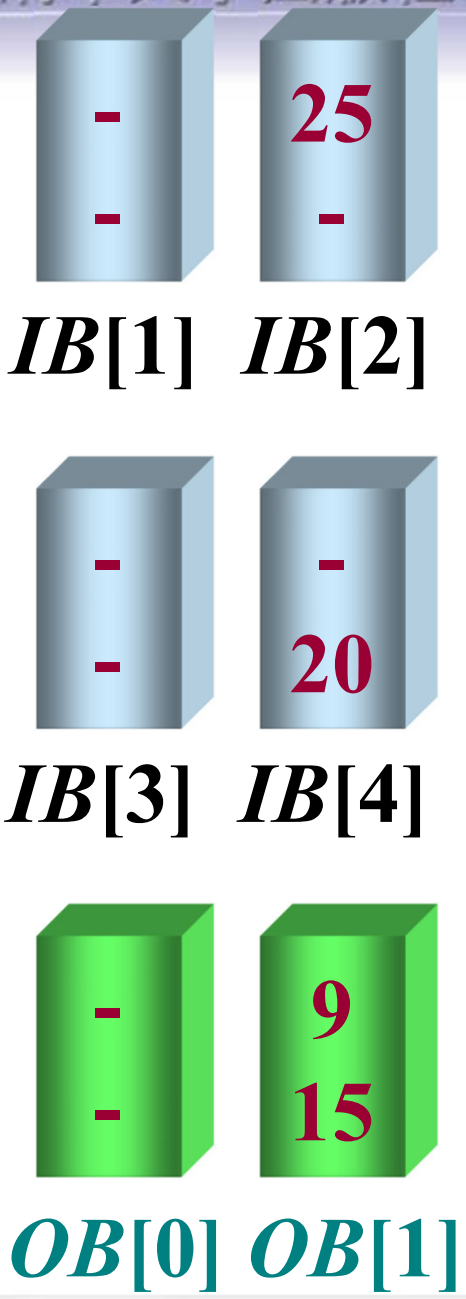
并行操作的缓冲区处理

- 如果采用 k 路归并对 k 个归并段进行归并，至少需要 k 个输入缓冲区和 1 个输出缓冲区。每个缓冲区存放一个页块的信息。
- 但要同时进行输入、内部归并、输出操作，这些缓冲区就不够了。例如，
  - 在输出缓冲区满需要向磁盘写出时，就必须中断内部归并的执行。
  - 在某一输入缓冲区空，需要从磁盘上再输入一个新的页块的对象时，也不得不中断内部归并。

- 由于内外存信息传输的时间与CPU的运行时间相比要长得多，所以使得内部归并经常处于等待状态。
- 为了改变这种状态，希望使输入、内部归并、输出并行进行。对于 k 路归并，必须设置 $2k$ 个输入缓冲区和 2 个输出缓冲区。
- 示例：给每一个归并段固定分配 2 个输入缓冲区，做 2 路归并。假设存在 2 个归并段：
  - Run0：对象的关键码是 1, 3, 7, 8, 9
  - Run1：对象的关键码是 2, 4, 15, 20, 25
- 假设每个缓冲区可容纳 2 个对象。需要设置 4 个输入缓冲区$IB[i]$, $1 \le i \le 4$，2 个输出缓冲区$OB[0]$和$OB[1]$。


1
3
2
4
IB[1] IB[2]
归并到
OB[0]
-
3
-
4
IB[1] IB[2]
归并到
OB[1]
- -
- -
IB[3] IB[4]
读入到
IB[3]
7
8
- -
IB[3] IB[4]
读入到
IB[4]
写出
OB[0]
- -
- -
OB[0] OB[1]
1
2
- -
OB[0] OB[1]


-
-
-
-
IB[1] IB[2]
归并到
OB[0]
9
-
-
-
IB[1] IB[2]
归并到
OB[1]
7
8
15
20
IB[3] IB[4]
读入到
IB[1]
写出
OB[1]
-
-
15
20
IB[3] IB[4]
读入到
IB[2]
写出
OB[0]
-
-
3
4
OB[0] OB[1]
7
8
-
-
OB[0] OB[1]



- 由示例可知，采用 $2k$ 个输入缓冲区和 2 个输出缓冲区，可实现输入、输出和 k 路内部归并的并行操作。
- 这 $2k$ 个输入缓冲区如果平均分配给 k 个归并段，当其中某一 个归并段比其它归并段短很多时，分配给它的缓冲区早早就空闲了。

- 因此，不应为各归并段分别分配固定的两个缓冲区，缓冲区的分配应当是动态的，可根据需要为某一归并段分配缓冲区。但不论何时，每个归并段至少需要一个包含来自该归并段的对象的输入缓冲区。

k 路归并时动态分配缓冲区的实施步骤

为 k 个初始归并段各建立一个缓冲区的**链式队列**，开始时为每个队列先分配一个输入缓冲区。另外建立空闲缓冲区的链式栈，把其余 k 个空闲的缓冲区送入此栈中。输出缓冲区OB定位于0号输出缓冲区。

用 LastKey[i] 存放第 i 个归并段最后输入的关键码，用 NextRun 存放 LastKey[i] 最小的归并段段号；若有几个 LastKey[i] 都 是最小时，将序号最小的 i 存放到 NextRun 中。如果LastKey [NextRun] $\neq \infty$，则从空闲缓冲区栈中取一个空闲缓冲区，预先链入段号为 NextRun 的归并段的缓冲区队列中。

使用函数 kwaymerge 对 k 个输入缓冲区队列中的对象进行 k 路归并，结果送入输出缓冲区OB 中。归并一直持续到输出缓冲区 OB 变满或者有一个关键码为 ∞ 的对象被归并到 OB 中 为止。

如果一个输入缓冲区变空，则 kwaymerge 进到该输入缓冲区队列中的下一个缓冲区，同时将变空的位于队头的缓冲区从队列中退出，加入到空闲缓冲区栈中。

但如果在输出缓冲区变满或关键码为 ∞ 的对象被归并到输出缓冲区*OB*的同时一个输入缓冲区变空，则 kwaymerge 不进到该输入缓冲区队列中的下一个缓冲区，变空的缓冲区也不从队列中退出，归并暂停。

一直等着，直到磁盘输入或磁盘输出完成为止，继续归并。

如果一个输入缓冲区读入完成，将它链入适当归并段的缓冲区队列中。然后确定满足 LastKey [NextRun] 的最小的 NextRun，确定下一步将读入哪一个归并段的对象。

如果 LastKey[NextRun] ≠ ∞，则从空闲缓冲区栈中取一个空闲缓冲区，从段号为 NextRun 的归并段中读入下一块，存到这个空闲缓冲区中。

开始写出输出缓冲区*OB*的对象，再将输出缓冲区定位于1号输出缓冲区。

如果关键码为 ∞ 的对象尚未被归并到输出缓冲区*OB*中，转到　继续操作；否则，一直等待，直到写出完成，然后算法结束。

- 示例：假设对如下三个归并段进行 3 路归并。每个归并段由3块组成, 每块有2个对象。各归并段最后一个对象关键码 为∞。
  - 归并段1 { 20, 25 } { 26, 28 } { 36, ∞}
  - 归并段2 { 23, 29 } { 34, 38 } { 70, ∞}
  - 归并段3 { 24, 28 } { 31, 34 } { 50, ∞}
- 设立6个输入缓冲区，2个输出缓冲区。利用动态缓冲算法，各归并段输入缓冲区队列及输出缓冲区状态的变化如图所示。


归并段1
归并段2
归并段3
输出0
25
26 28
36 ∞
29
24 28
20 23
归并段1
归并段2
归并段3
输出1
26 28
36 ∞
29
28
31 43
24 25
归并段1
归并段2
归并段3
输出0
36 ∞
29
34 38
28
31 43
26 28
归并段1
归并段2
归并段3
输出1
36 ∞
34 38
70 ∞
31 43
28 29

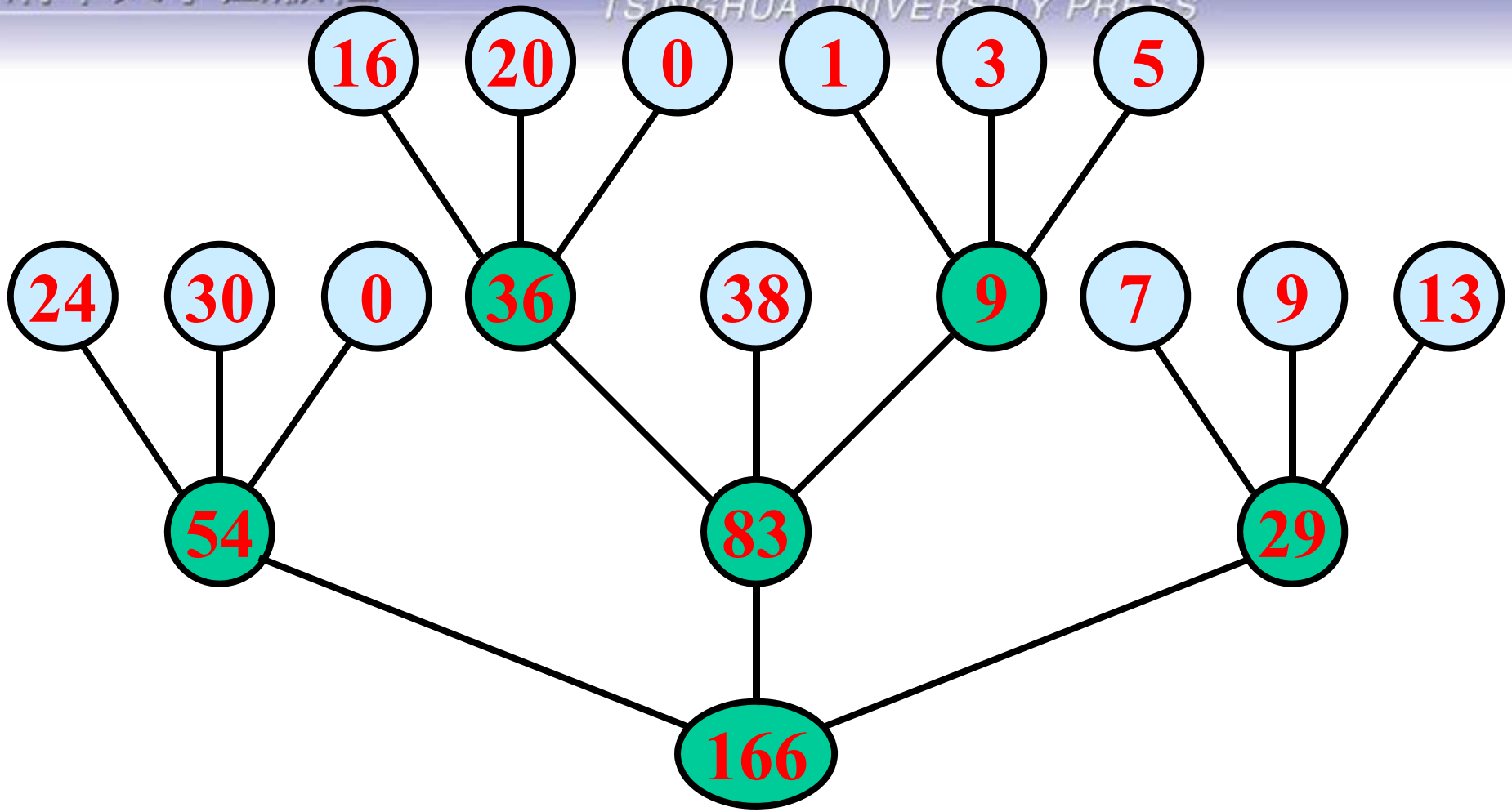

归并段1
归并段2
归并段3
输出0
36 ∞
38
70 ∞
43
50 ∞
31 34
归并段1
归并段2
归并段3
输出1
∞
70 ∞
43
50 ∞
36 38
归并段1
归并段2
归并段3
输出0
∞
70 ∞
∞
43 50
归并段1
归并段2
归并段3
输出1
∞
∞
70 ∞

- 对于较大的k, 为确定哪一个归并段的输入缓冲区最先变空，可对 $LastKey[i]$, $0 \le i \le k-1$，建立一棵败者树。通过 $\log_2 k$ 次比较就可确定哪一 个归并段的缓冲区队列最先变空。
- 对于较大的k，函数 kwaymerge 使用了败者树进行归并。
- 除最初的 k 个输入页块的读入和最后一个输出页块的写出外，其他所有输入，输出和内部归并都是并行执行的。此外，也有可能在 k 个归并段归并完后，需要立即开始对另外 k 个归并段执行归并。所以，在对 k 个归并段进行归并的最后阶段，就开始下一批 k 个归并段的输入。

最佳归并树

- 归并树是描述归并过程的 m 叉树。因为每一次做 m 路归并都需要有 m 个归并段参加，因此，归并树是只有度为0和度为 m 的结点的正则 m 叉树。
- 示例：设有13个长度不等的初始归并段，其长度(对象个数)分别为

  0，0，1，3，5，7，9，13，16，20，24，30，38
- 其中长度为 0 的是空归并段。对它们进行 3 路归并时的归并树如图所示。



此归并树的带权路径长度为

WPL **= (24+30+38+7+9+13)*2+ (16+20+1+3+5)*3**

= 377。

- 在归并树中
  - 各叶结点代表参加归并的各初始归并段
  - 叶结点上的权值即为该初始归并段中的对象个 数
  - 根结点代表最终生成的归并段
  - 叶结点到根结点的路径长度表示在归并过程中的读对象次数
  - 各非叶结点代表归并出来的新归并段
  - 归并树的带权路径长度 *WPL* 即为归并过程中的总读对象数。因而，在归并过程中总的读写对象次数为 $2*WPL = 754$。

- 不同的归并方案所对应的归并树的带权路径长度各不相同。
- 为了使得总的读写次数达到最少, 需要改变归并方案, 重新组织归并树。
- 可将霍夫曼树的思想扩充到 m 叉树的情形。在归并树中，让对象个数少的初始归并段最先归并, 对象个数多的初始归并段最晚归并, 就可建立总读写次数达到最少的最佳归并树。
- 例如, 假设有11个初始归并段, 其长度(对象个数)分别为

  1, 3, 5, 7, 9, 13, 16, 20, 24, 30, 38

  做3路归并。

- 为使归并树成为一棵正则三叉树，可能需要补入空归并段。补空归并段的原则为：
  - 若参加归并的初始归并段有 n 个，做 m 路平衡归并。因归并树是只有度为 0 和度为 m 的结点的正则 m 叉树，设度为 0 的结点有 $n_0(=n)$ 个，度为 m 的结点有 n_m 个，则有

$$n_0 = (m-1)n_m + 1$$

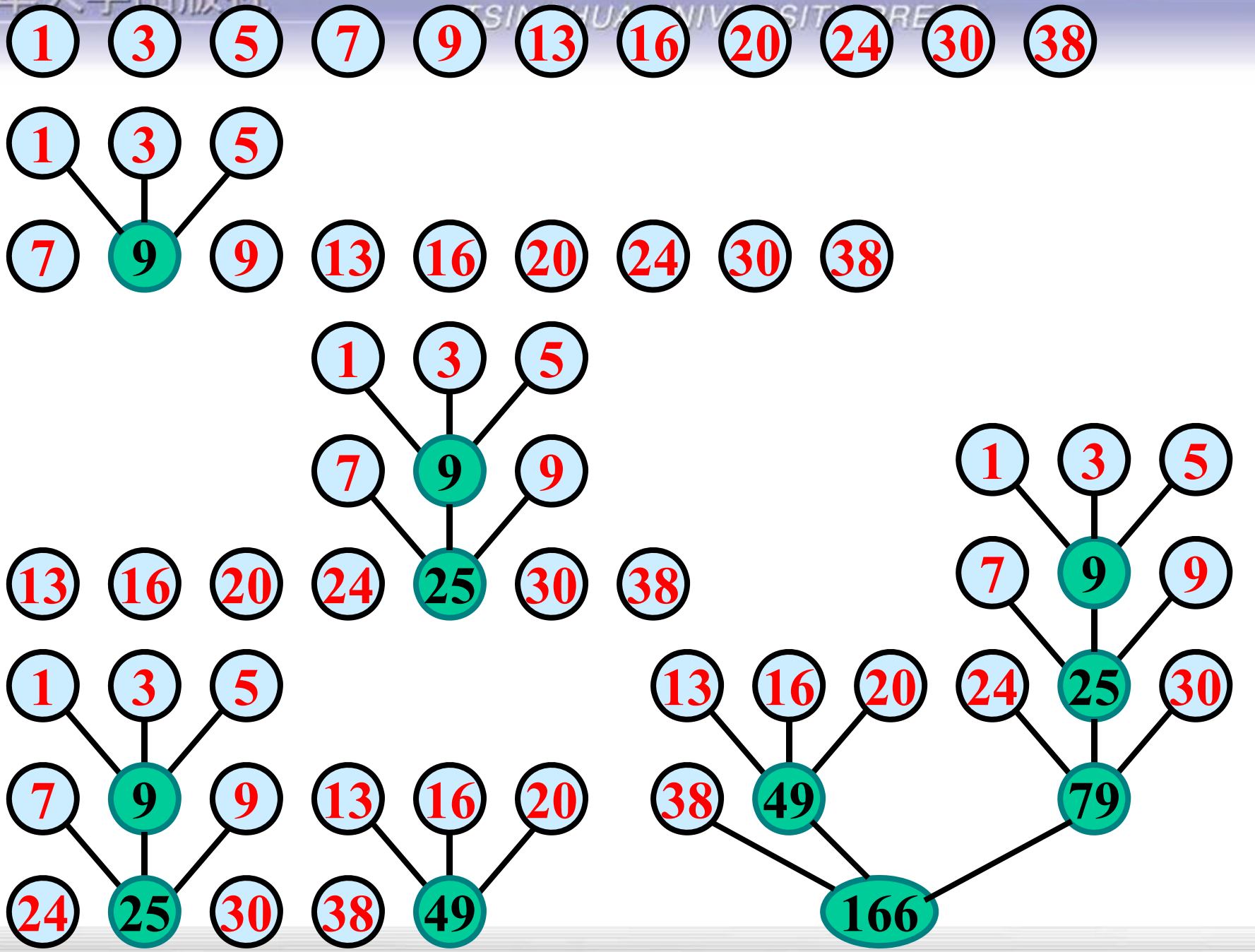
$$n_m = (n_0 - 1) / (m - 1)。$$

  - 若 $(n_0-1) \% (m-1) = 0$，则说明这 n_0 个叶结点正好可以构造 m 叉归并树。
  - 若 $(n_0-1) \% (m-1) = u \neq 0$，则对于这 n_0 个叶结点，其中的 u 个不足以参加 m 路归并。

- 故除了有 n_m 个度为 m 的内结点之外, 还需增加一个内结点。它在归并树中代替了一个叶结点位置, 被代替的叶结点加上刚才多出的 u 个叶结点, 再加上 $m-u-1$ 个对象个数为零的空归并段, 就可建立归并树。
- 在示例中, $n_0 = 11$, $m = 3$, $(11-1) \% (3-1) = 0$, 可以不加空归并段, 直接进行3路归并。

- 它的带权路径长度

$$WPL = 38*1+(13+16+20+24+30)*2+(7+9)*3$$
$$+ (1+3+5)*4 = 328。$$


1 3 5 7 9 13 16 20 24 30 38
1 3 5
7 9 9
13 16 20 24 30 38
1 3 5
7 9 9
13 16 20 24 25 30 38
1 3 5
7 9 9
24 25 30
1 3 5
7 9 9
24 25 30
13 16 20
38 49
13 16 20
38 49
166
1 3 5
7 9 9
24 25 30
79

- 如果做5路归并，让 $m = 5$，则有

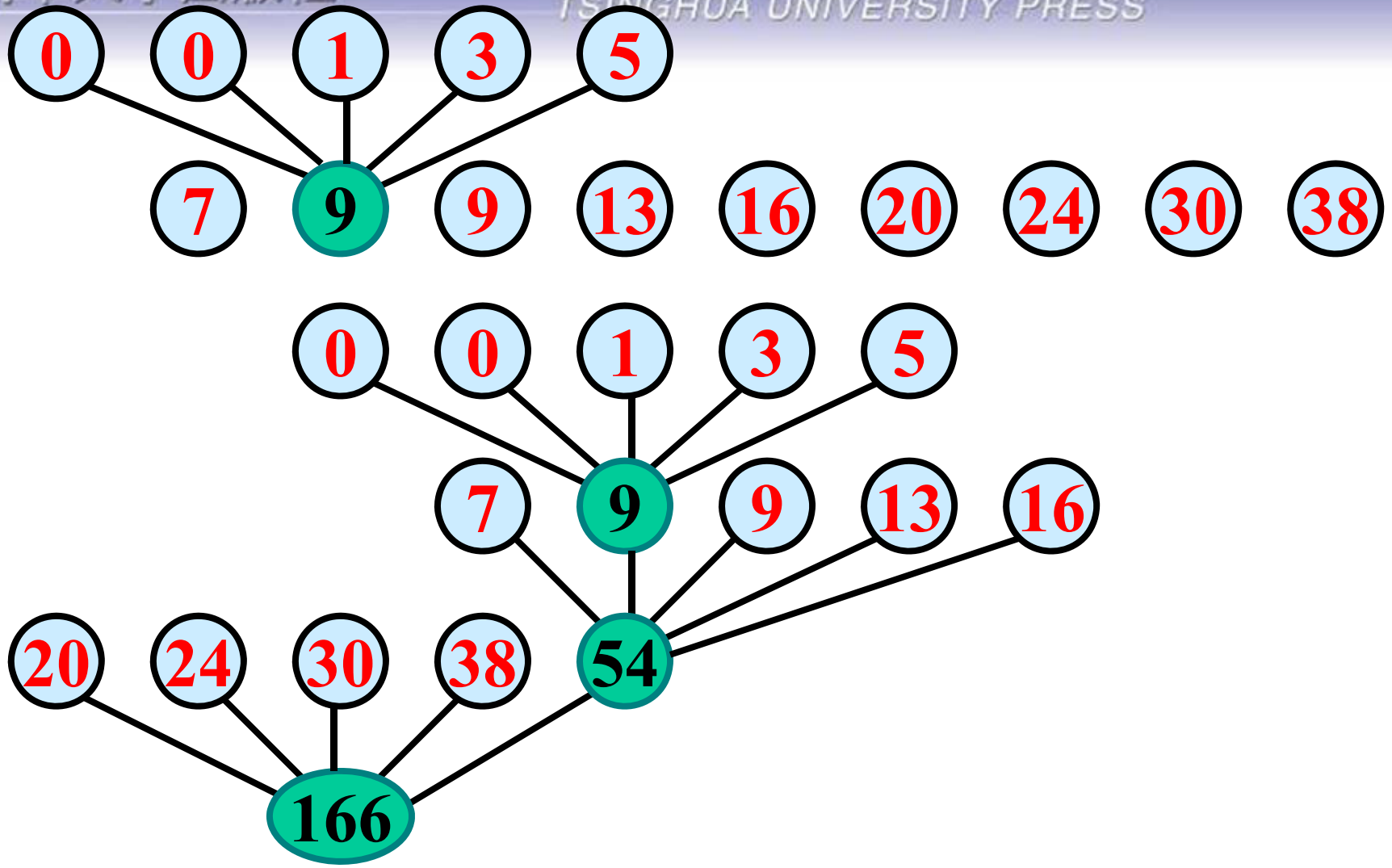
  $(11-1) / (5-1) = 2$

- 表示有2个度为5的内结点；但是，

  $u = (11-1) \% (5-1) = 2 \neq 0$

- 需要加一个内结点,它在归并树中代替了一 个叶结点的位置, 故一个叶结点参加这个内结点下的归并，需要增加的空初始归并段数为 $m-u-1 = 5-2-1 = 2$

- 应当补充 2 个空归并段。则归并树如图所示。



带权路径长度 $WPL = (1+3+5)*3+(7+9+13+16)*2+(20+24+30+38) = 229$