数据结构与算法设计

课程内容

课程内容：数据结构部分

概述	数组与广义表	图	外部排序
线性表	串	查找	
栈与队列	树	内部排序	

课程内容：算法设计部分

概述	贪心	……	计算模型
分治	回溯	……	可计算理论
动态规划	……	……	计算复杂性

Contents

本章内容

- 外部排序的概念
- 多路平衡归并排序方法
- 置换-选择排序
- 最佳归并树

10.1 外部排序的概念和方法

外部排序：基于外部存储设备（或文件）的排序技术就是外部排序。

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。

在排序过程中必须不断地在内存与外存之间传送数据。

外部存储设备：磁带、磁盘

外部排序方法

基于磁盘进行的排序多使用归并排序方法。

排序过程主要分为两个阶段：

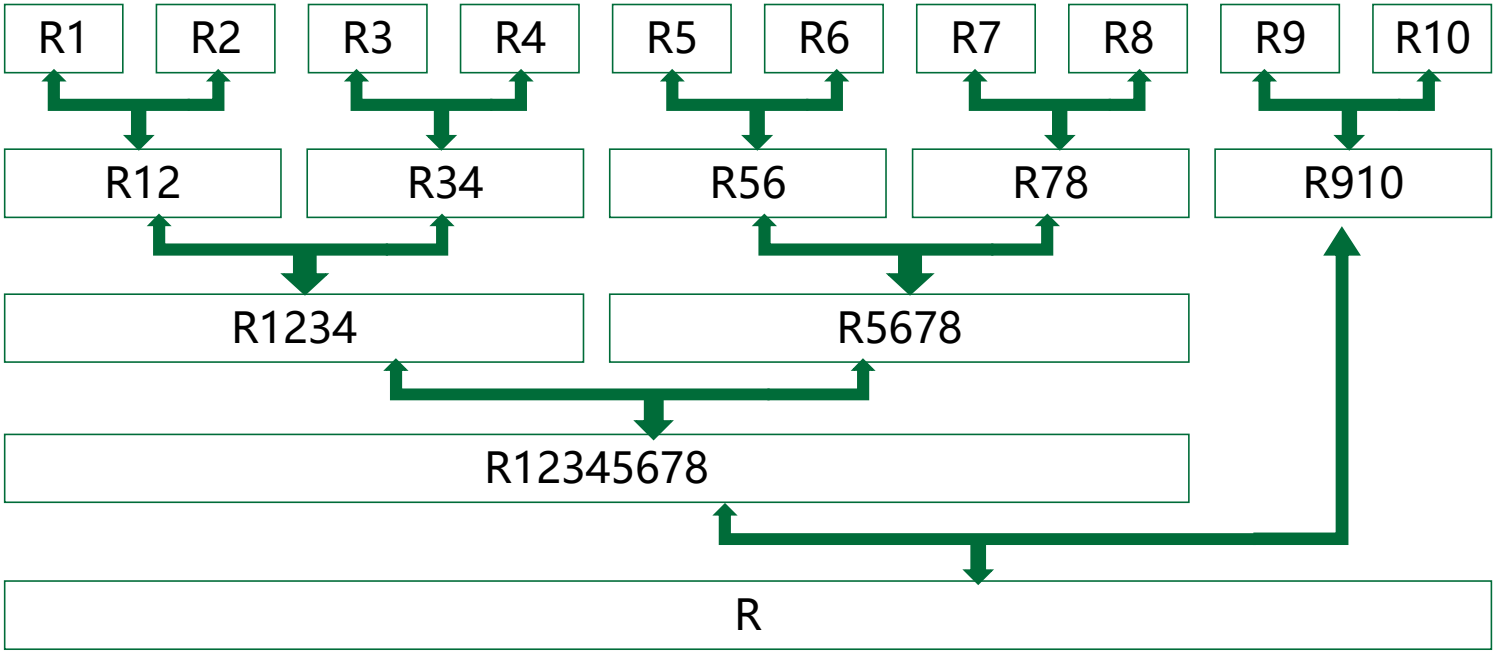
（1）建立用于外排序的内存缓冲区。根据它们的大小将输入文件划分为若干段，用某种内排序方法对各段进行排序。这些经过排序的段叫做初始归并段或初始顺串 (Run)。当它们生成排序片段后就被写到外存中去。

（2）仿照内排序中所介绍过的归并模式，把第一阶段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。

假设有u=10000个记录，分为m=10个段，每段有1000个记录。

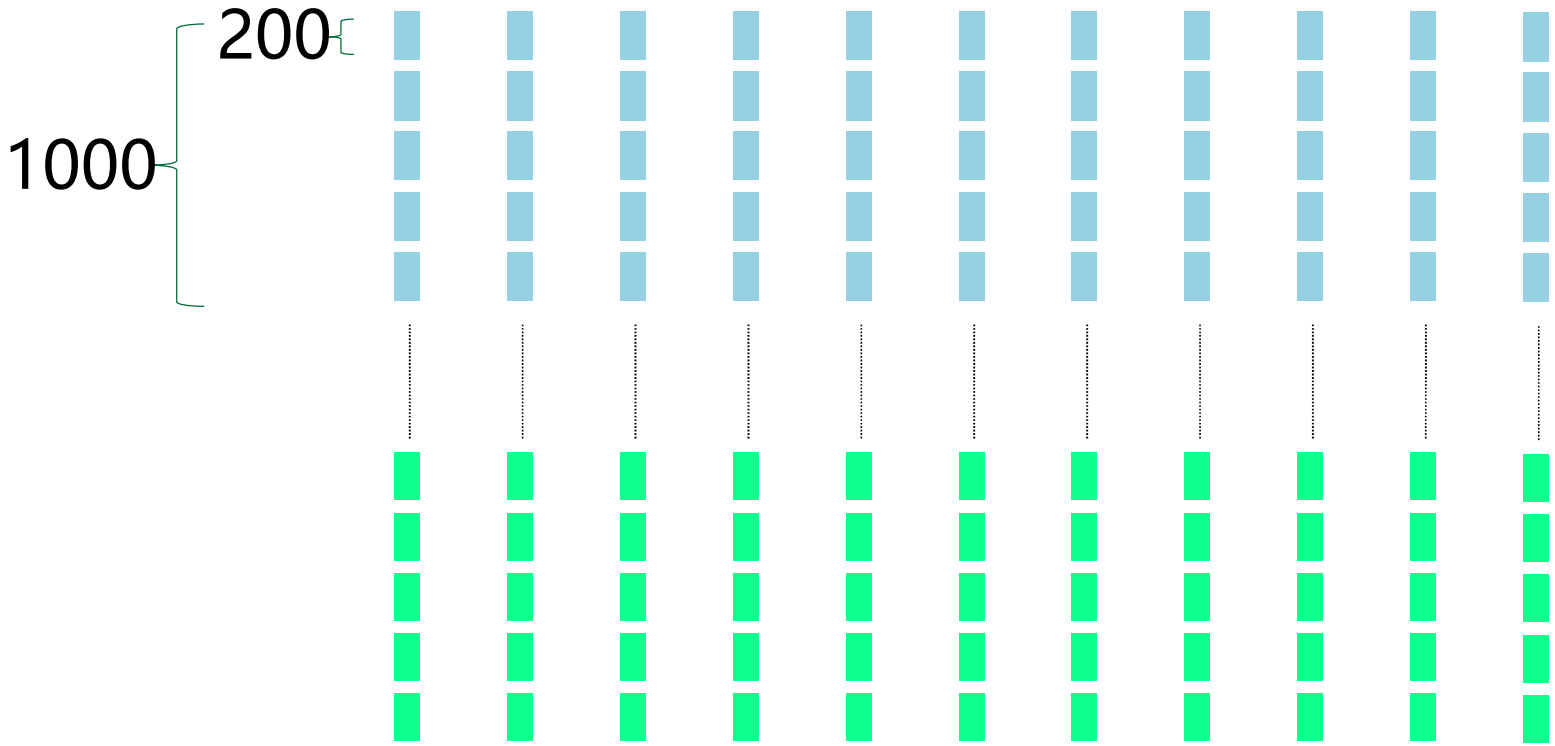
进行k=2路归并，则需要进行s=4趟归并。

总归并趟数s等于归并树的高度$\lceil \log_2 m \rceil$



假设有u=10000个记录分为m=10个段，每段有1000个记录。

假设磁盘每个物理块可容纳200个记录，10个初始段排序，共100次读写操作。

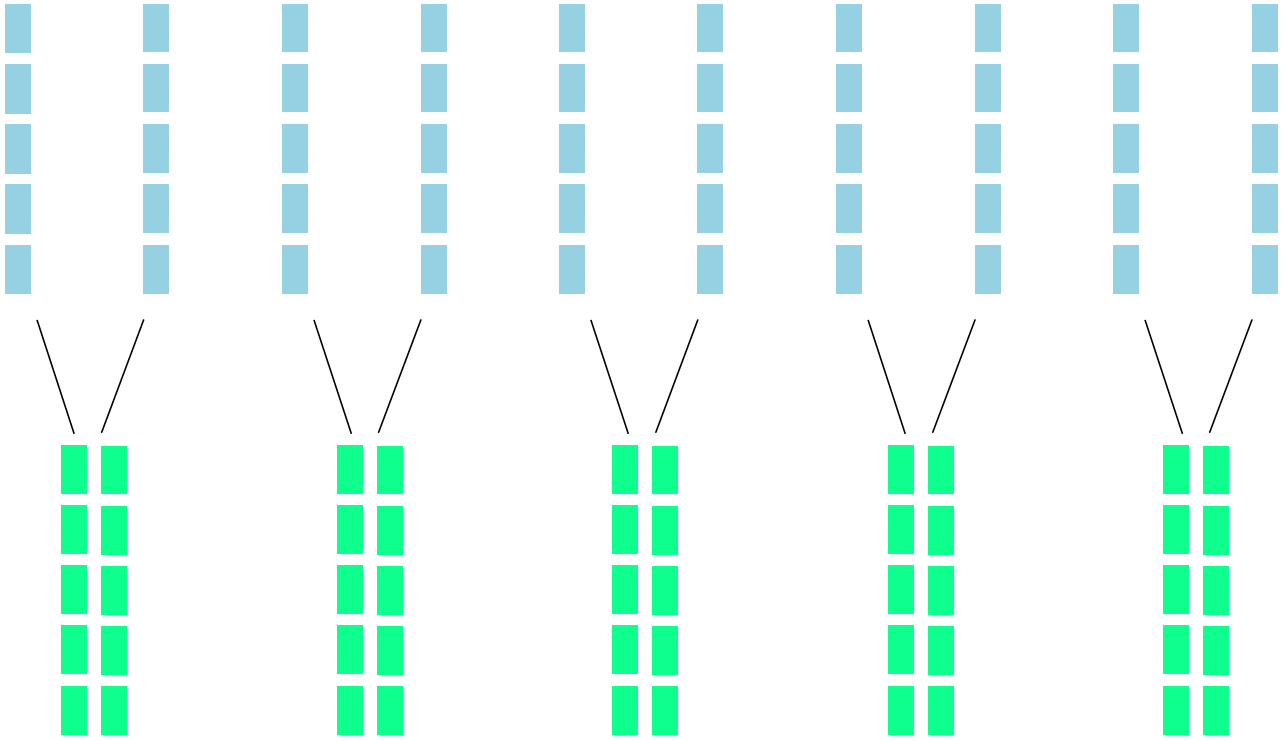


初始化

2路平衡归并树：10个初始块排序

假设有u=10000个记录分为m=10个段，每段有1000个记录。

磁盘每个物理块可容纳200个记录，归并第1趟需要50次读50次写，共100次操作。

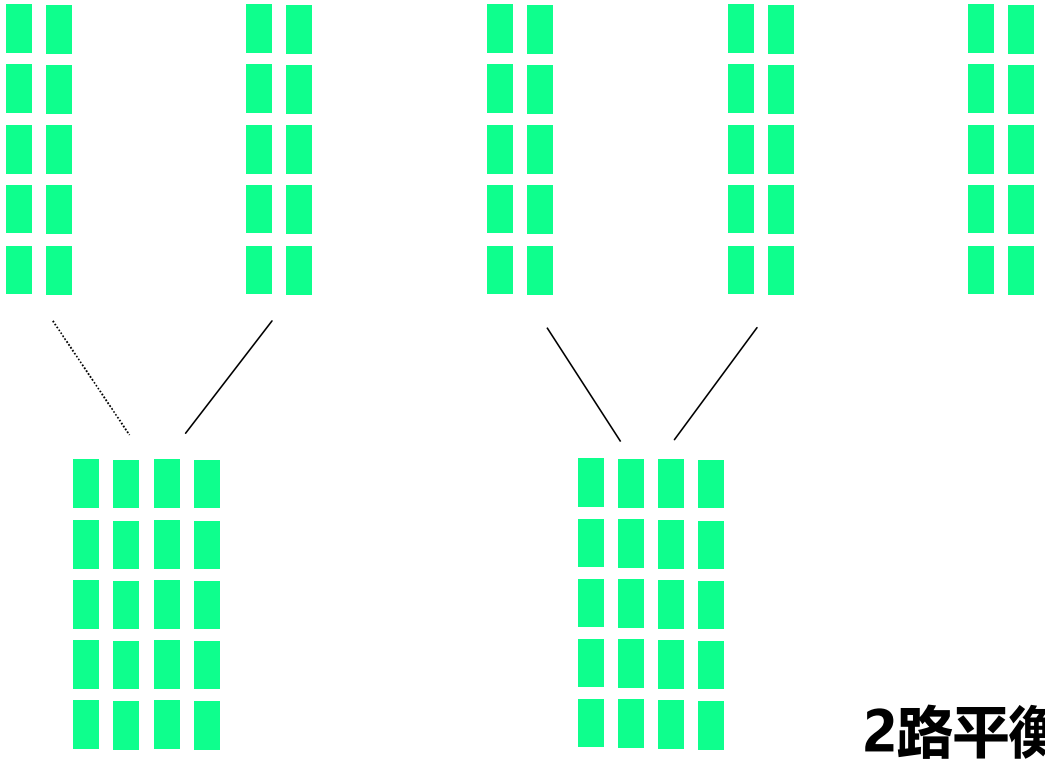


第1趟归并

2路平衡归并树：10个初始块归并为5个

假设有u=10000个记录分为m=10个段，每段有1000个记录。

磁盘每个物理块可容纳200个记录，归并第2趟需要40次读40次写，共80次操作。

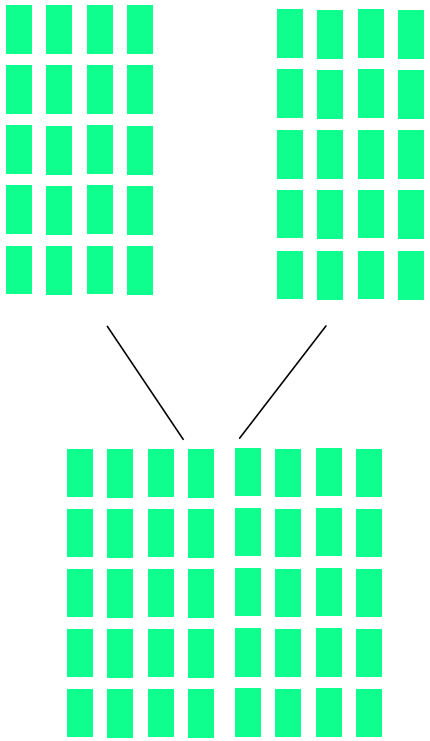


第2趟归并

2路平衡归并树：5个初始块归并为3个

假设有u=10000个记录分为m=10个段，每段有1000个记录。

磁盘每个物理块可容纳200个记录，归并第3趟需要40次读40次写，共80次操作。

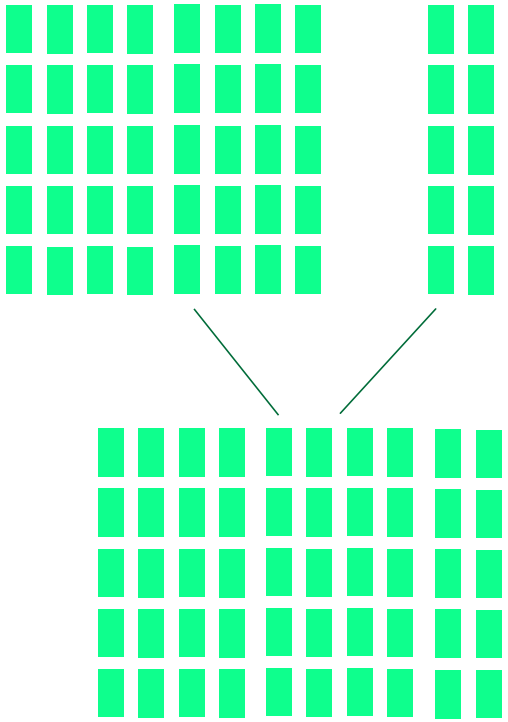


第3趟归并

2路平衡归并树：3个初始块归并为2个

假设有u=10000个记录分为m=10个段，每段有1000个记录。

磁盘每个物理块可容纳200个记录，归并第4趟需要50次读50次写，共100次操作。



第4趟归并

2路平衡归并树：2个初始块归并为1个

假设有u=10000个记录分为m=10个段，每段有1000个记录。

磁盘每个物理块可容纳200个记录，

全部排序共需要读写次数为：d=100+（100+80+80+100）=460。

外部排序需要的总时间为：

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

- $m * t_{IS}$：内部排序所需总时间
- $d * t_{IO}$：外存读写所需总时间
- $s * u * t_{mg}$：内部归并所需总时间

因为 $t_{IO} >> t_{mg}$，要提高外排序速度，应减少读写次数 d.

则上例中 $t_{ES} = 10 * t_{IS} + 460 * t_{IO} + 36000 * t_{mg}$

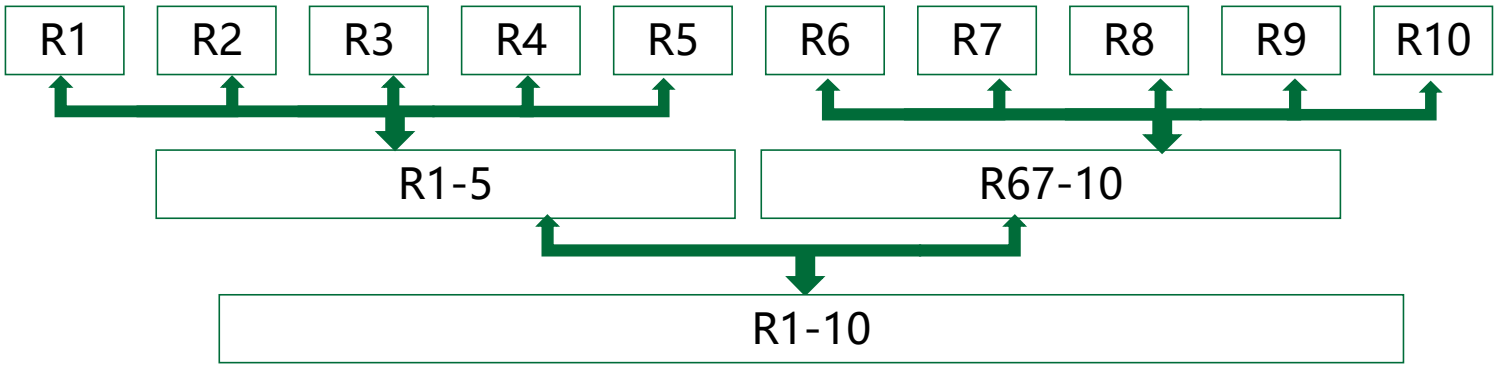
假设有u=10000个记录分为m=10个段，每段有1000个记录。

因为 $t_{IO} >> t_{mg}$，要提高外排速度，应减少读写次数 d.

外存读写次数d和归并过程的关系：

归并路数 k	总读写磁盘次数 d	归并趟数s
2	460	4
5	300	2
10	200	1

S越小，d越小



为了减小外存读写次数**d**，**应该减小**总归并趟数**s**。

$s = \lceil \log_k m \rceil$

减小<u>总读写次数 d</u>的途径：

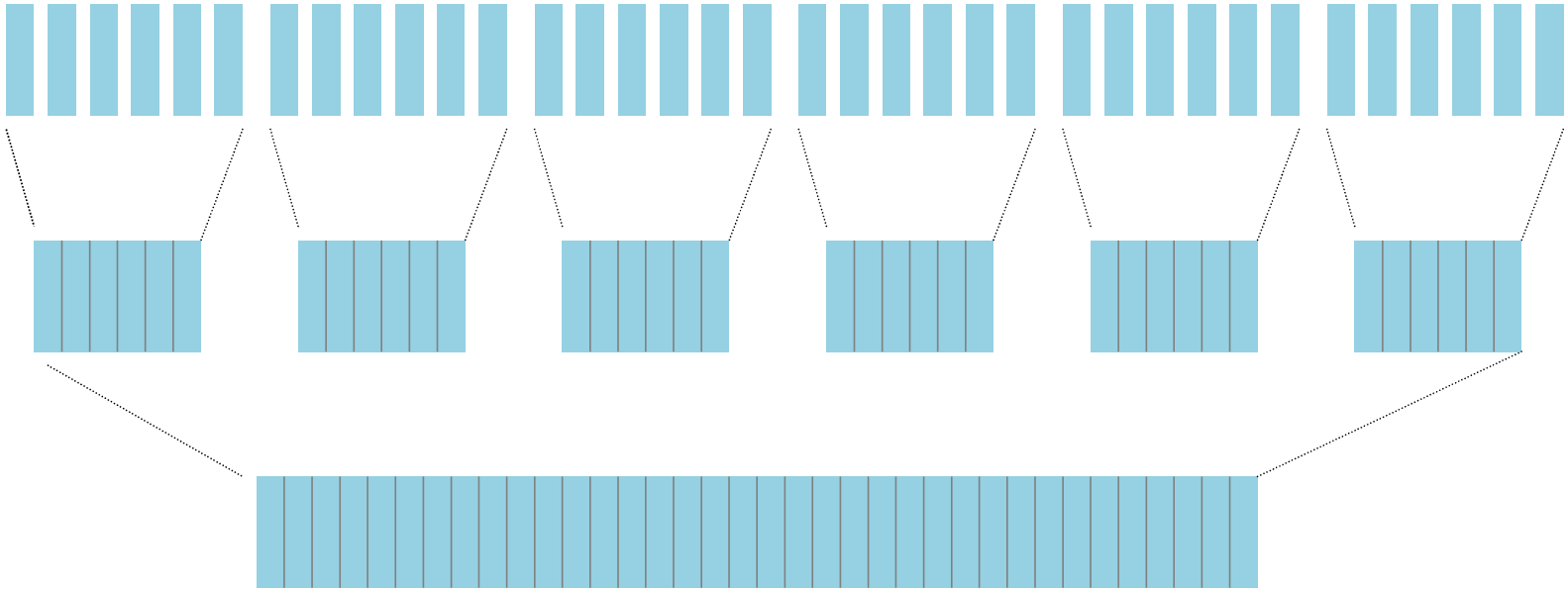
- **增加<u>归并路数k</u>**
- **减小<u>初始段数m</u>。**

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

内部排序所需总时间　　**外存读写所需总时间**　　**内部归并所需总时间**

10.2 多路平衡归并排序 k-way Balanced merging

对于k路平衡归并，如果有 m 个初始归并段，需要归并$s = \lceil \log_k m \rceil$趟。



6路平衡归并树：36个初始归并段

减小总读写磁盘次数 d的途径1：增加归并路数k

10.2 多路平衡归并排序

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 $(k*m)$ 个对象需要做$(n-1)*(k-1)$次比较

则s趟归并总共需要的比较次数为：

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

$$= \frac{logm}{logk}(k-1)(n-1)$$

$$= \frac{(k-1)}{logk}(logm)(n-1)$$

随k增长而增长，
增大k，会使得归并的时间增大

办法：减小k个对象中选最小的比较次数

减小总读写磁盘次数 d的途径1：增加归并路数k

败者树

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

则s趟归并总共需要的比较次数：

$$\frac{(k-1)}{\log_2 k}\log_2 m(n-1)$$

$$\frac{\log_2 k}{\log_2 k}\log_2 m(n-1) = \log_2 m(n-1)$$

利用败者树，只要内存空间允许，增大归并路数 k，将有效地减少归并树深度，从而减少读写磁盘次数 d，提高外排序的速度。

败者树

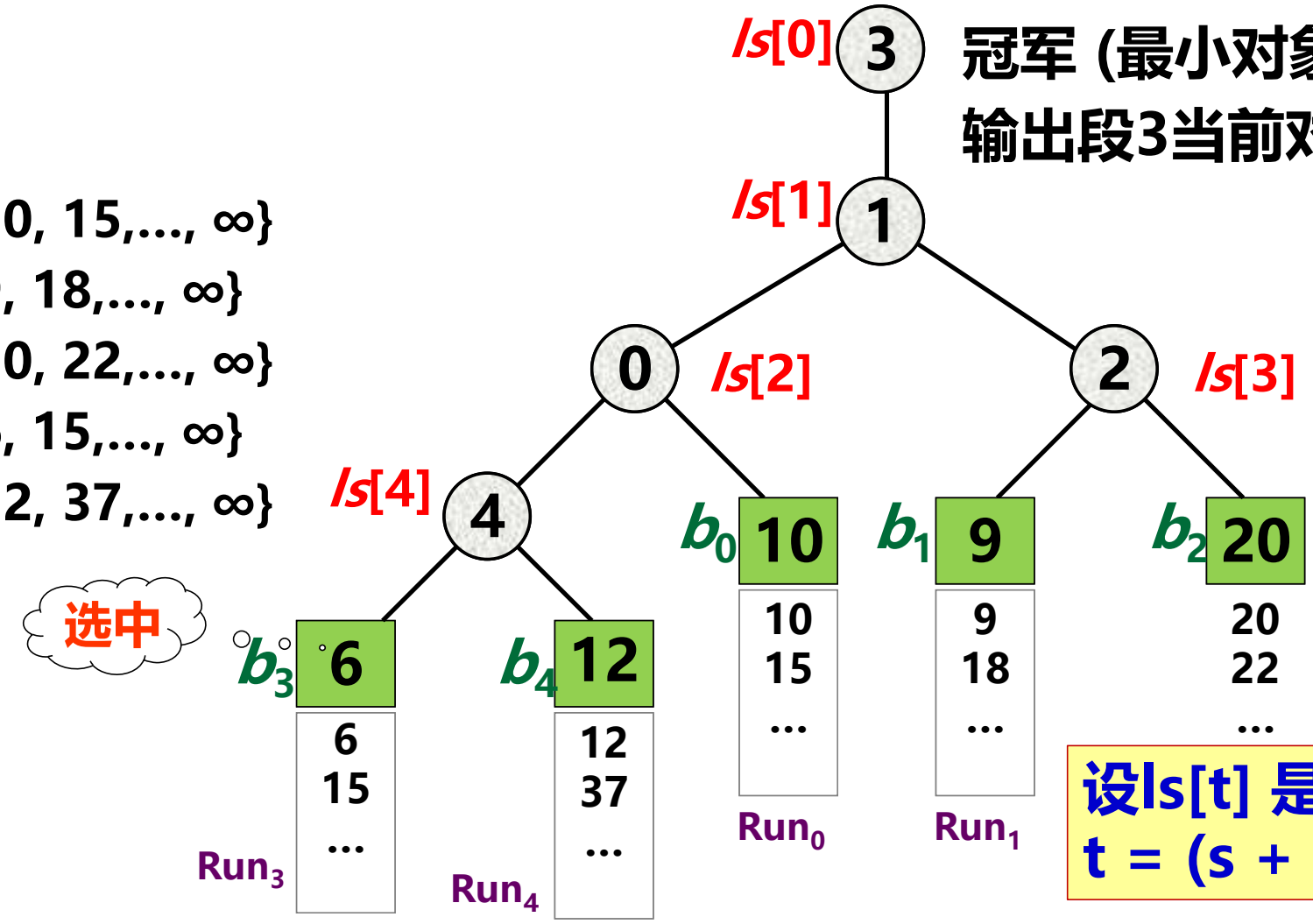
败者树是一棵**正则的完全二叉树**，其中

- 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**关键码大**的结点(即败者)；

```
typedef int LoserTree[k];

typedef struct{
    KeyType key;
} ExNode, External[k+1];
```

Run0: {10, 15,..., ∞}
Run1: {9, 18,..., ∞}
Run2: {20, 22,..., ∞}
Run3: {6, 15,..., ∞}
Run4: {12, 37,..., ∞}

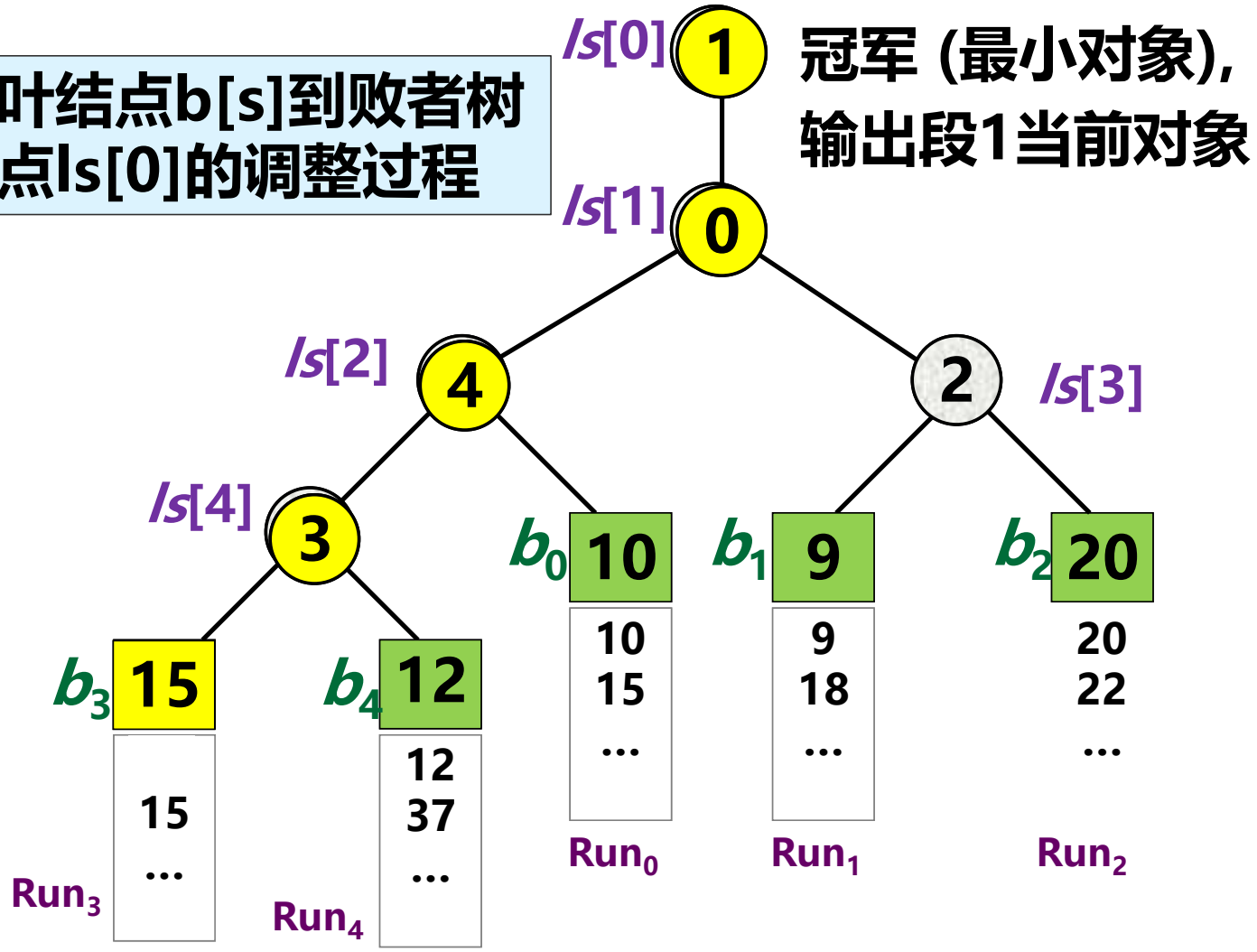


设ls[t] 是b[s]的父节点
t = (s + k) / 2

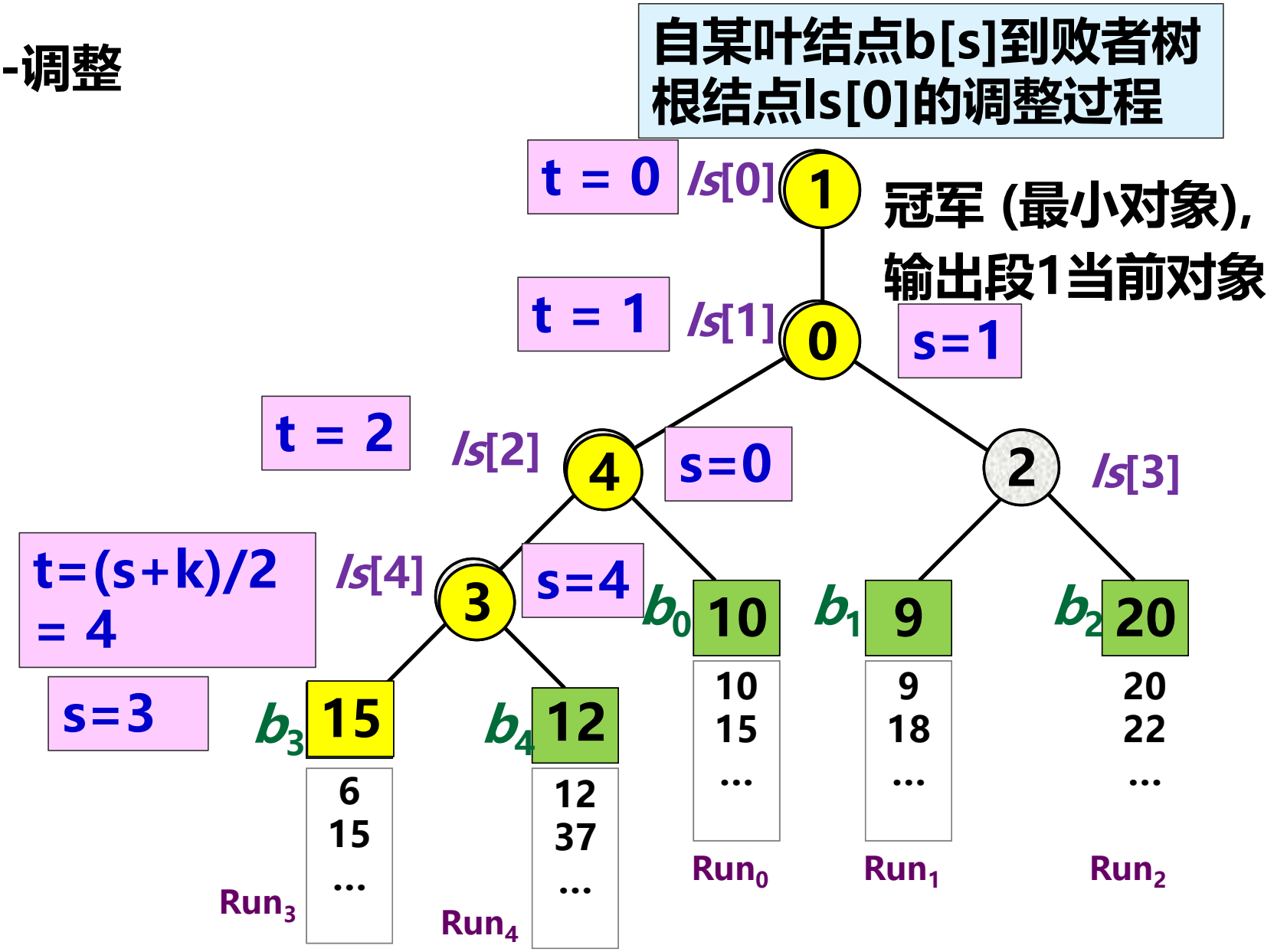
LoserTree	ls[0]	ls[1]	ls[2]	ls[3]	ls[4]
	3	1	0	2	4

败者树-调整

自某叶结点b[s]到败者树根结点ls[0]的调整过程



败者树-调整


自某叶结点b[s]到败者树根结点ls[0]的调整过程
t = 0
ls[0]
1
冠军 (最小对象),
输出段1当前对象
t = 1
ls[1]
0
s=1
t = 2
ls[2]
4
s=0
2
ls[3]
t=(s+k)/2
= 4
ls[4]
3
s=4
b0
10
b1
9
b2
20
s=3
b3
15
b4
12
10
15
…
9
18
…
20
22
…
6
15
…
12
37
…
Run3
Run4
Run0
Run1
Run2

败者树-调整

```
//自某叶结点b[s]到败者树根结点ls[0]的调整算法
void adjust ( LoserTree &ls, int s )
{//从叶结点b[s]开始，依次将当前的b[s]与父结点指示的失败者进行比较
//将失败者所在归并段的段号记入父节点中。
    t = (s + k) / 2;  // ls[t] 是b[s]的父节点
    while(t > 0) {
          if (b[s].key > b[ls[t]].key)  {s ←赢家;  ls[t] ←输家;}
          t = t/2;
    }
    ls[0] = s;
}// adjust
```

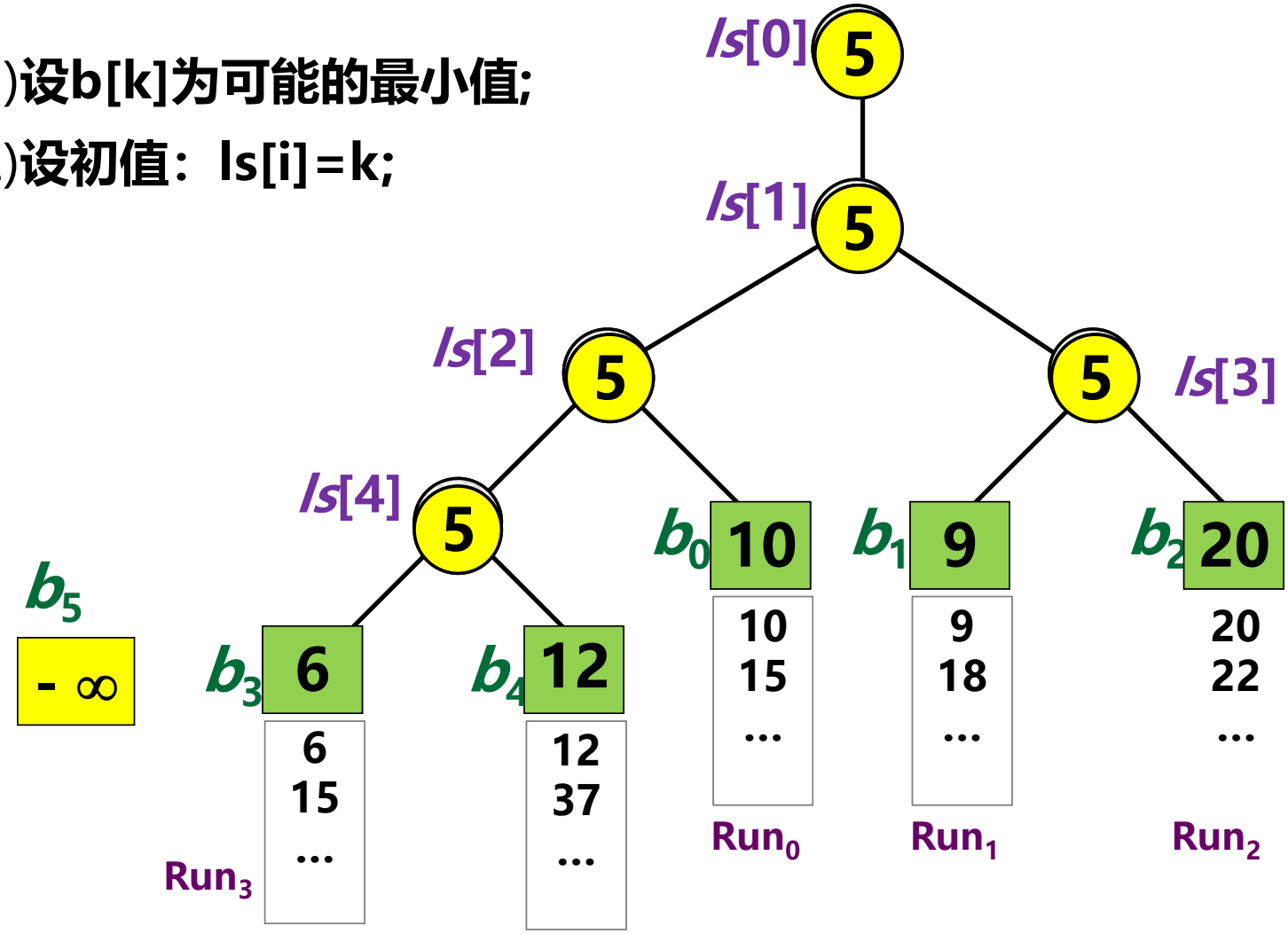
败者树的高度为$\lceil \log_2 k \rceil$，在每次调整，找下一 个具有最小关键码对象时，最多做$\lceil \log_2 k \rceil$次关键码比较。

败者树-创建

初始化败者树

1)设b[k]为可能的最小值;

2)设初值：ls[i]=k;



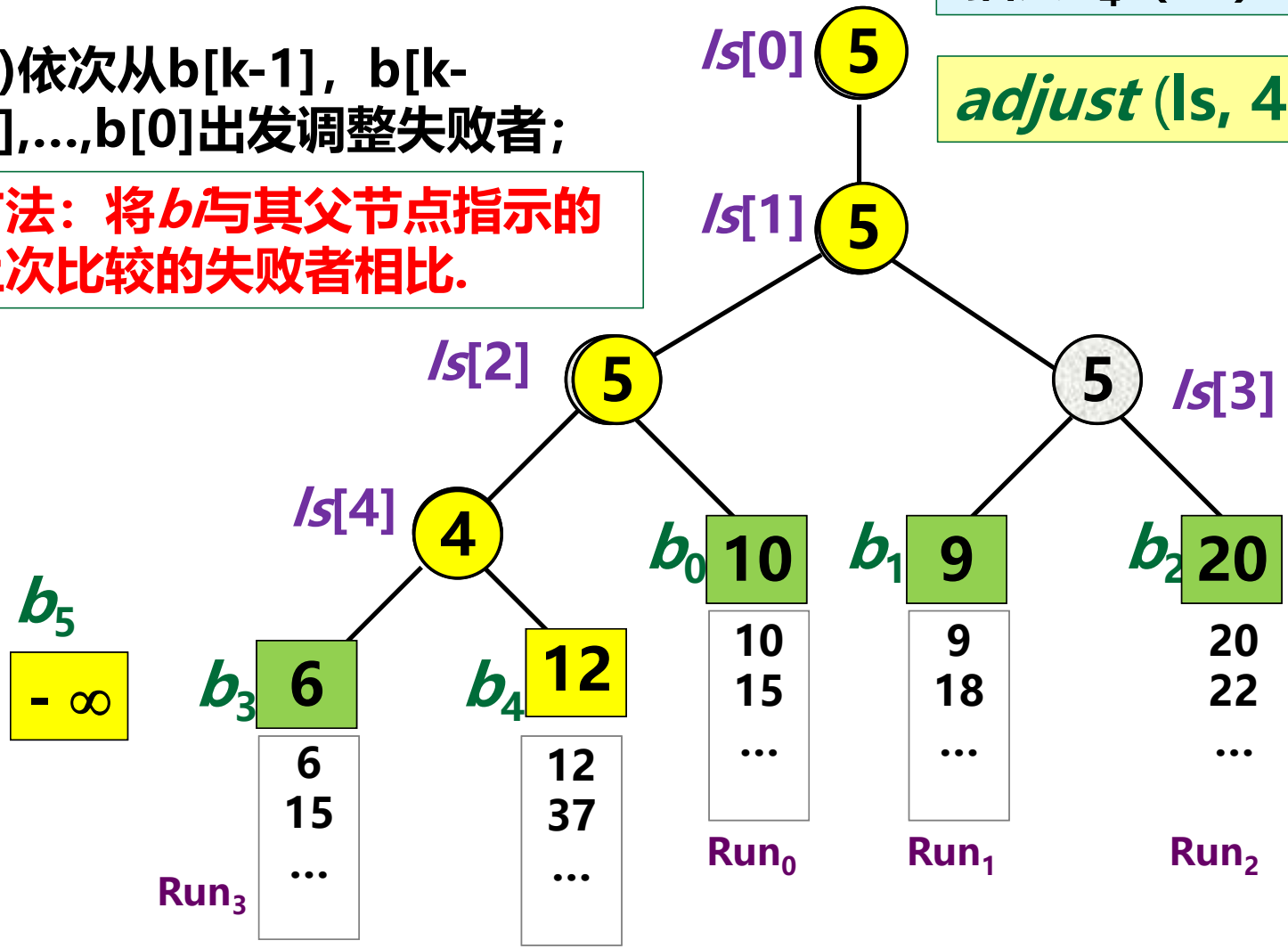
败者树-创建

创建败者树：
插入b_4（12）

3)依次从b[k-1]，b[k-2],...,b[0]出发调整失败者；

方法：将*b*与其父节点指示的上次比较的失败者相比.

adjust (ls, 4)



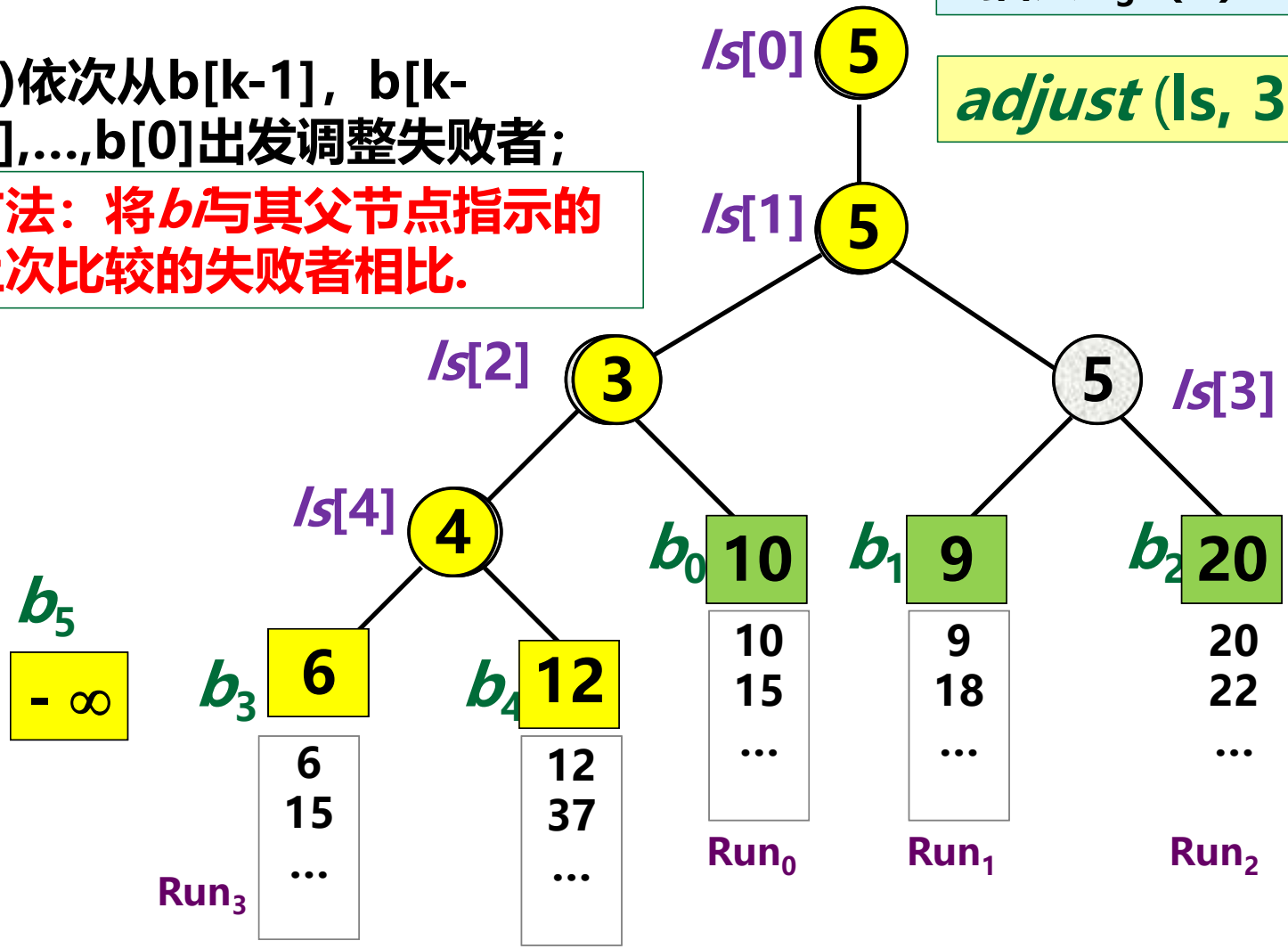
败者树-创建

创建败者树：
插入b_3（6）

adjust (ls, 3)

3)依次从b[k-1]，b[k-2],...,b[0]出发调整失败者；

方法：将*b*i与其父节点指示的上次比较的失败者相比.



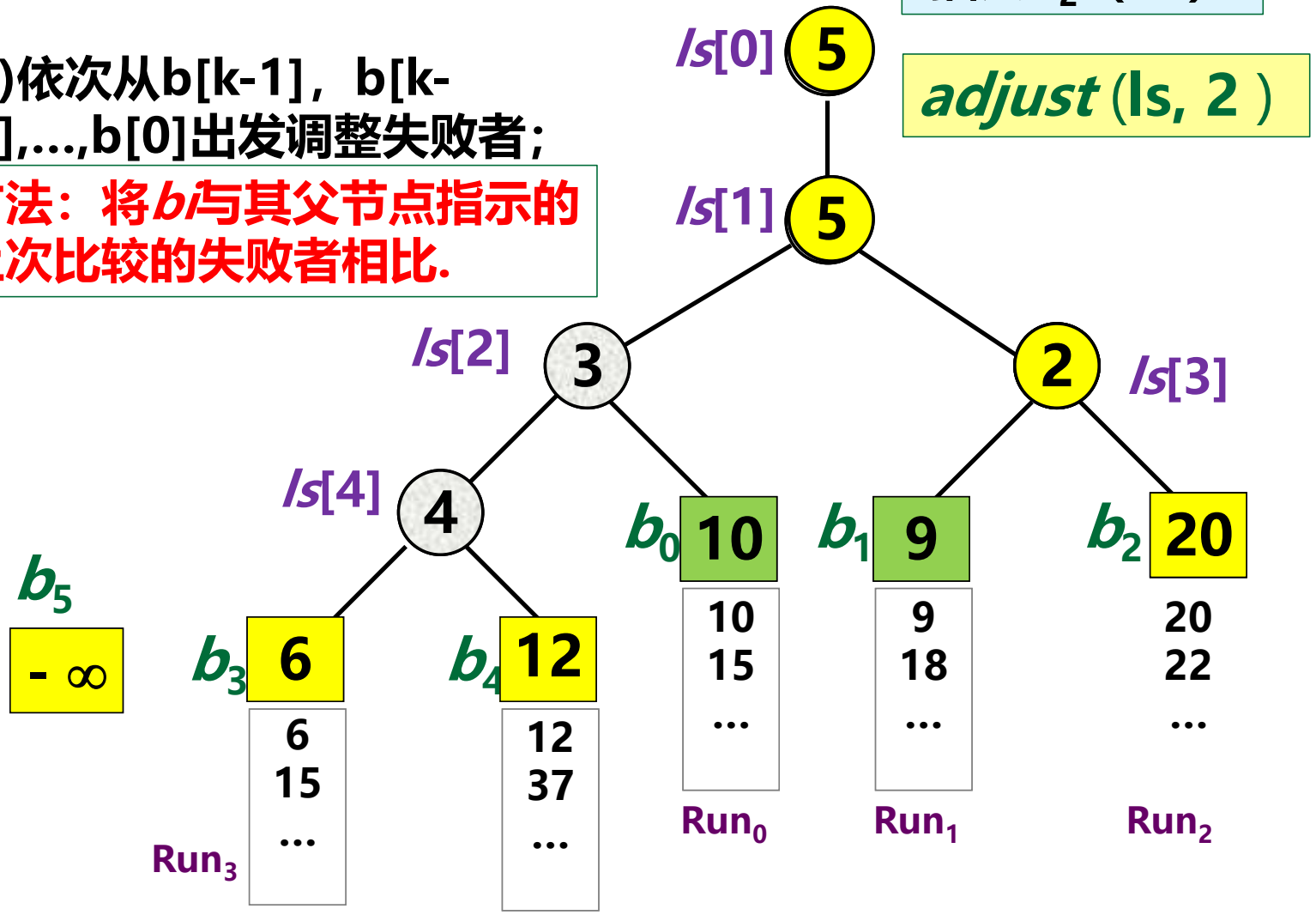
败者树-创建

创建败者树：
插入b_2（20）

adjust (ls, 2)

3)依次从b[k-1]，b[k-2],...,b[0]出发调整失败者；

方法：将*b*i与其父节点指示的上次比较的失败者相比.



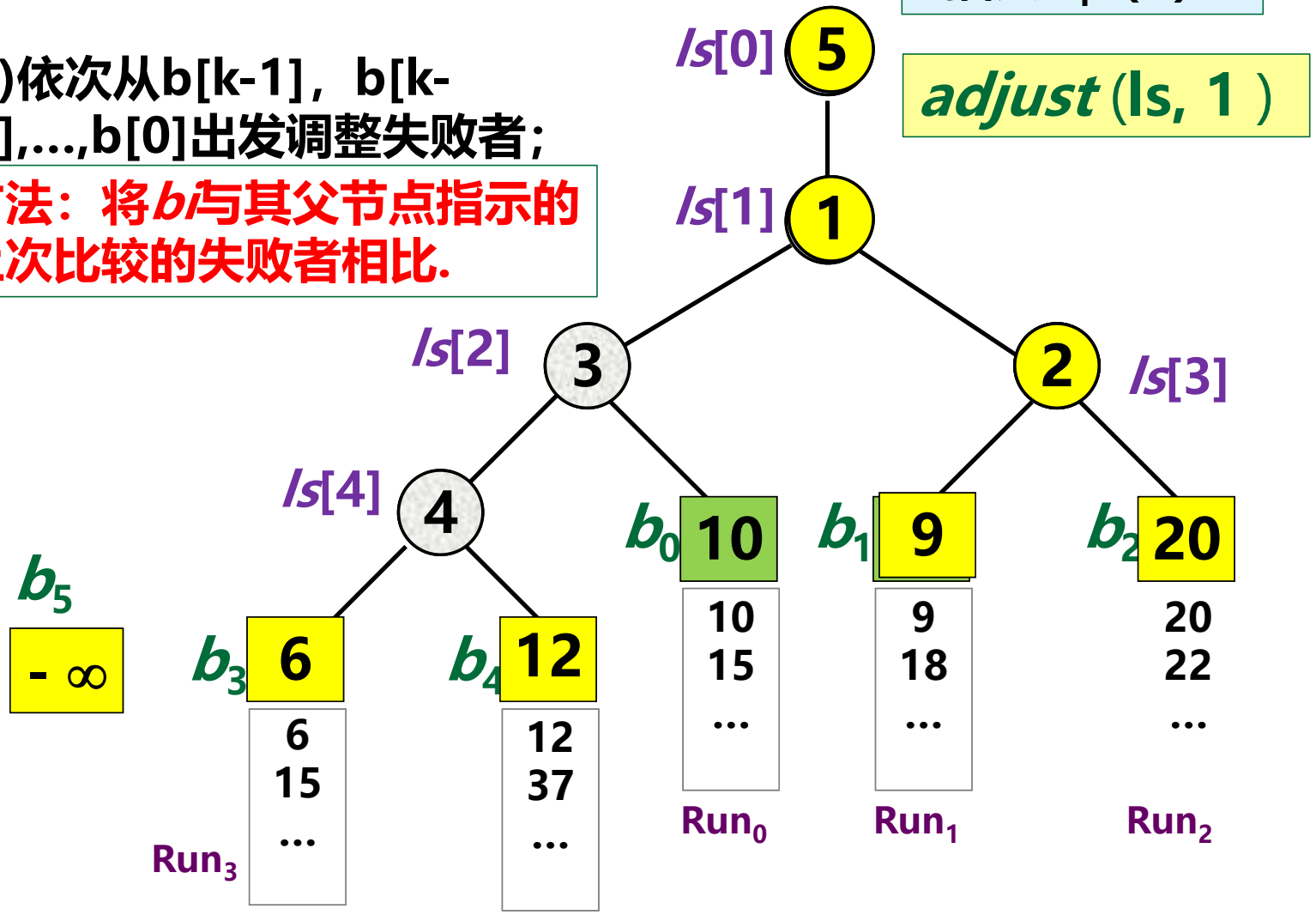
败者树-创建

创建败者树：
插入b_1（9）

adjust (ls, 1)

3)依次从b[k-1]，b[k-2],...,b[0]出发调整失败者；

方法：将*b*i与其父节点指示的上次比较的失败者相比.



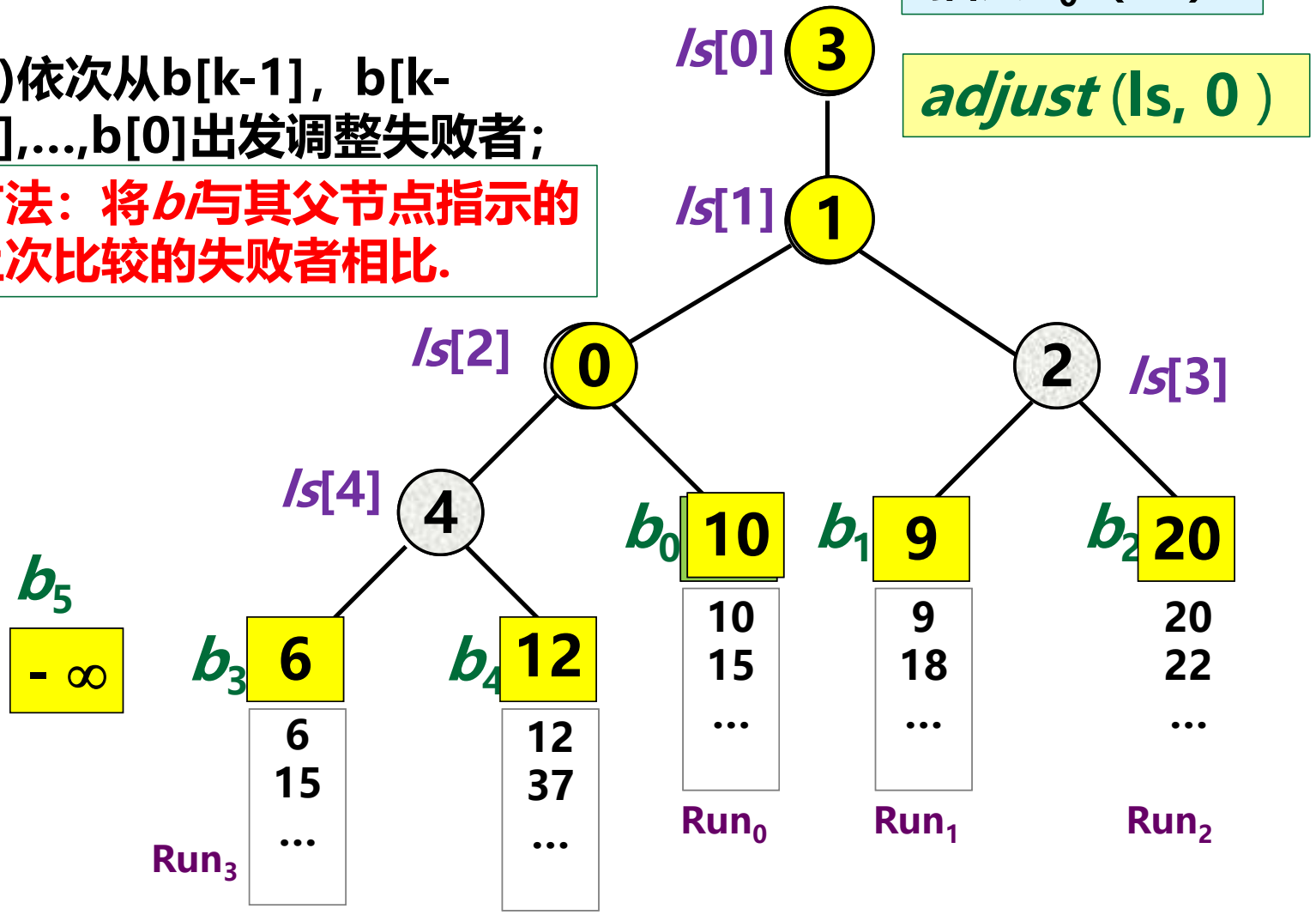
败者树-创建

创建败者树：
插入b_0（10）

adjust (ls, 0)

3)依次从b[k-1]，b[k-2],...,b[0]出发调整失败者；

方法：将*b*i与其父节点指示的上次比较的失败者相比.



败者树-创建

```
//创建初始失败者树的算法
void CreateLoserTree ( LoserTree &ls)
{//已知b[0]到b[k-1]为完全二叉树ls的叶子节点，
//其中存有k个关键字；b[k]为辅助节点。
//沿从叶子节点到根节点的k条路径将ls调整为失败者树
  b[k].key = MINKEY; //设MINKEY为可能的最小值
  for ( i =0; i < k; ++i ) ls[i] = k;//所有中间节点初始化为k
  for ( i =k-1; i >=0; --i )  Adjust( ls, i);

}// CreateLoserTree
```

k-merge 算法

```
void k-Merge( LoserTree &ls, External &b )
{//利用败者树将编号从0到k-1的k个输入归并到输出段。
//b[0]到b[k-1]记录k个输入段中当前记录的关键字
  for( i =0; i < k; ++i )  input( b[i].key ); //输入
  CreateLoserTree(ls); //创建初始败者树
  while ( b[ls[0]].key ! = MAXKEY){
        q = ls[0]; // q指示当前最小关键字所在段号
        output(q); //输出q段中当前记录
        input(b[q].key, q); //输入q段中下一个记录
        Adjust(ls, q); //调整败者树
  } //while
}// k-Merge
```

回顾：外部排序

当内存一定的情况下，归并路数k 不能无限制的增加。
每一个叶子节点对应的内存段至少要等于一个外存页面大小。

- **减小总读写次数 d的途径：**
  - **途径1：增加归并路数k**
  - **途径2：减小初始段数m。**

10.3 置换选择排序

归并的趟数：s = $=\lceil \log_k m \rceil$

- **段数m = 记录总数n / 内存可容纳的记录数目；**

- **如果减小段数m，则需要增加内存的使用量。**
- **但是内存的限制是一定的，如何在不增加内存的情况下减少段数m呢?**

减小总读写磁盘次数 d的途径2：减小初始段数m.

10.3 置换选择排序

Replacement-Selection Sorting

在整个排序的过程中，选择最小关键字和输入输出交叉或平行进行。

设输入文件*F*中24个对象的关键码序列为

{51,49, 39, 46, 38, 29, 14, 61, 15, 30, 1, 48, 52, 3, 63, 27, 4, 13, 89, 24, 46, 58, 33, 76}

假设内存工作区WA可容纳6个记录，得：

Run1: 29, **38, 39, 46, 49, 51**

Run2: 1,14, 15, 30, 48, 61

Run3: 3, 4, 13, 27, 52, 63

Run4: 24, 33, 46, 58, 76, 89

减小总读写磁盘次数 d的途径2：减小初始段数m.

10.3 置换选择排序

按照选择和置换方法构造初始排序段：
过程的步骤如下：

1. 从输入文件*FI*中把w个对象读入内存WA中，并构造败者树。
2. 利用败者树在w中选择一个关键码最小的对象记为MINMAX。
3. 将MINMAX记录写到输出文件*FO*中。

FO	WA (6)	FI
空	空	51,49, 39, 46, 38, 29, 14, 61, 15, 30, 1, 48, 52, 3, 63, 27, 4, 13, 89, 24, 46, 58, 33, 76
空	51, 49, 39, 46, 38, 29	14, 61, 15, 30, 1, 48, 52, 3, 63, 27, 4, 13, 89, 24, 46, 58, 33, 76
29	51, 49, 39, 46, 38, 14	61, 15, 30, 1, 48, 52, 3, 63, 27, 4, 13, 89, 24, 46, 58, 33, 76
29,38	51, 49, 39, 46, 61, 14	15, 30, 1, 48, 52, 3, 63, 27, 4, 13, 89, 24, 46, 58, 33, 76
29,38,39	51, 49, 15, 46, 61, 14	30, 1, 48, 52, 3, 63, 27, 4, 13, 89, 24, 46, 58, 33, 76
29,38,39,46	51, 49, 15, 30, 61, 14	1, 48, 52, 3, 63, 27, 4, 13, 89, 24, 46, 58, 33, 76
29,38,39,46,49	51, 1, 15, 30, 61, 14	48, 52, 3, 63, 27, 4, 13, 89, 24, 46, 58, 33, 76
29,38,39,46,49, 51	48, 1, 15, 30, 61, 14	52, 3, 63, 27, 4, 13, 89, 24, 46, 58, 33, 76
29,38,39,46,49, 51,61	48, 1, 15, 30, 52, 14	3, 63, 27, 4, 13, 89, 24, 46, 58, 33, 76

10.3 置换选择排序

4. 若*F*未读完, 则从*F*读入下一个对象到WA中。
5. 从WA中所有比关键字MINMAX大的记录中，选择最小的记录，作为新的MINMAX;
6. 重复3～5 ，直到WA中选不出新的MINMAX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, 直到WA为空。

- 所得结果为：
  - Run1： 29, 38, 39,46,49,51,61
  - Run2: 1, 3, 14, 15, 27, 30, 48, 52, 63, 89
  - Run3: 4, 13, 24, 33, 46 ,58, 76

10.3 置换选择排序

在WA中选择MINMAX记录，需要用败者树实现。

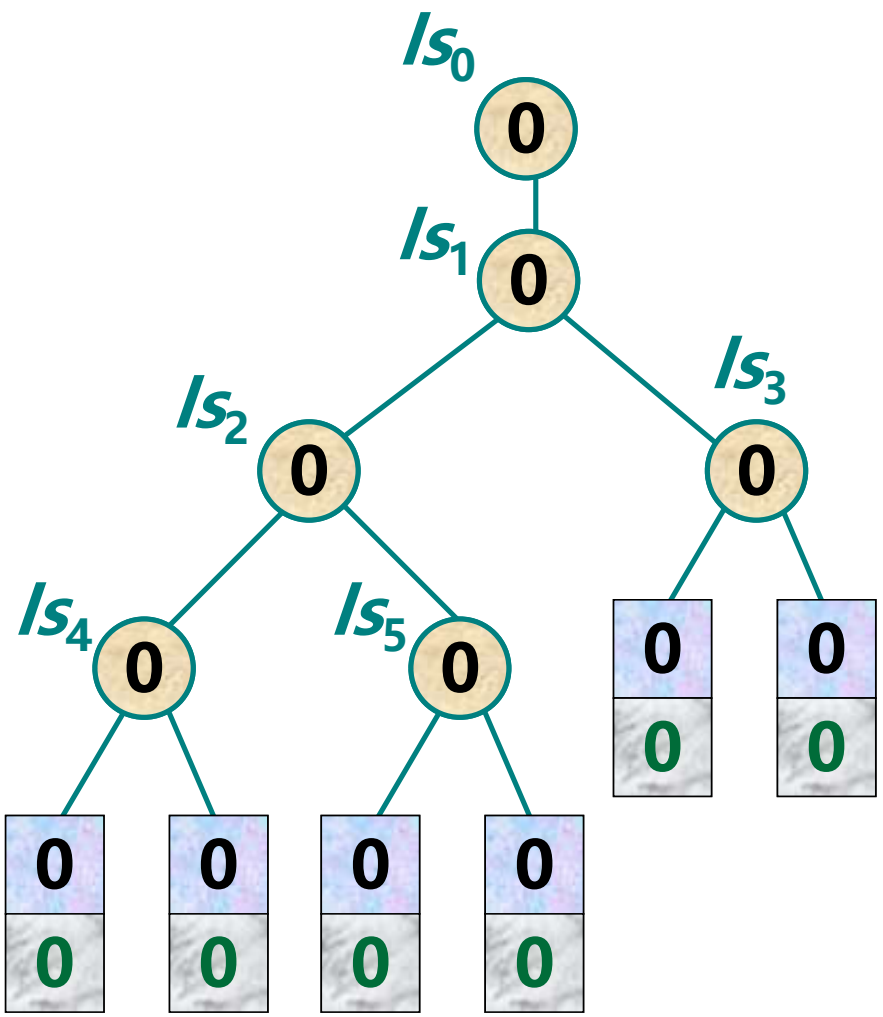
说明：

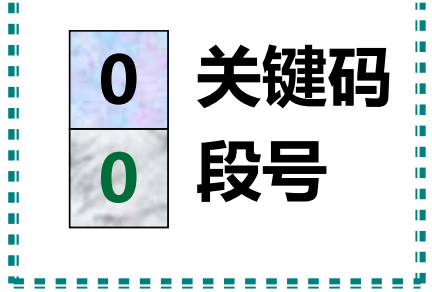
（1）WA中的记录作为败者树的外部节点，败者树中的根节点的父节点指示WA中关键字最小的记录；

（2）为了便于选出MINMAX记录，为每个记录除了关键字外，附加一个所在归并段序号。在比较时，先比段号，段号小的为胜；

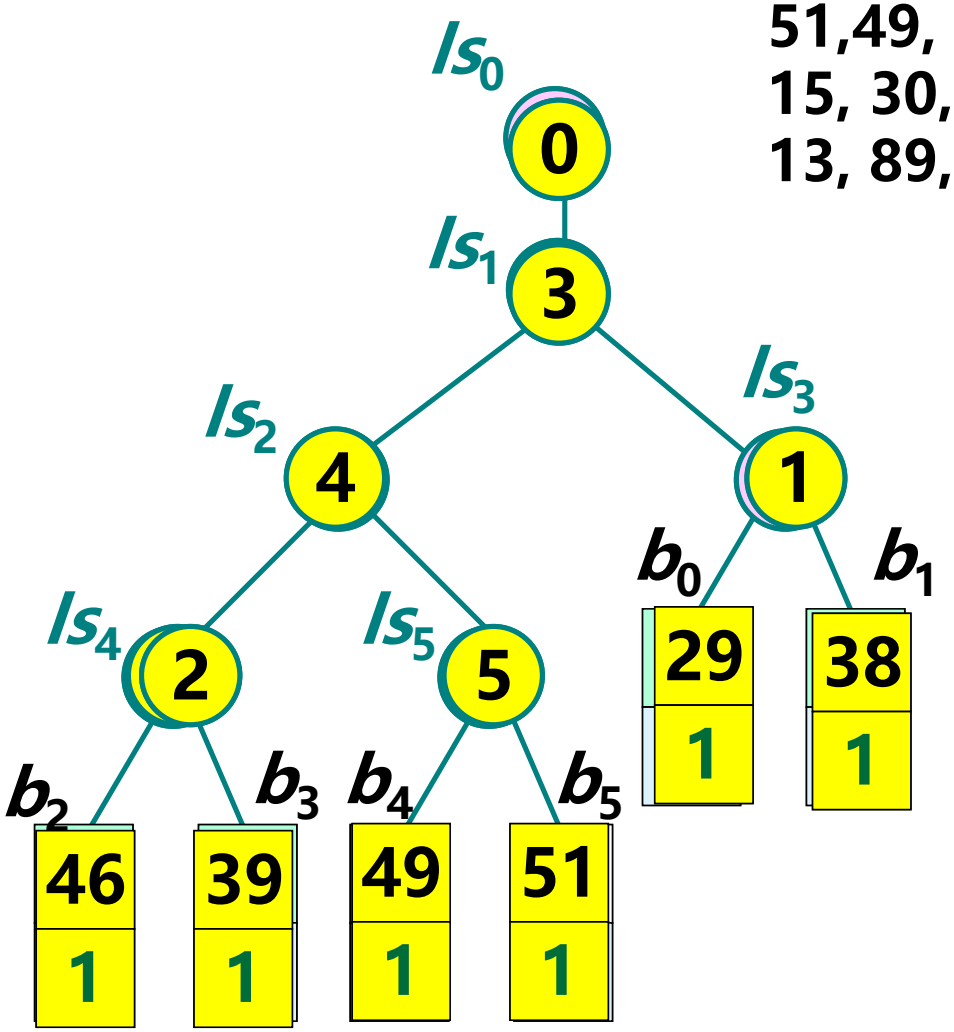
（3）败者树的建立可从设工作区中所有记录的段号均为“零”开始，然后从FI中输入w个记录，自上而下调整败者树。

置换选择排序中的败者树——初始化树





置换选择排序中的败者树——初始化树



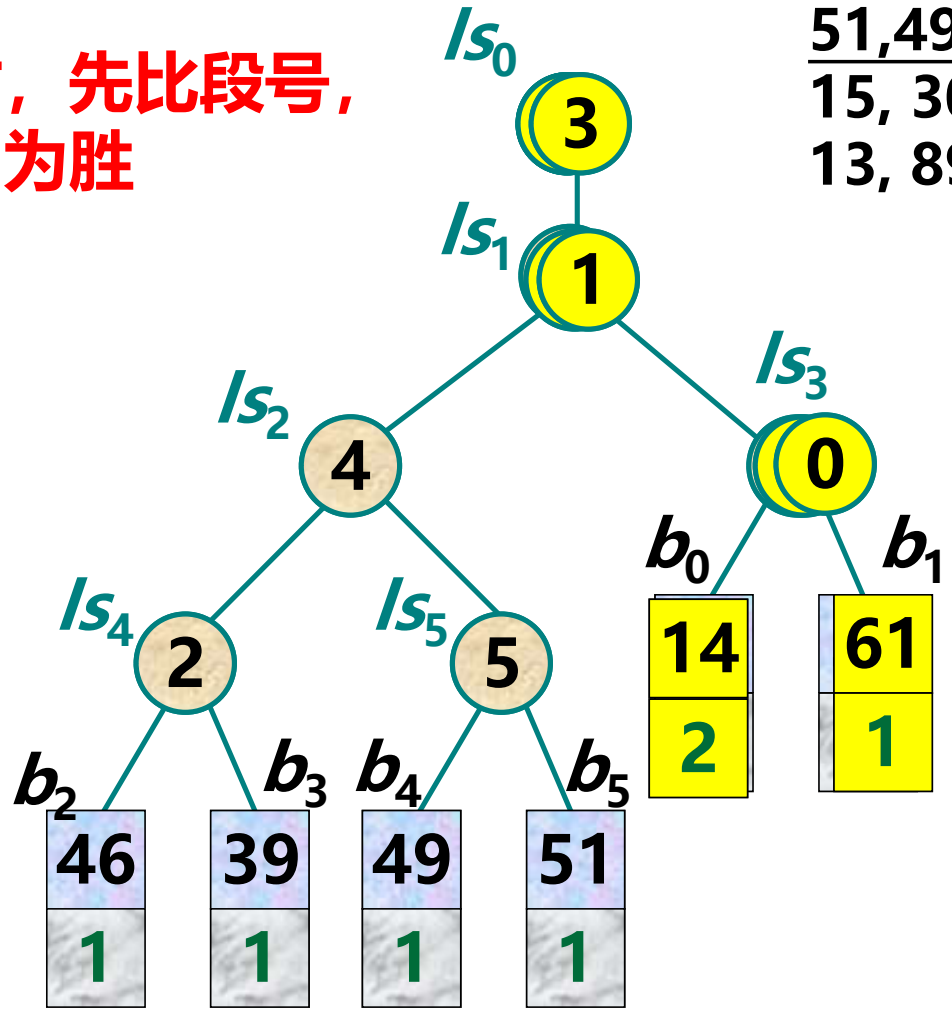
51,49, 39, 46, 38, 29, 14, 61,
15, 30, 1, 48, 52, 3, 63, 27, 4,
13, 89, 24, 46, 58, 33, 76

MINMAX=29

置换选择排序中的败者树——初始化树

在比较时，先比段号，
段号小的为胜

51,49, 39, 46, 38, 29, 14, 61,
15, 30, 1, 48, 52, 3, 63, 27, 4,
13, 89, 24, 46, 58, 33, 76

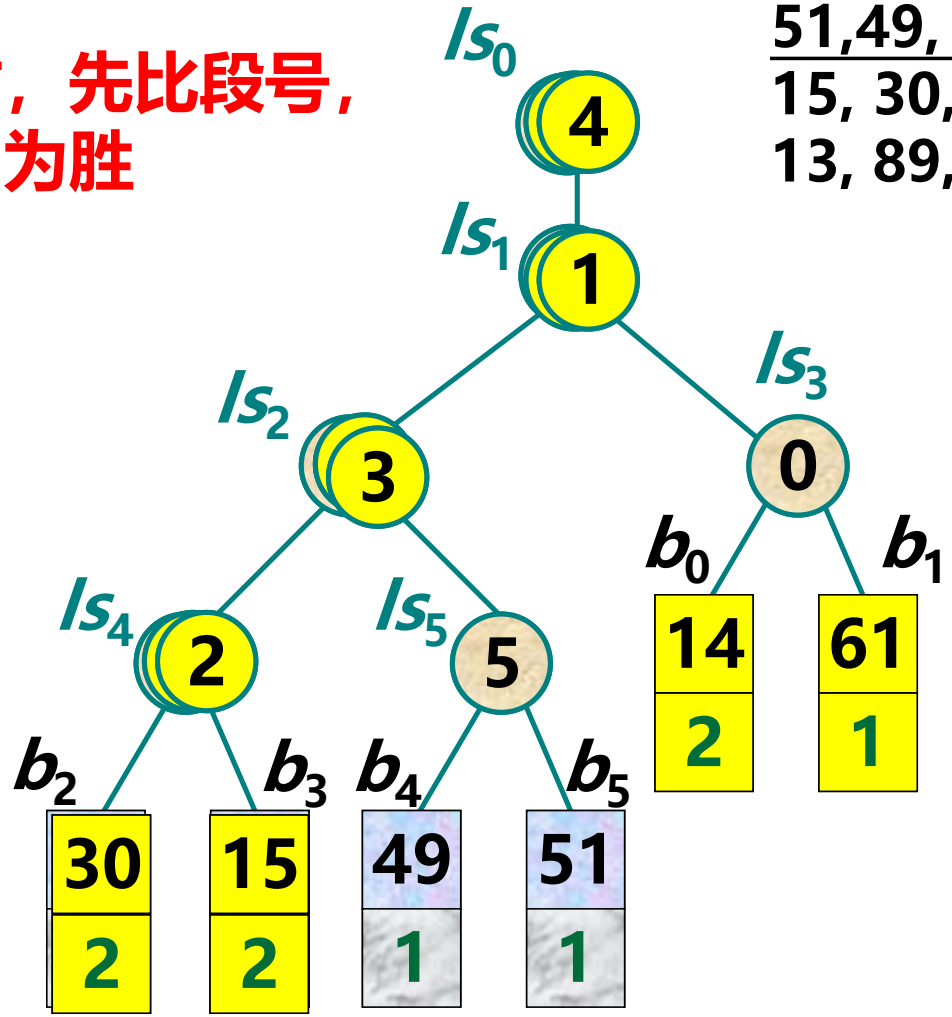


MINMAX

29	38

置换选择排序中的败者树——初始化树

在比较时，先比段号，段号小的为胜



51,49, 39, 46, 38, 29, 14, 61,
15, 30, 1, 48, 52, 3, 63, 27, 4,
13, 89, 24, 46, 58, 33, 76

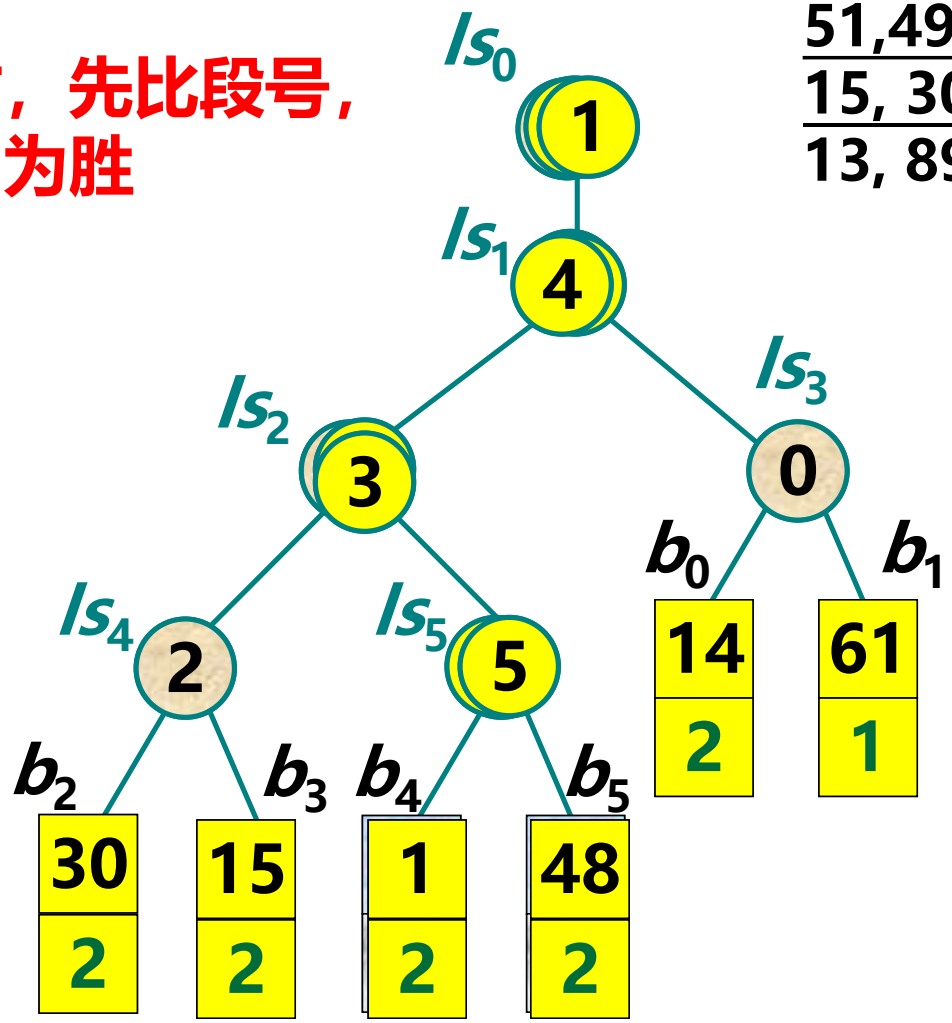
MINMAX

29	38	39	46	49

41

置换选择排序中的败者树——初始化树

在比较时，先比段号，段号小的为胜



51,49, 39, 46, 38, 29, 14, 61,
15, 30, 1, 48, 52, 3, 63, 27, 4,
13, 89, 24, 46, 58, 33, 76

MINMAX

29	38	39	46	49
51	61			

10.3 置换选择排序

特殊情况：**输入对象序列已经按关键码大小排好序**

只生成一个初始归并段。

若输入文件有 n 个对象，生成初始归并段的时间

在一般情况下，每输出一个对象，对败者树进行调整需要时间为$O(\log_2 w)$。

对于n 个对象，生成初始归并段的时间开销是$O(n\log_2 w)$

减小总读写磁盘次数 d的途径1：增加归并路数k

减小总读写磁盘次数 d的途径2：减小初始段数m.

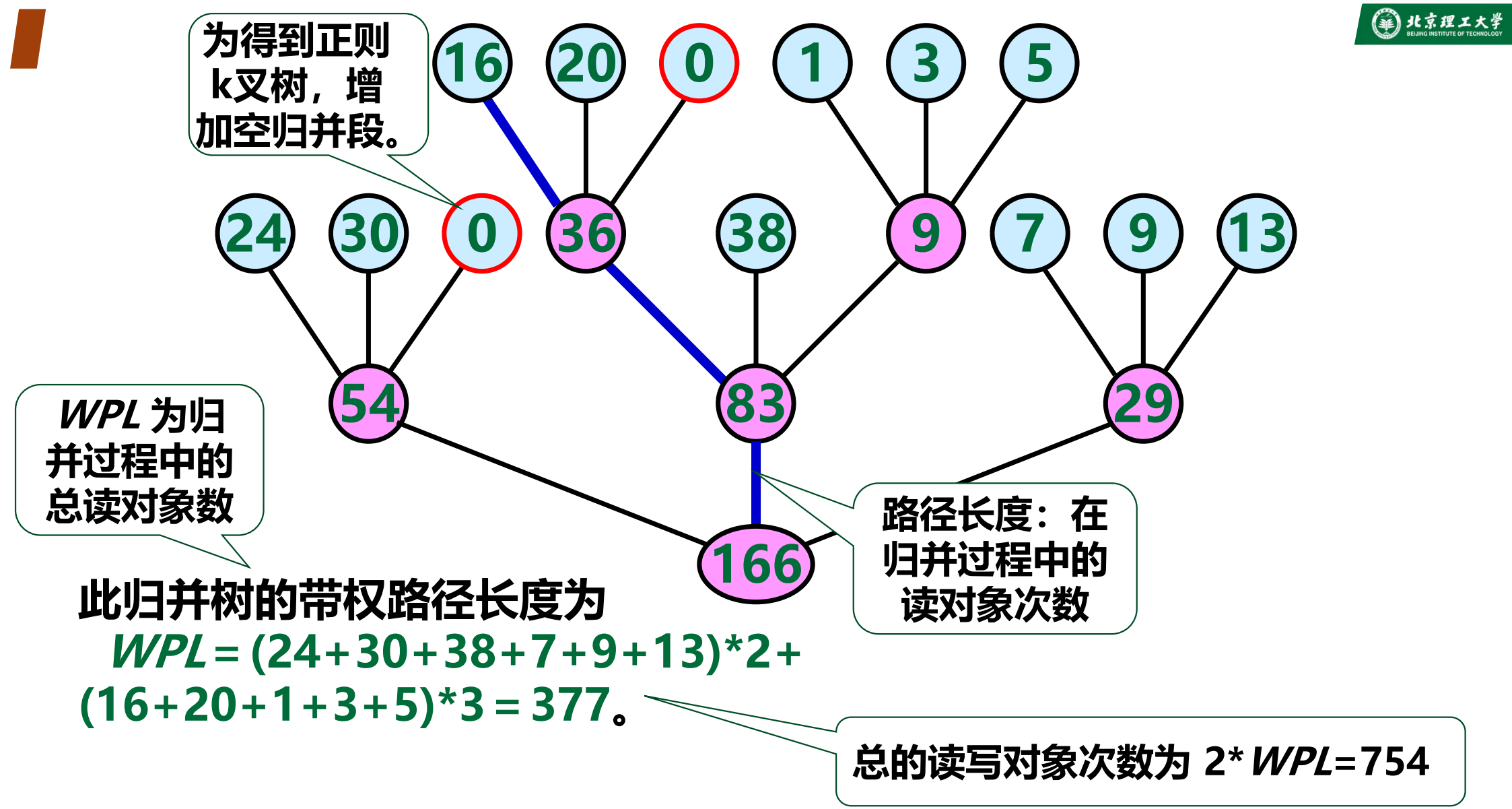
10.4 最佳归并树

归并树是描述归并过程的 k 叉树。因为每一次做 k 路归并都需要有 k 个归并段参加，因此，归并树是只有度为0和度为 k 的结点的正则 k 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。


为得到正则k叉树，增加空归并段。
16
20
0
1
3
5
24
30
0
36
38
9
7
9
13
54
83
29
166
WPL 为归并过程中的总读对象数
路径长度：在归并过程中的读对象次数
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。
总的读写对象次数为 2* WPL=754

10.4 最佳归并树

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。

将哈夫曼树的思想扩充到 k叉树的情形。

10.4 最佳归并树

例如，假设有m=6个初始归并段, 其长度(对象个数)分别为：　　13, 16, 20, 24, 30, 38

做3路归并。

归并树是只有度为 0 和度为 3的结点的正则 3叉树。

$$n_0 = 2n_3 + 1$$

$$n_3 = (n_0 - 1) / 2。$$

假设需要增加的虚段数为：u，则

n_3 =(m + u - 1) / 2　为整数

u = (m-1) % 2

正则 3叉树: $n_0 + n_3 = 3n_3+1$

10.4 最佳归并树

为使归并树成为一棵正则k叉树，可能需要补入空归并段。补空归并段的方法为：

归并树是只有度为 0 和度为 k 的结点的正则 k 叉树，设度为 0 的结点有 n_0个，度为 k的结点有 n_k 个，则有：

$$n_0 = (k - 1)n_k + 1$$

$$n_k = (n_0 - 1) / (k - 1)。$$

如果 $(m-1) \% (k-1) = 0$, 则说明这 m 个叶结点(即初始归并段)正好可以构造 k叉归并树。

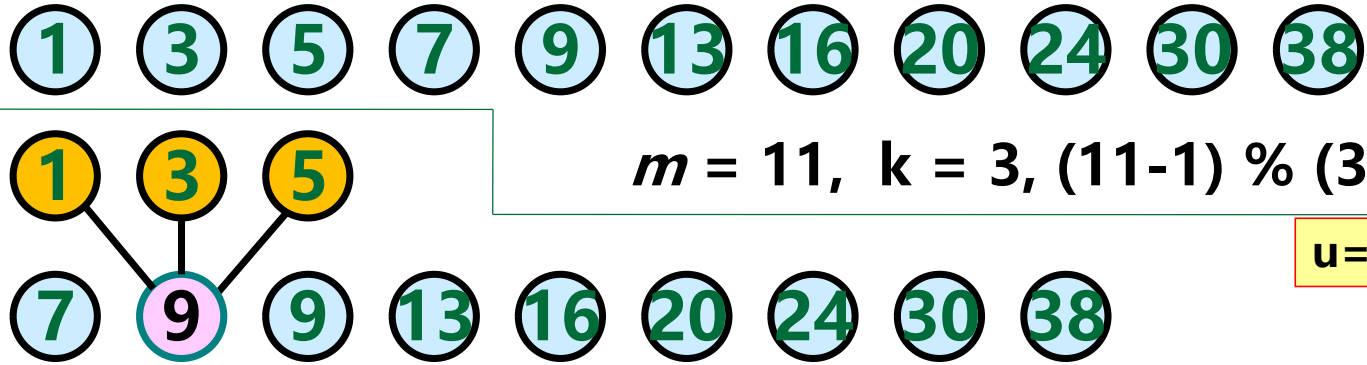
正则 k叉树: $n_0 + n_k = kn_k + 1$

10.4 最佳归并树

$n_k = (n_0 - 1) / (k - 1)$。

如果 (m-1) % (k-1) < > 0
需要增加u个空归并段，使满足：
n_k =(m+u-1) /(k-1) 为整数;
则： u = k-1 - (m-1) % (k-1) ;

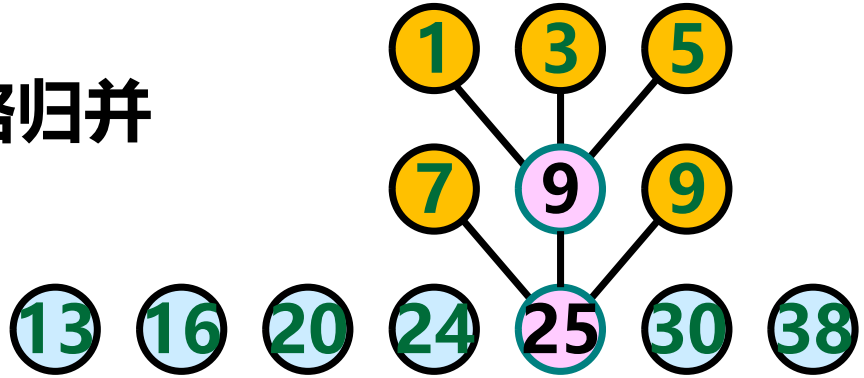
$$u = \begin{cases} 0, if(m-1)\%(k-1) = 0 \\ k-1-(m-1)\%(k-1), else \end{cases}$$

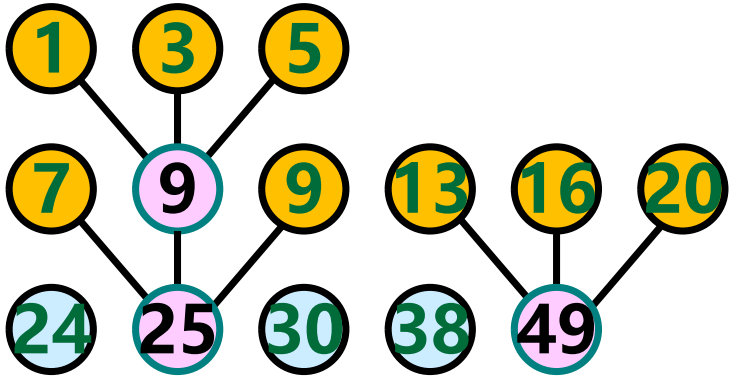


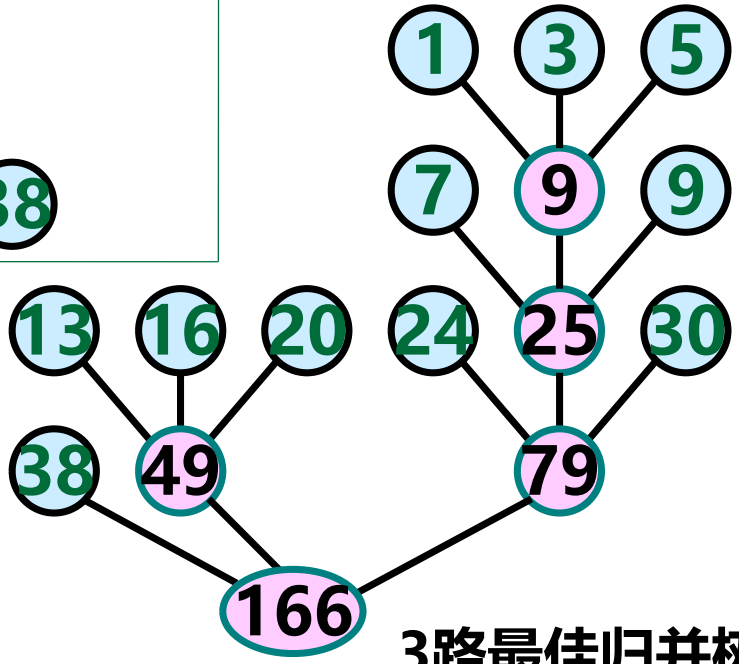
m = 11, k = 3, (11-1) % (3-1) = 0

u= k – 1 - (m-1) % (k-1)

11段—3路归并







3路最佳归并树

1 3 5 7 9 13 16 20 24 30 38

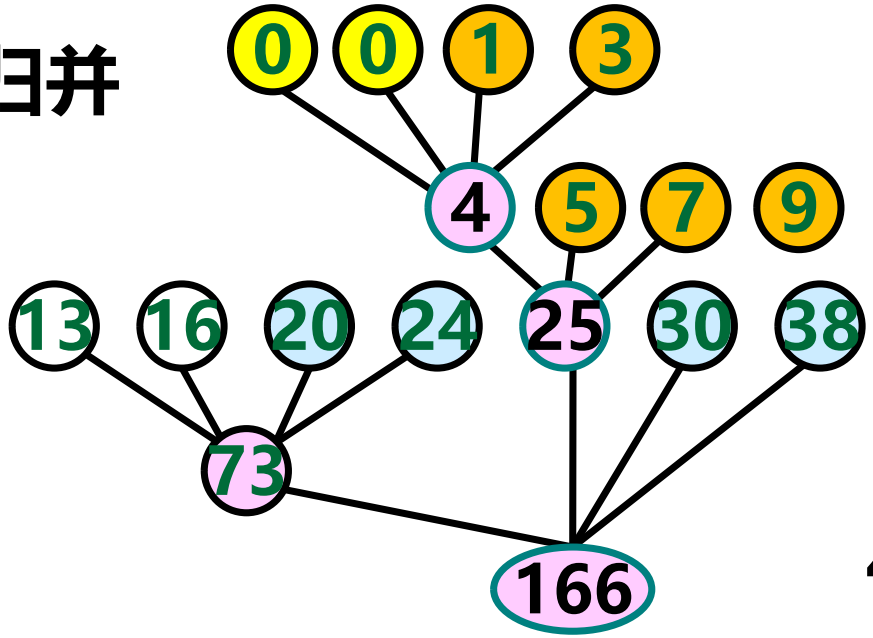
m= 11, k = 4, (11-1) % (4-1) = 1　　需补充(k-1)-1=2个空段

0 0 1 3

4 5 7 9 13 16 20 24 30 38

u= k – 1 - (m-1) % (k-1)

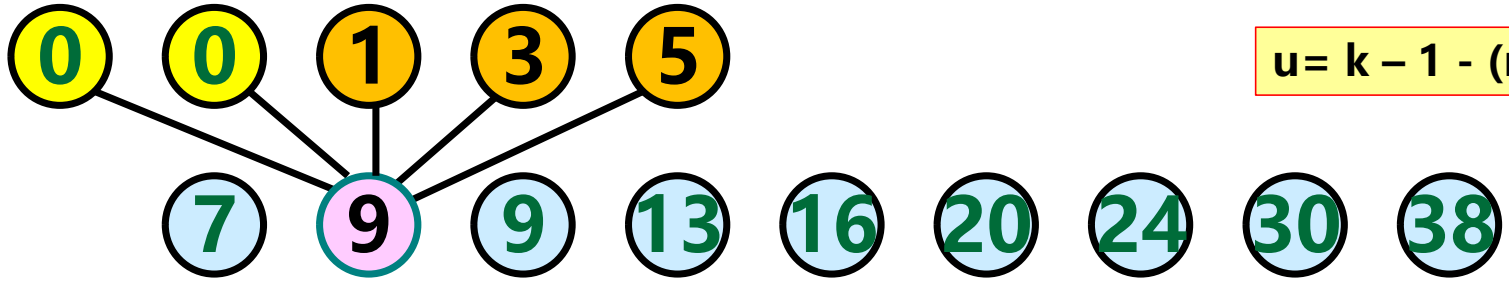
11段—4路归并

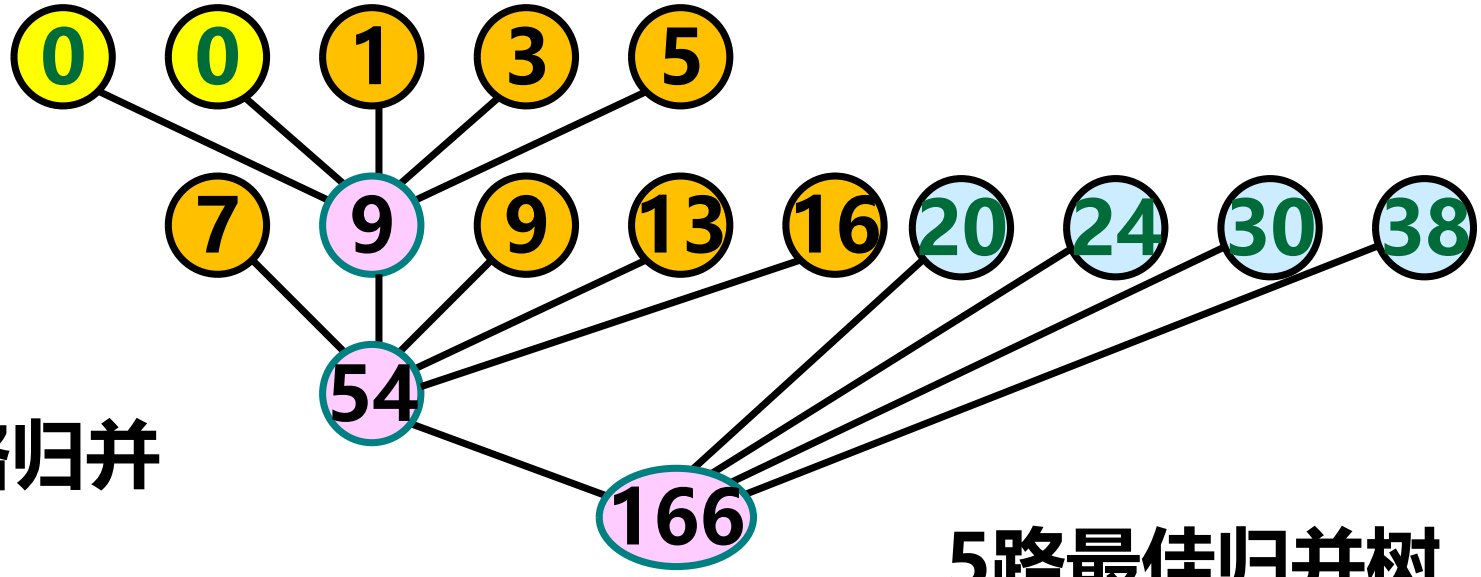


4路最佳归并树

m= 11, k = 5, (11-1) % (5-1) = 2　需补充k-1-2=2个空段

u= k – 1 - (m-1) % (k-1)





11段—5路归并

5路最佳归并树

外部排序小结

1、内部排序过程中不需要数据的内、外存交换，待排序的记录全部存放在内存中；

2、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。

3、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；多路归并。

4、外部排序主要研究的技术问题是：

(1) 如何进行多路归并以减少文件的归并遍数；

(2) 根据外存的特点选择较好的产生初始归并段的方法。

本章学习要点

1. 理解外部排序的概念和方法。
2. 熟练掌握多路平衡归并排序方法（败者树）。
3. 理解置换-选择排序的基本思想。
4. 掌握最佳归并树的构造方法。

1、 对于输入文件{101,51,19,61,3,71,31,17,19,100,55,20,9,30,50,6,90}，当k=6时，适用置换-选择排序的方法，写出建立的初始败者树及生成的初始归并段。

2、 设有11个长度不同的初始归并段，他们所包含的记录个数分别是25,40,16,38,77,64,53,88,9,48,98。试对它们做四路平衡归并，要求：

（1）构造最佳归并树。

（2）根据最佳归并树计算记录读写的总次数。

3、 假设使用置换-选择排序算法产生文件F的初始归并段，在假设内存缓冲区的长度为w，请回答下面各个问题：

（1）能够产生的最长和最短的归并段的长度各是多少？

（2）为什么产生的第二个归并段通常比第一个归并段长？

（3）最后产生的归并段的长度能否大于w？为什么？