第十二章 文件结构

一.概述

就工作方式来看存贮器一般分为两部分，一部分称内存贮器(也称主存储器)，另一部分称外存贮器(也称辅助存储器)。

根据外部存储设备的特性，主流存储设备大体上可以分为两类：

- 顺序存取设备(例如，磁带)
- 直接存取设备(例如，磁盘)

这里简单介绍一下磁带和磁盘存储器。

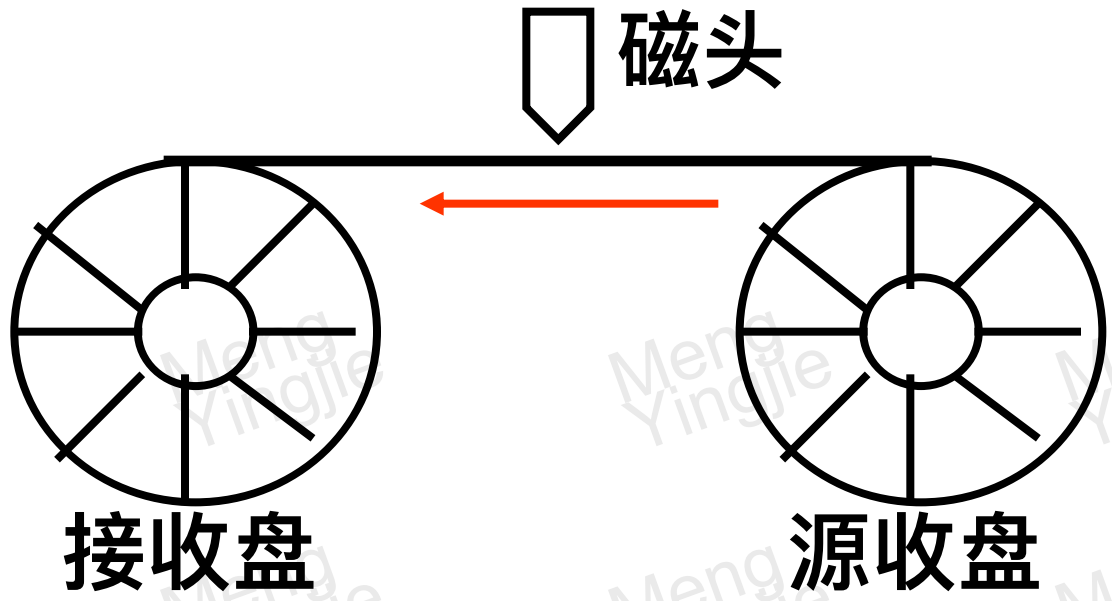
二.磁带存储器(magnetic tape storage):

磁带存储器始于1950s，是计算机中可重复使用的一种顺序访问存储器。

因其价格便宜，使用方便，容量大而广泛使用。

磁带存储装置由磁带介质、读/写磁头和磁带驱动器组成。

磁带装置工作示意图如下：



从1953年IBM推出利用真空缓冲带箱的726型开始，磁带驱动设备开始了新的局面，并奠定了一直沿用至今的0.5英寸带宽的通用标准。

传统的典型尺寸：宽0.5英寸，长2400，带厚0.002英寸，存储密度800字节/英寸或1600字节/英寸，带面有9道信息(其中1位奇偶校验位，8道数据)

存取时间主要用于定位上(转到所需位置)，一般在20毫秒~10分钟，定位完成后才能进行读/写。读/写速度由带速和存储密度等因素决定。

磁带的启动时间和停止时间的统称启停时间。

在启停时间不能进行读/写操作,因此，磁带上信息的存储是分块的，块与块之间存在间隙。

为适应顺序存取和成批处理，一般采取分页、块存储方法(在每页、块中均有大量记录)，页、块之间留间隙以减少存储空间的浪费。

经过近几十年的几代技术改进，主要指标的变化：

存储密度：100b/inch → 19000b/inch;

带速： 75inch/second → 200inch/second

传输率： 7.5kb/second → 1250kB/second

块间间隙：19.05mm → 7.62mm

道数：9 → 168 (其中有伺服信号24道)

容量：几兆 → 几十GB

目前常见的小型盒带多为4mm和8mm。

三.磁盘存储器(magnetic disk storage):

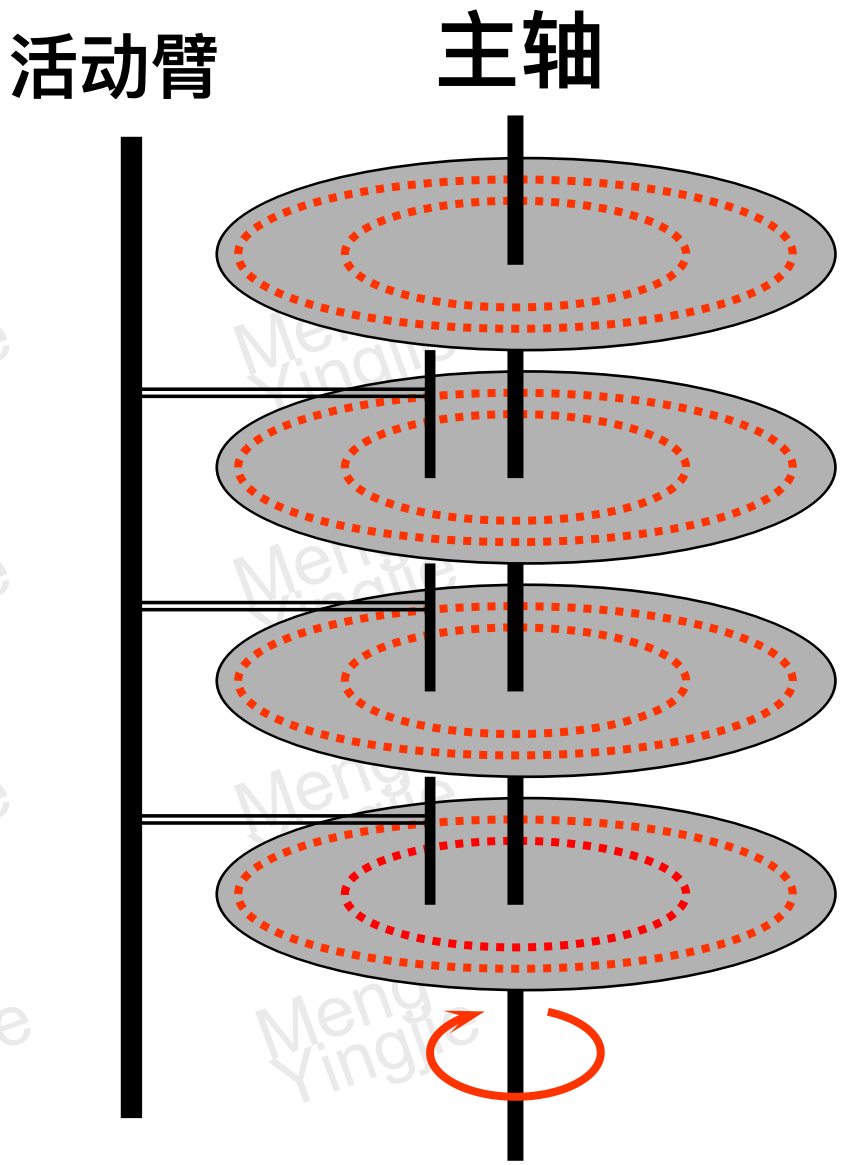
在恒速旋转的圆形磁性媒体表面沿同心环形轨迹，通过磁头电磁转换器件进行数据记录的直接存取存储设备。也称旋转磁表面型存储设备。

磁盘存储器由盘片介质、磁盘驱动器以及磁盘控制器构成。

是目前首选的辅助存储器。

主要有磁盘组(disk pack)和软磁盘。

磁盘组示意图：



几个概念：

- **磁道**，磁头的写入磁场在记录盘面上形成的一条磁化轨迹。
- **柱面**，各记录盘面上相同半径的磁道合在一起称为一个柱面.
- **扇区**，将磁道分成若干段，每段称一个扇区。信息的组织以扇区为单位(簇)进行。

本节结束

一.概述

在计算机中，大量的数据存储在外部存储中，因此需研究外部存储器上数据的组织。

外部存储器上数据的组织一般以文件的形式进行组织。

文件，为了进行存取控制、检索和修改而组织在一起的数据记录的集合。

二.文件的逻辑结构:

(从用户角度来看文件的结构)分为两种形式:

字符流文件，有序的字符流序列，文件基本单位为字节或字。

记录文件，数据记录的集合，文件的基本单位为记录(定长或变长记录)。

三.文件的物理组织:

从存储结构来看，文件的组织主要有3种类型：

1.顺序结构：

按照数据到达的时间先后次序进行组织。

按该方式组织的文件称**顺序文件**。

2.计算寻址结构:

按照散列方式组织文件。该方式组织的文件称**散列文件**。多适合于随机存取，因此也称随机文件。

3.带索引的结构：

组织数据时需要带一个索引表。按该结构组织的文件统称为索引文件。主要是利用树形结构组织索引。

四.顺序文件：

从本质上说，顺序文件就是顺序表，因此就操作来讲其运算与顺序表上的运算类似，但算法应立足于尽量减少访问外存的次数。

五.散列文件:

在随机文件的组织中(例如磁盘文件), 一般以若干记录成组存放,即若干记录组织成一个存储单位——桶(bucket)。

使用该方法的同时,需要注意的是除散列函数外,还要注意处理桶的溢出问题。溢出处理可采用哈希表中处理冲突的方法。

除桶散列以外，常见的散列方法还有虚拟散列(Virtual Hashing) 、动态散列、散列树、可扩充散列等。

六.倒排文件:

索引结构采用倒排表形式组织的文件称倒排文件。

在信息检索中，倒排文件是由检索词和文献记录的控制键构成的行列矩阵。

七、多关键字文件:

索引结构采用多重表形式组织的文件称多关键字文件。

本节结束

一.概述

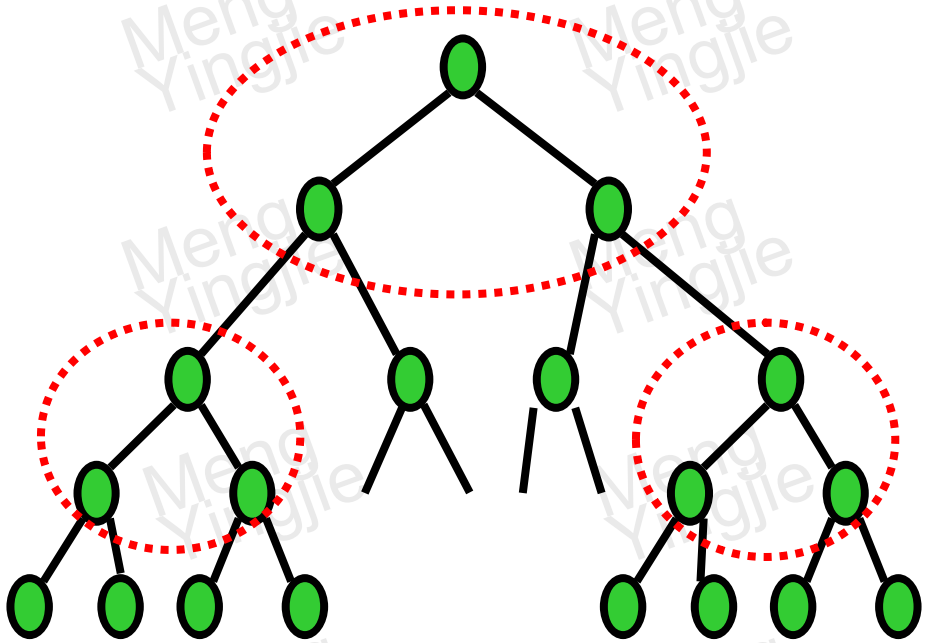
在计算机中,为了提高数据的组织效率,在文件的组织中目前普遍采用了索引结构。由于索引本身的有序性,因此索引文件本身内容的物理顺序与其逻辑顺序可以是一致,也可以是不一致的。

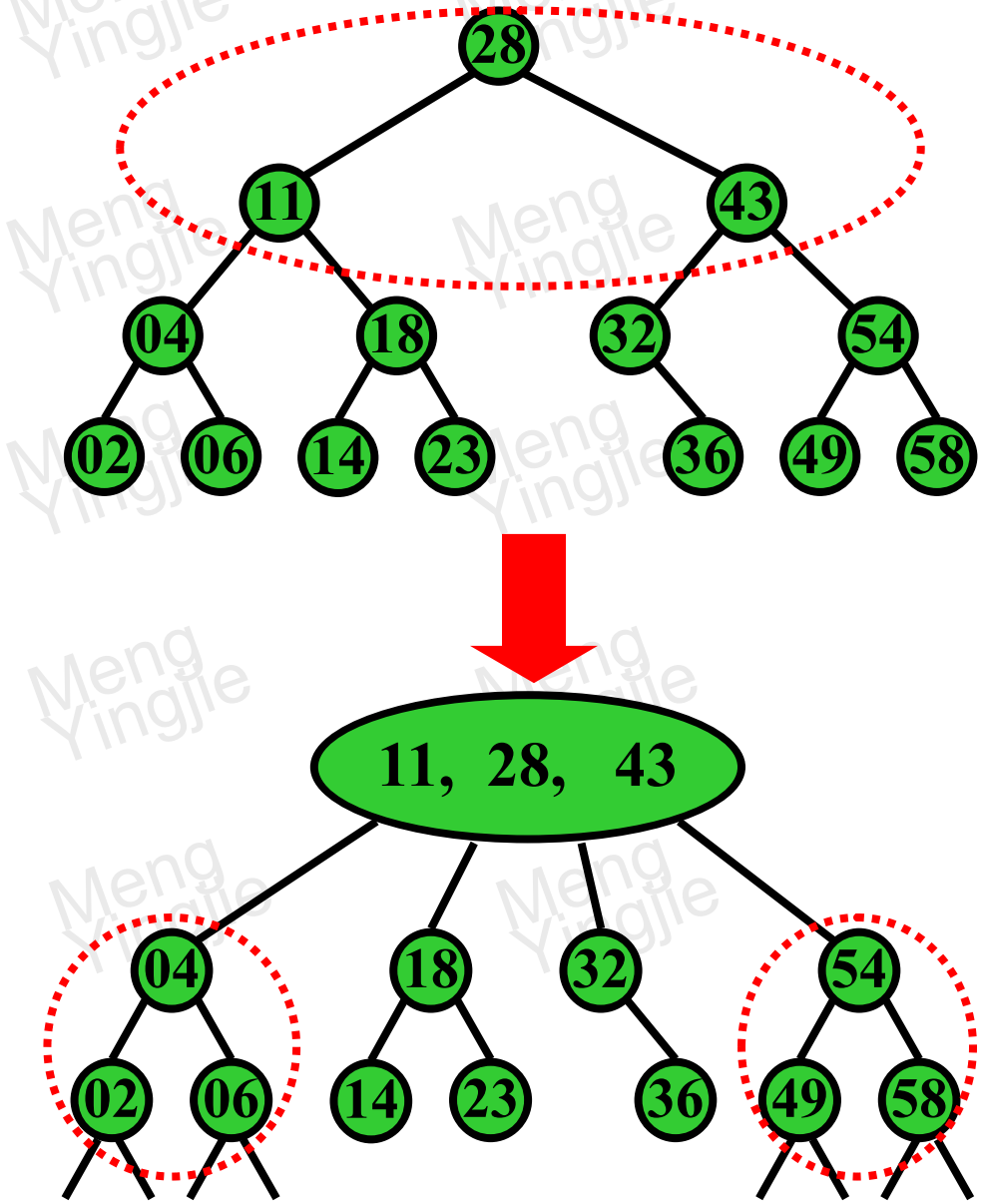
按照物理顺序与逻辑顺序一致的情况组织的索引文件我们称为索引顺序文件。

根据系统运行期间索引结构可否动态变化，索引结构主要有静态索引结构和动态索引结构。

在索引顺序文件的组织中主要利用的技术就是

多分树结构。





二.静态索引结构

1、概念:

静态索引结构是指索引结构在文件创建，初始装入记录时生成，一旦生成就固定下来，在系统运行(例如插入/删除)过程中索引结构并不发生变化，只有当文件重组时才允许改变索引结构。

2、索引顺序存取法(ISAM):

ISAM(indexed sequential access method)是现今磁盘存取文件的常用组织方法。

实际运用中ISAM又分为两种：

- 基本索引顺序存取法(BISAM)；
- 排队索引顺序存取法(QISAM)。

(1).基本索引顺序存取法(BISAM):

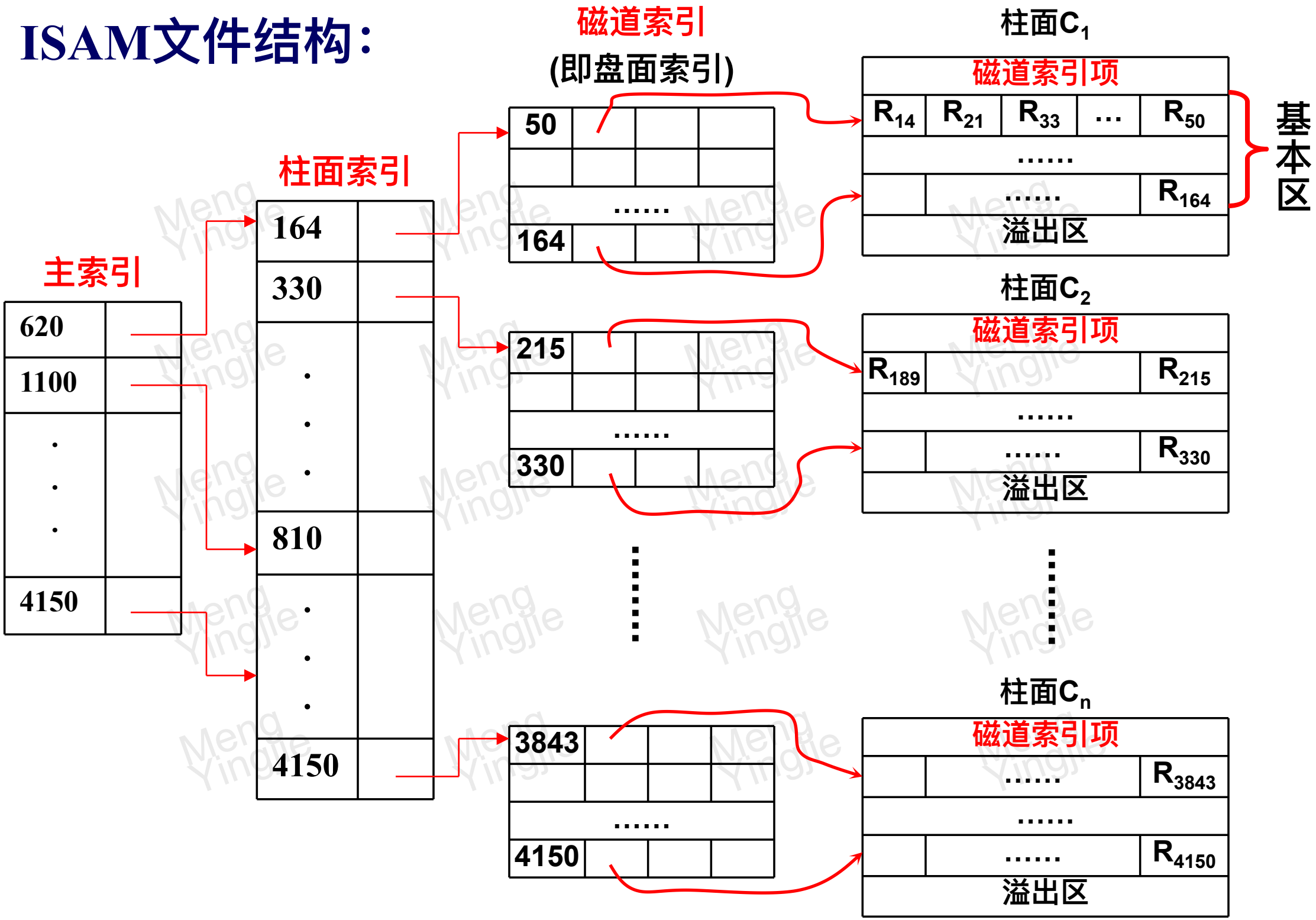
BISAM(basic indexed sequential access method)是使用索引来确定数据集在直接存取(即随机存储器)装置上的位置的一种存取法。

它可以直接检索或修改数据集的特定块。索引和数据集一起记录在直接存取存储器上。

初建时,一般用两个区(例如在硬盘扫描中可以观察到有两个区:系统区、数据区),一个区用作索引的存储,一个区用作数据记录的存储,还可以有索引的索引。

用ISAM方法组织的磁盘文件由多级主索引、柱面索引、磁道索引(即柱面的盘面索引，习惯称磁道索引)和主文件组成。

ISAM文件结构：


磁道索引
(即盘面索引)
柱面索引
主索引
620
1100
4150
164
330
810
4150
50
164
215
330
3843
4150
柱面C1
磁道索引项
R14 R21 R33 … R50
R164
溢出区
基本区
柱面C2
磁道索引项
R189 R215
R330
溢出区
柱面Cn
磁道索引项
R3843
R4150
溢出区

(2).排队索引顺序存取法(QISAM):

QISAM(queued indexed sequential access method),是基本索引顺序存取法的一种扩充。用此法时，将等待处理的输入数据块排成队列，或将已处理好的而正等待传送到辅助存储器或输出装置的输出数据块排成队列。

三、动态索引结构

1、概念

动态索引结构是指文件创建，初始装入记录时所生成的索引结构，在系统运行(例如插入/删除)过程中索引结构本身也能够发生改变。

改变索引结构的目的是为了索引结构保持较好的性能，例如较高的检索效率。

2、B-树(B-tree):

就其本质来说，这是一种平衡的多分树。

由于对B树的查找、插入、删除均始终能够保持B树的动态平衡，因此B-树及其它的一些改进形式(例如B+树、B*树、前缀B树、两分B树等)已经成为组织索引文件的一种标准的有效结构，已得到了广泛的应用。

(1)B-树的定义:

一棵m阶的B-树满足如下条件：

①每个结点至多有(≤) m个孩子结点；

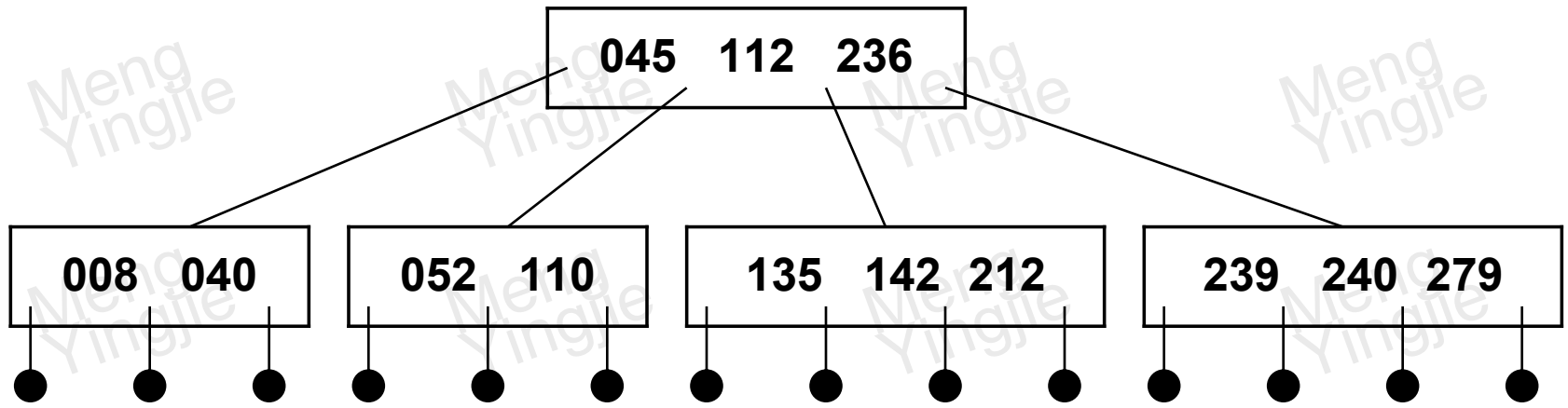
②除根结点和叶结点之外的每个结点至少有(≥)⌈m/2⌉个孩子结点；

③根结点至少要有两个孩子结点(除非它本身又是叶结点)；

④具有k个孩子结点的非叶结点包含k-1个关键字；

⑤所有的叶结点都在同一层上，并且它们都不带有关键字信息。

例，m=6的B-树:



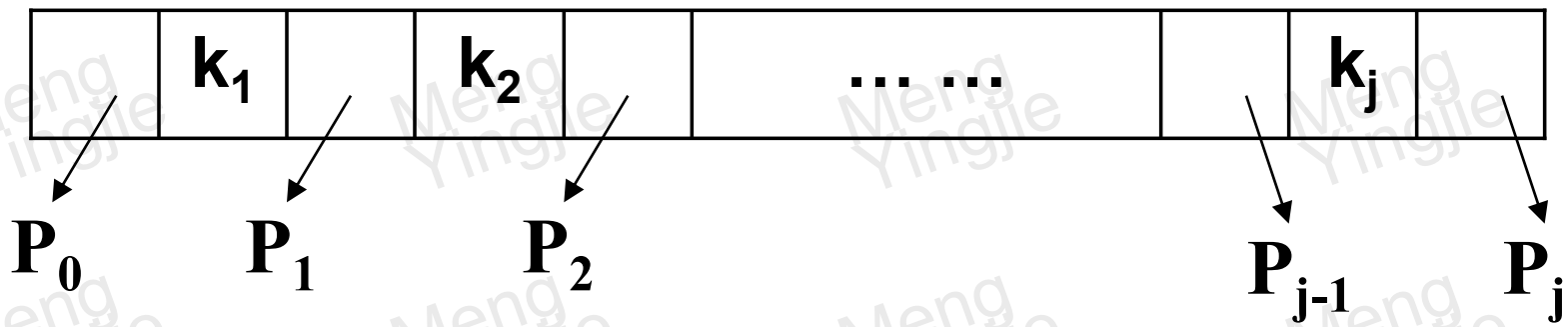
其中每个结点(除根和叶),有⌈6/2⌉到6个孩子，因此结点中可以包含的关键字可以有2、3、4或5。

显然我们对1阶和2阶的B-树不感兴趣，我们这里只考虑m ≥3的情况。

阶为3的B-树，称为2-3树(2个关键字3棵子树，也可称为3-2树)

B–树结点形式:

一个包含j个关键字，j+1个指针的结点的一般形式：

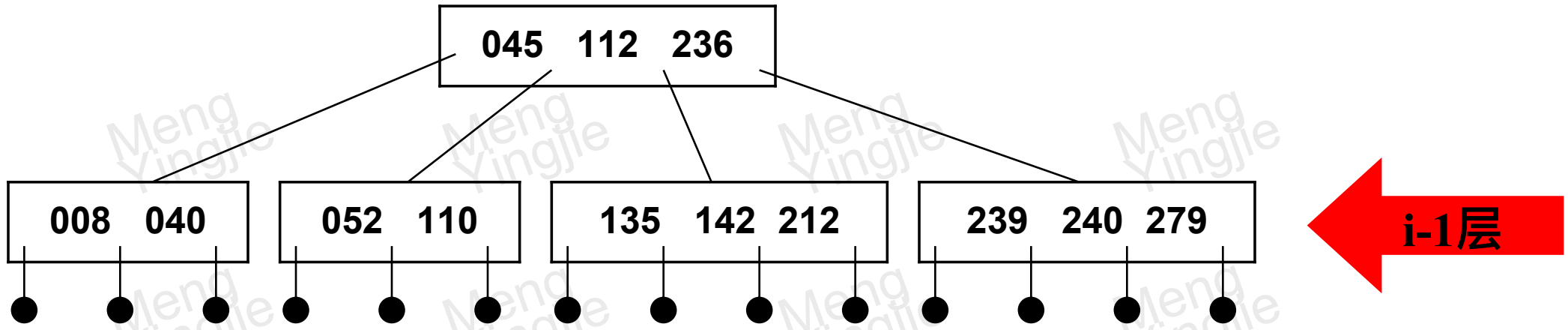


其中：k_i为关键字的值， $k_1 < k_2 < \ldots < k_j$

P_i为指针，指向包括k_{i-1}到k_i之间的关键字的子树。

以下简单讨论一下B-树的运算：检索、插入、删除。

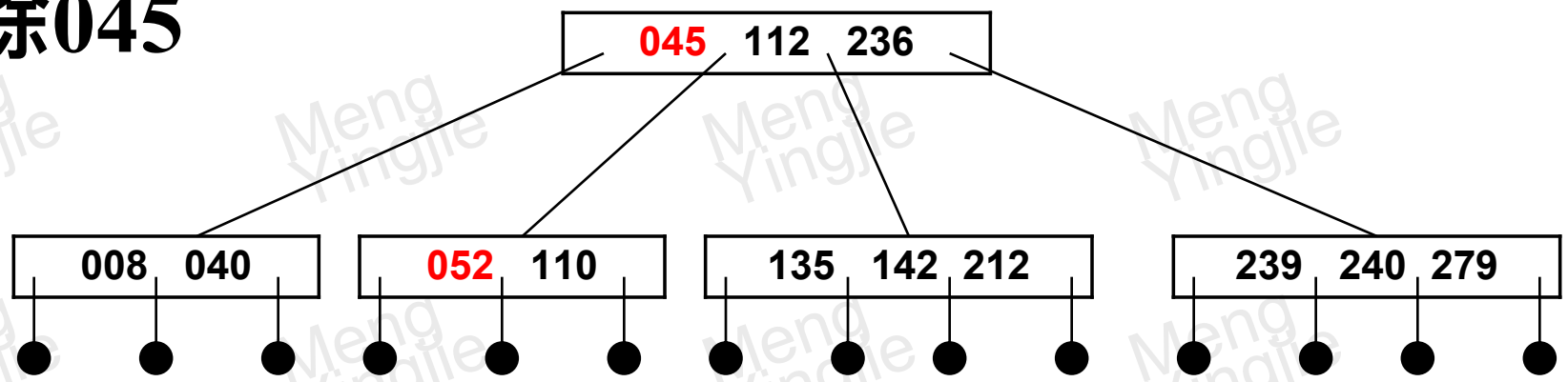
(2).B–树中的运算



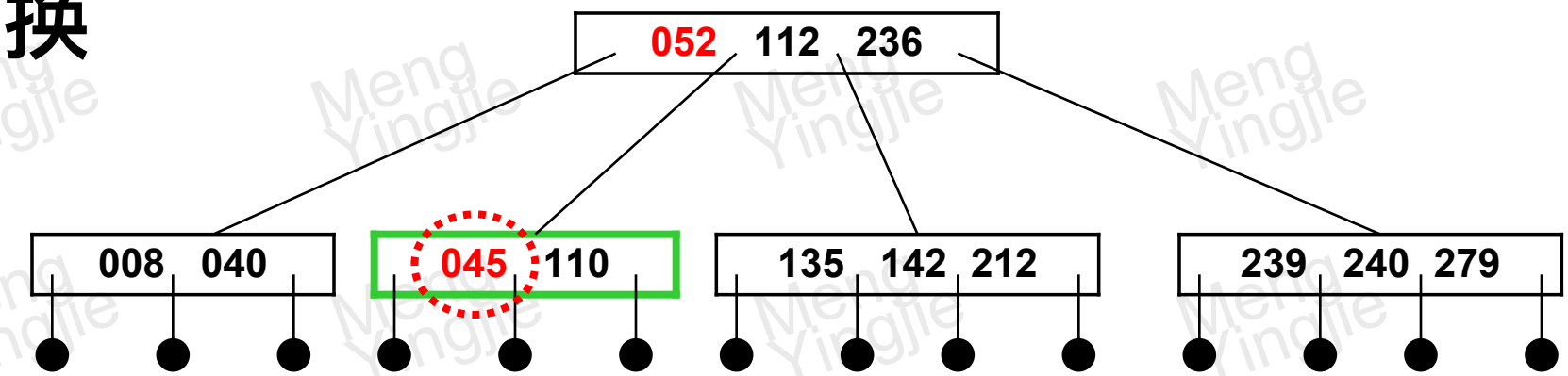
- **检索**： 从根开始。
- **插入**： 对于叶结点处于第i层的B–树,插入关键字总是首先进入第i–1层的结点中。未满插入，已满分解，可能导致高度生长。
- **删除**:与插入类似，删除关键字总是从第i–1层的结点开始，若删除的关键字不在第i–1层，则先要将此关键字与它在B–树中的直接后继交换位置，交换到第i–1层,然后删除，可能导致高度减小。

例，m=6的B-树中进行删除:

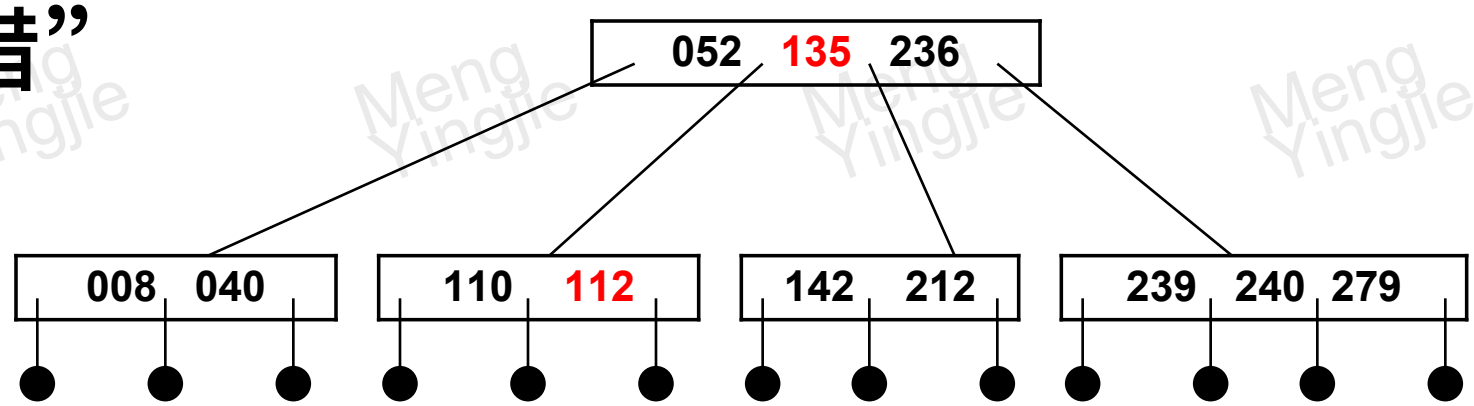
删除045



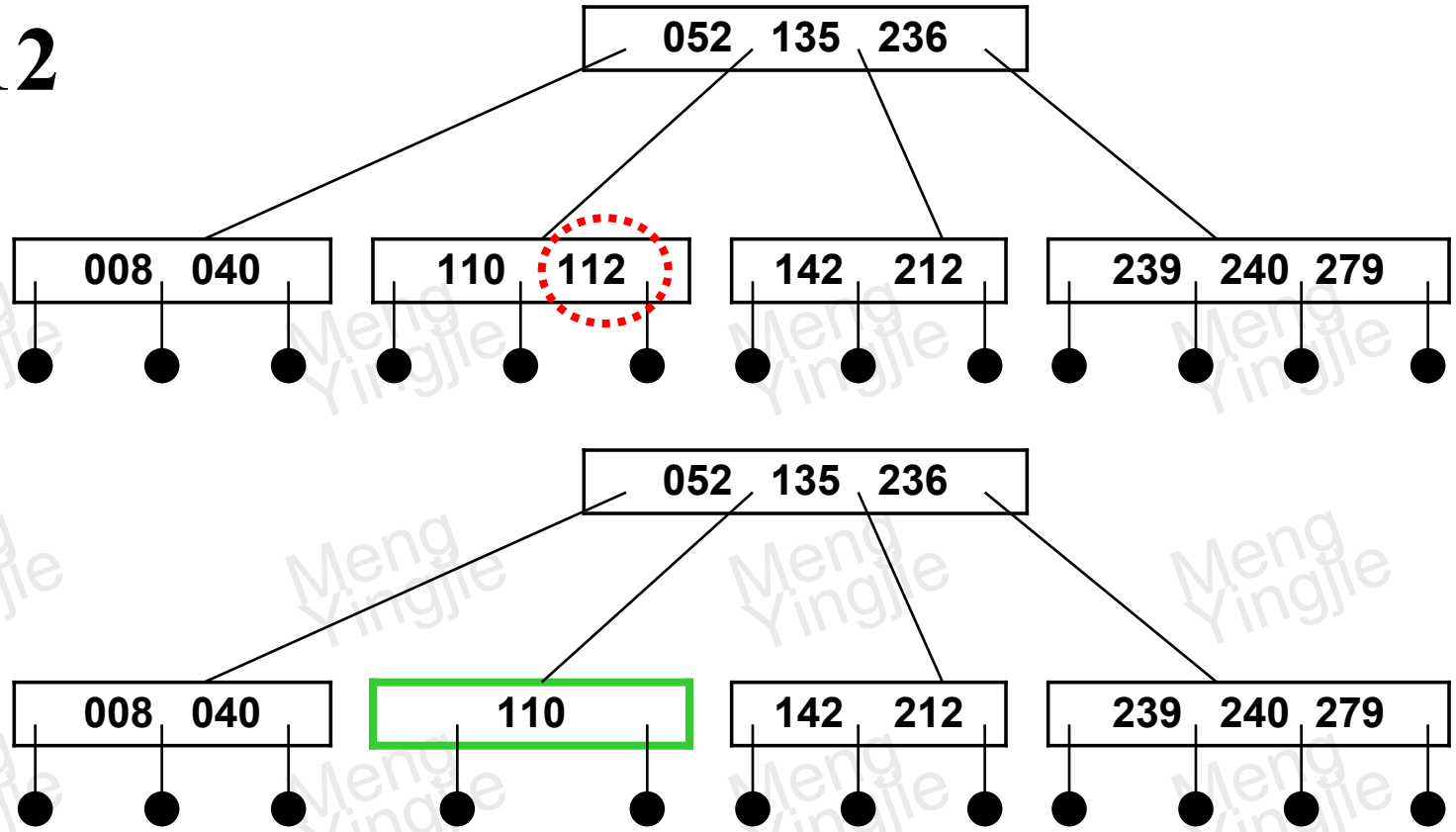
交换



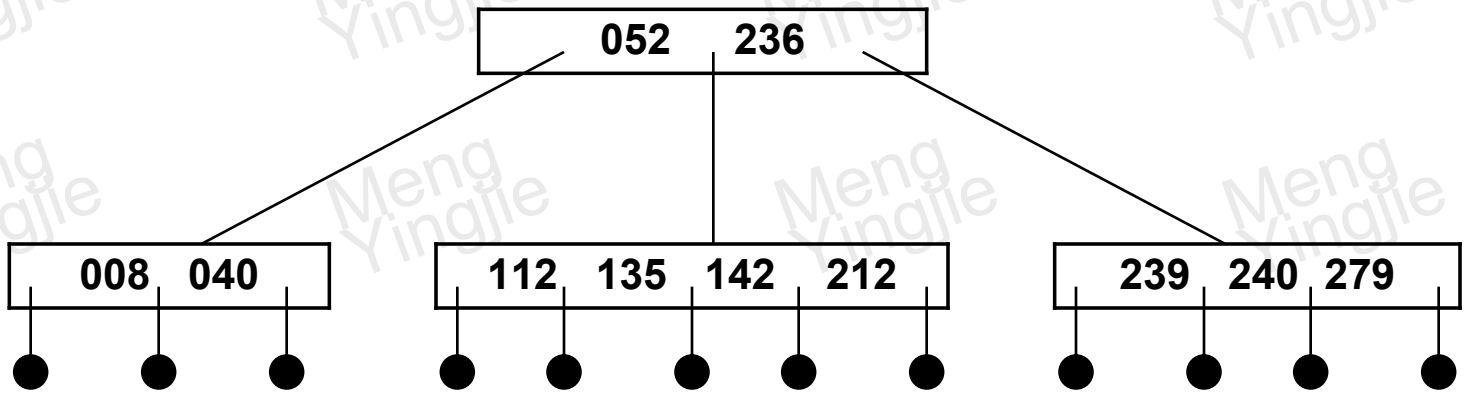
“借”



删除112



不能借，需合并



3、B+树

在实际应用中B-树产生了许多变形。其中B+树就是应用最广泛的一种B-树的变形。

B+树的定义:

B+树是B-树的变形,可以在叶结点存储关键字信息。

一棵m阶的B+树满足如下条件:

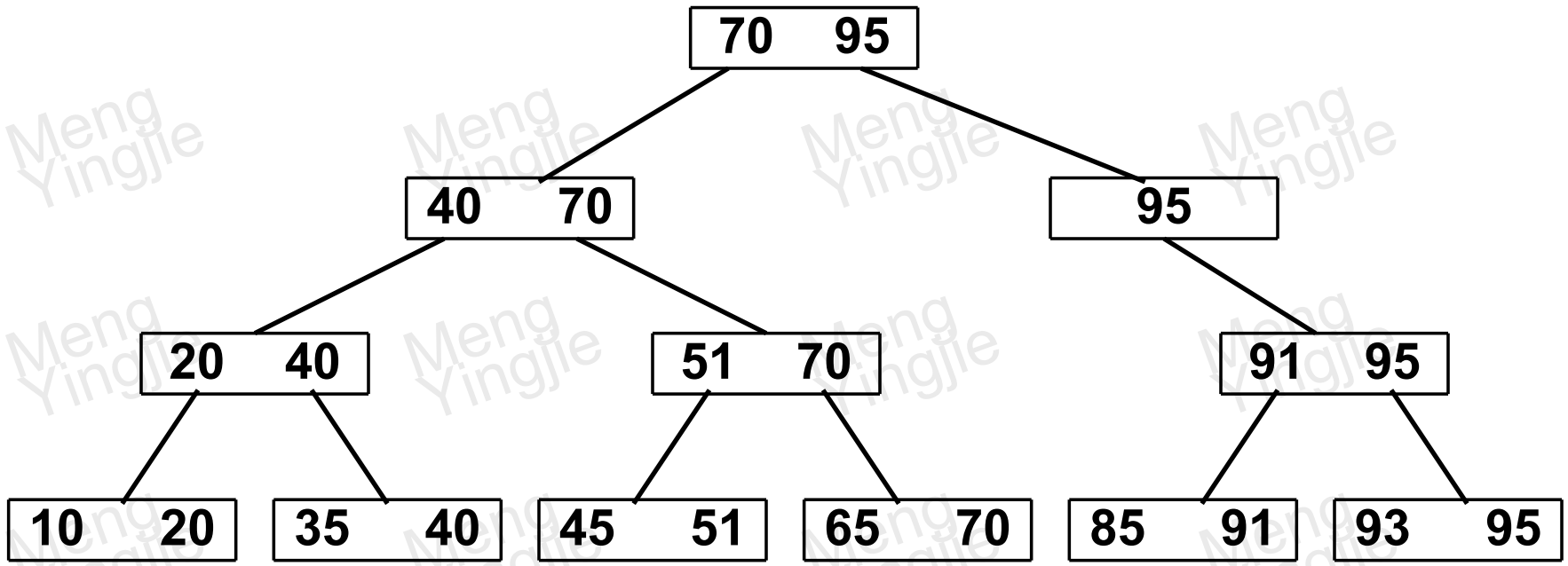
①每个结点至多有(≤) m个孩子结点;

②每个结点(除根之外)至少有(≥)⌈m/2⌉个孩子结点;

③根结点至少要有两个孩子结点;

④具有k个孩子的结点必有k个关键字;

例，m=2的B+树:



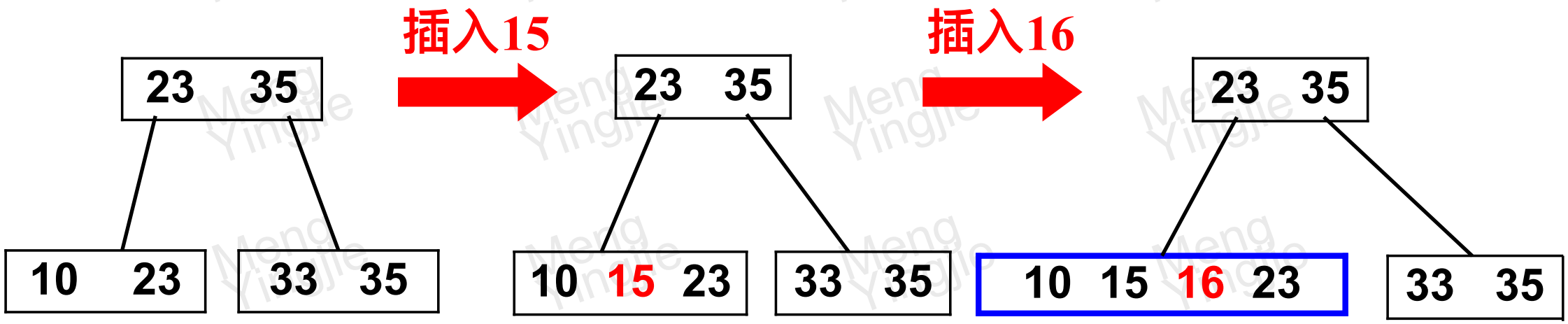
所有关键字均出现在叶结点上,其构造是由下而上的,m限定了结点的大小,其上层结点均是下层中最大关键字的复写(当然也可以采用最小关键字复写原则)。

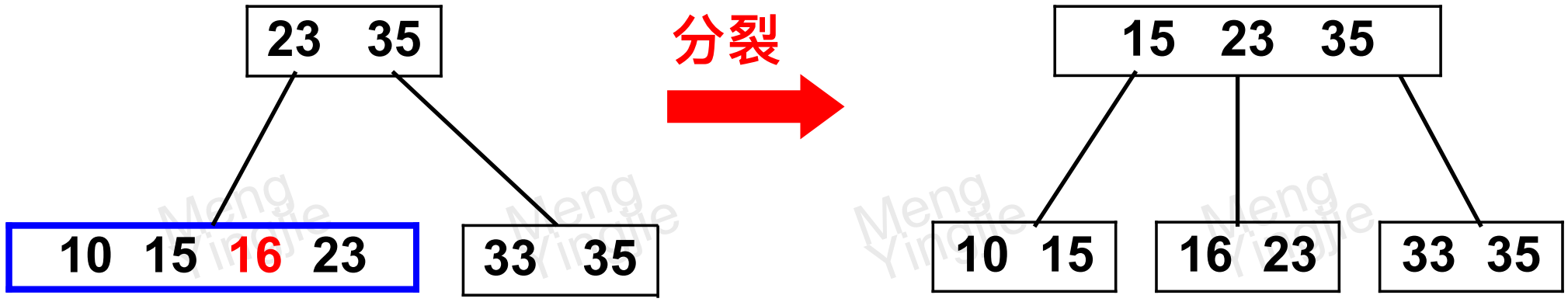
B+树的运算与B-树类似。

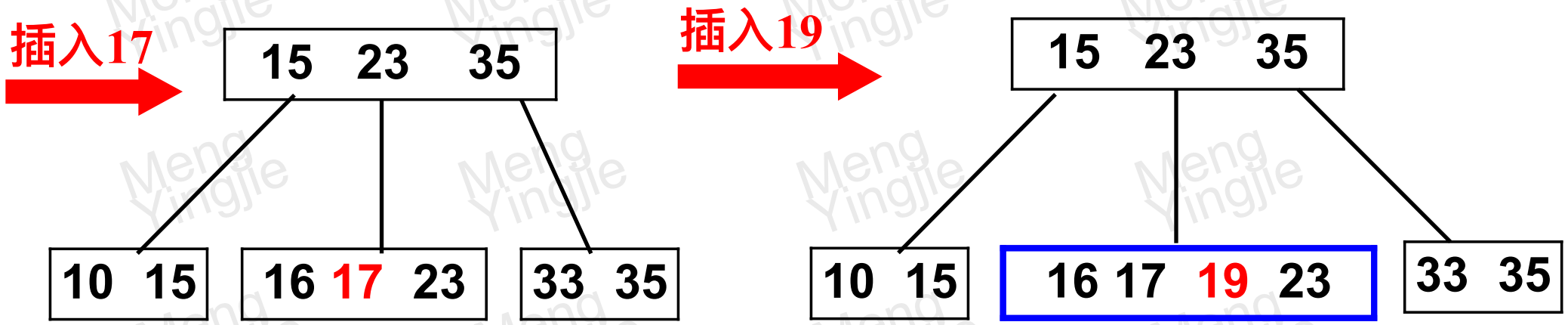
例：B+树中的插入：

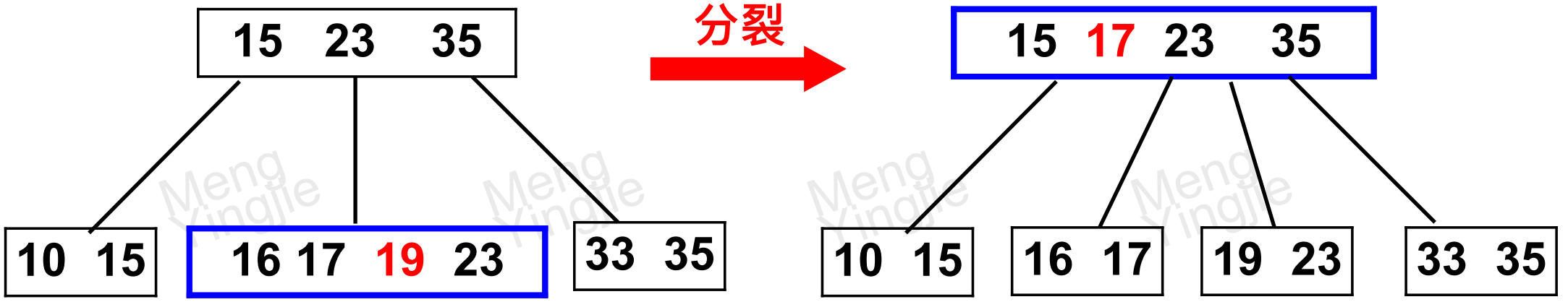
与B+树类似，插入较为简单，同样可能带来B+树的生长，但要注意上层结点是下层结点的复写。

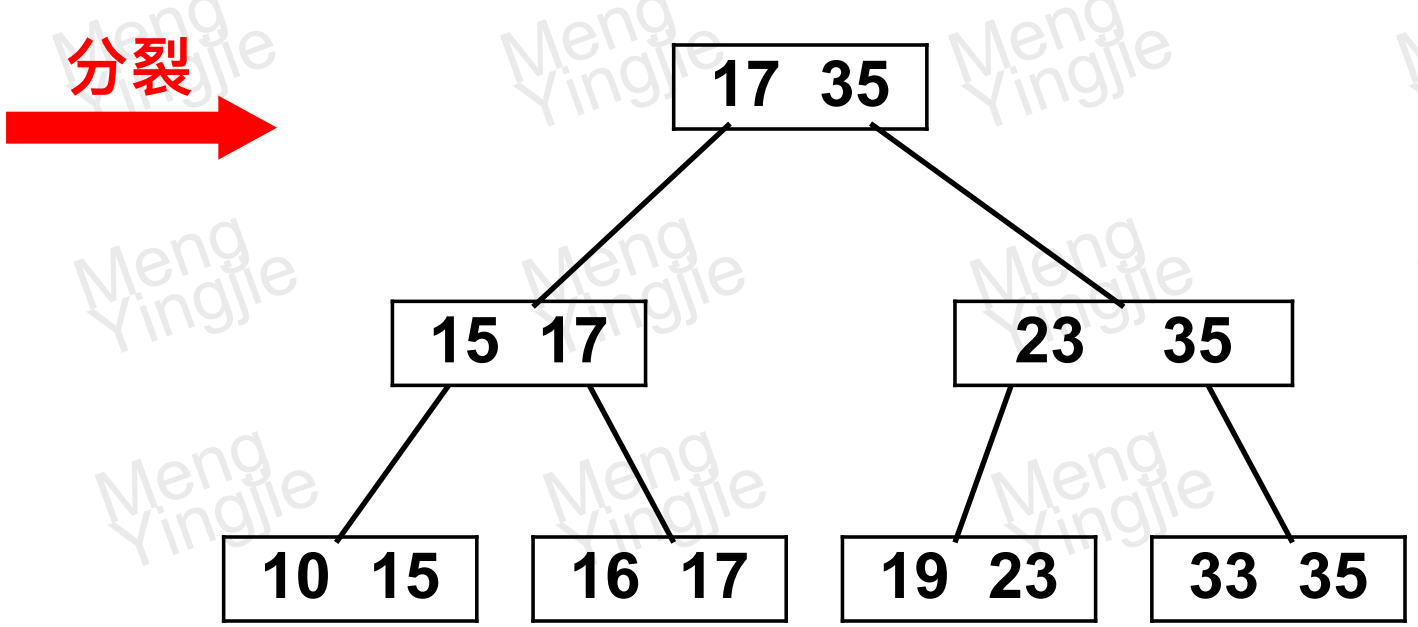
m=3的B+树中进行插入：




23 35
10 15 16 23
33 35
分裂
15 23 35
10 15
16 23
33 35


插入17
15 23 35
10 15
16 17 23
33 35
插入19
15 23 35
10 15
16 17 19 23
33 35


15 23 35
分裂
15 17 23 35
10 15
16 17 19 23
33 35
10 15
16 17
19 23
33 35


分裂
17 35
15 17
23 35
10 15
16 17
19 23
33 35

B+树的删除与B-树类似，但关键字在叶结点层删除后，其在上层的副本可以保留，以便作为一个“分界”关键存在。

若因为删除导致结点中的关键字个数小于⌈m/2⌉，其处理方法与B-树类似。

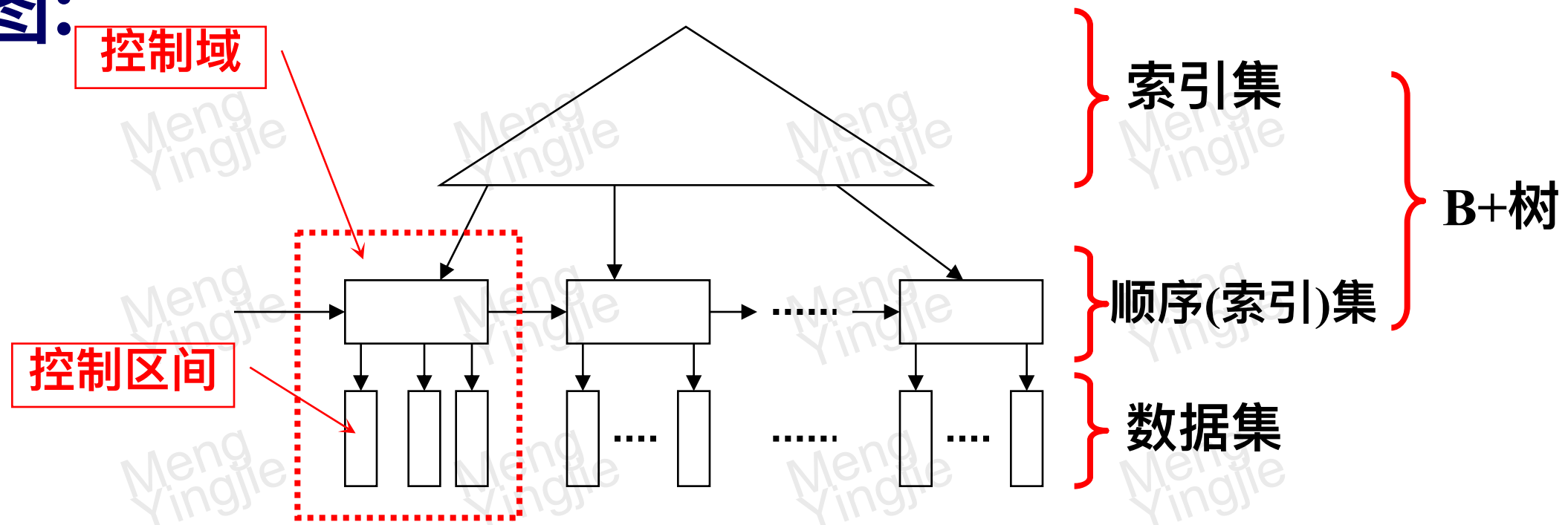
4、VSAM文件:

虚拟存储存取法(VSAM，Virtual Storage Access Method)是B+树应用的一个典型例子。它是IBM370系统中使用的一种文件存取方法。

它由三级索引集构成：索引集、顺序(索引)集和数据集。

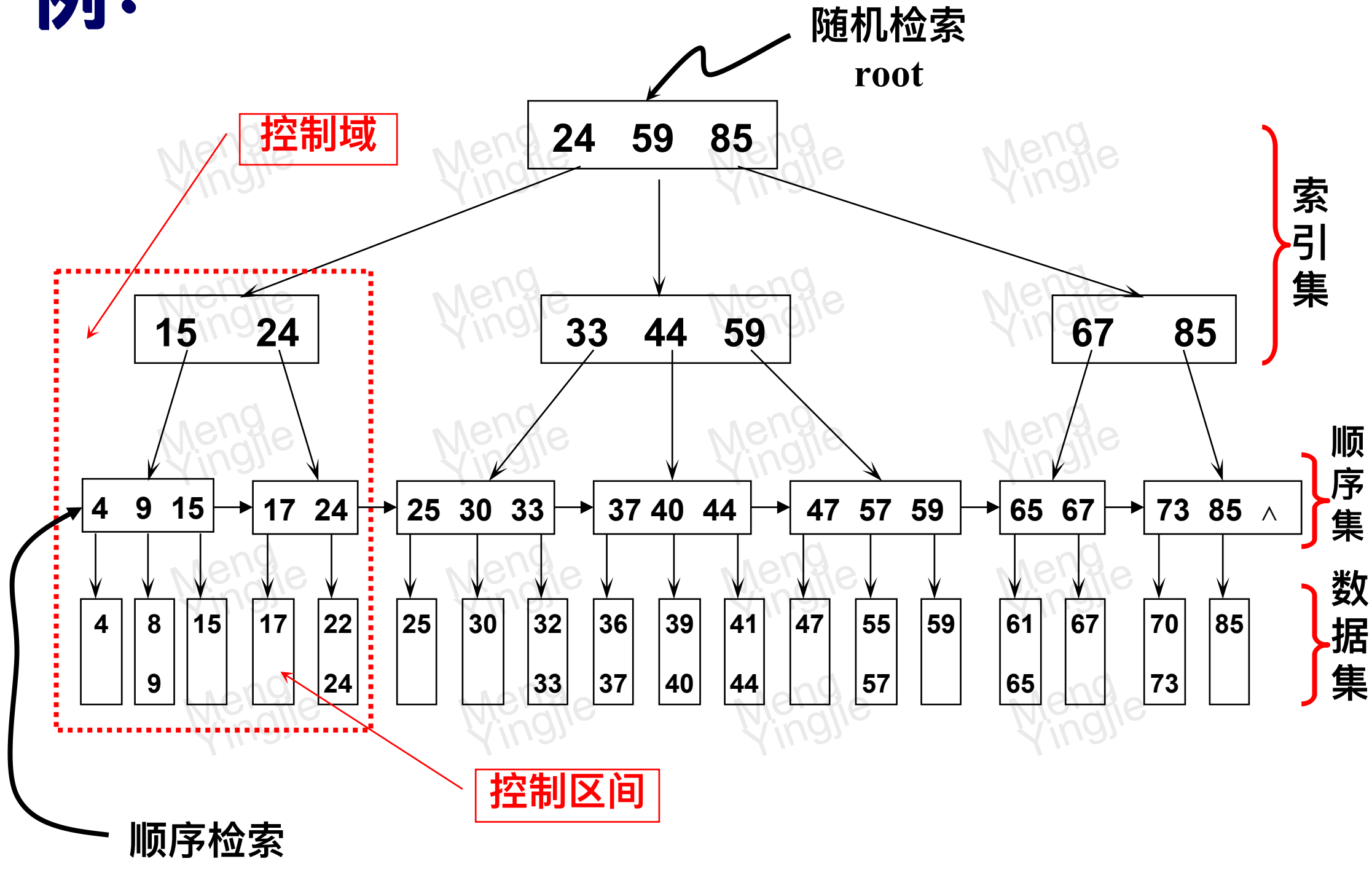
文件的记录均存放在数据集中，顺序(索引)集也是文件的一部分。索引集和顺序(索引)集一起形成一棵B+树结构的文件索引。

VSAM文件结构示意图:



数据存储划分为控制区域(Control area)，控制区域内再划分为控制间隔(也称为控制区间，是I/O中数据传输的基本单位，Contral Interval)。每个控制区域都有一个顺序集索引。顺序集索引的各个记录项指出各对应控制间隔的最大关键字值。

例：


随机检索
root
控制域
24 59 85
15 24
33 44 59
67 85
索引集
4 9 15
17 24
25 30 33
37 40 44
47 57 59
65 67
73 85 ∧
顺序集
4
8
9
15
17
22
24
25
30
32
33
36
37
39
40
41
44
47
55
57
59
61
65
67
70
73
85
数据集
控制区间
顺序检索

本节结束

一.概述

在第九章中讨论的排序技术都是针对较少量的记录而言的,对于记录的集合来说都是可以容纳于内存中的。

然而对于大量的数据而言,根据存储器的性质都是以文件的方式存放于外部存储器上,这些数据已经不可能全部装入到内存中。

因此对于这些数据的排序过程就要不断地进行内、外存数据交换并分阶段进行。

当然，外部排序的具体方法与数据所处的外存设备特征有关，但是最通用的方法是合并排序，这种方法基本上由两个阶段组成：

1.生成初始顺串：

首先用某种有效的内排序方法对文件的各段进行排序，这种经过排序的段通常称为顺串 (或归并段)，当它们生成后即写到外存上，这样外存上就形成了许多初始归并段。

2.合并初始顺串：

对于第一阶段生成的初始顺串使用某种归并方法(例如2-路归并)，进行多遍归并，最后形成整个文件的单一顺串。

以下以磁盘排序为例说明如何合并初始顺串，其过程类似于选择树。

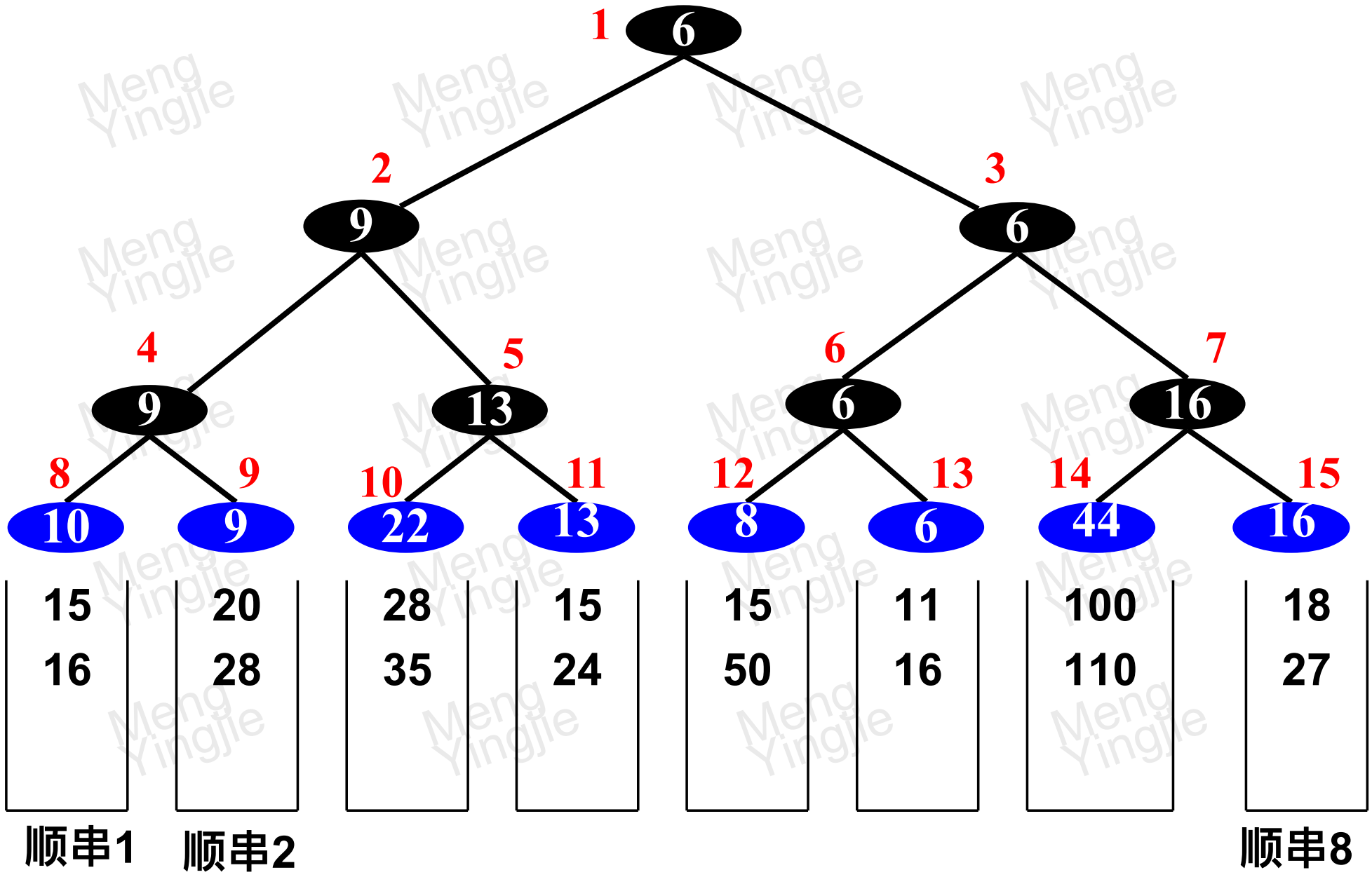
值得注意的是，外排序时必须要尽可能地减少内外存数据的交换次数，以提高I/O效率。

二.k–路归并

k-路归并方法的外部排序过程与2-路归并过程类似。

若开始有m个顺串，为了完成归并过程，与2-路归并排序的不同在于利用了选择树来辅助完成归并，即需要建立一个m个顺串的归并树。

例，8-路归并排序的选择树:



k–路合并的排序过程(该过程也称锦标赛过程):

1. 建立顺序存储的二叉树模型，其叶结点为每个顺串的第一元素(类似于建堆)；

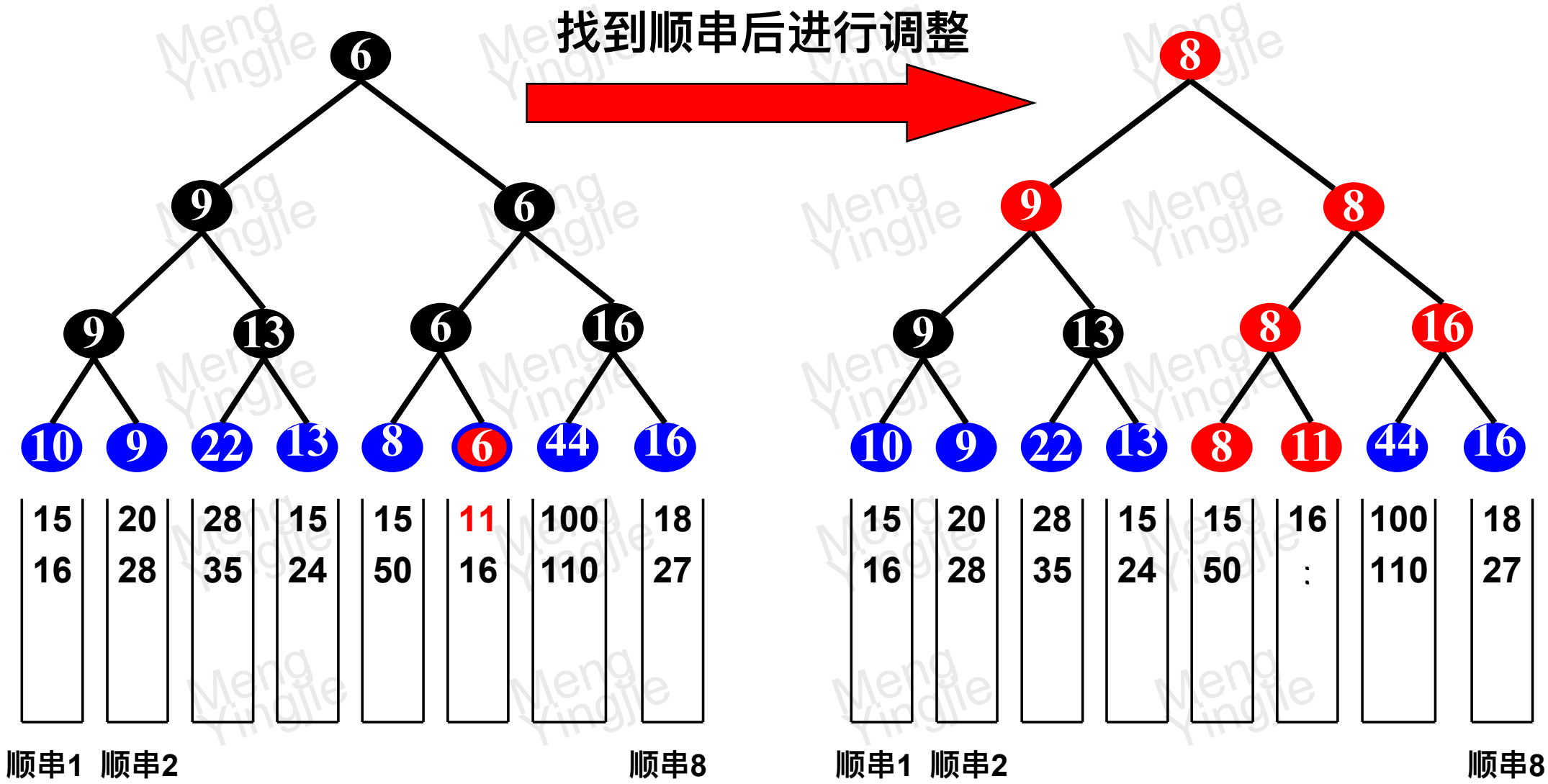
2. 删除根结点(将其写入另一个顺串)，查找根结点元素所在的顺串的序号；

3. 将此顺串的下一元素加入选择树中；

4. 调整选择树；

5. 重复2-4步。

例，输出根结点后的调整:


找到顺串后进行调整
6
9
6
9
13
6
16
10
9
22
13
8
6
44
16
15 16
20 28
28 35
15 24
15 50
11 16
100 110
18 27
顺串1
顺串2
顺串8
8
9
8
9
13
8
16
10
9
22
13
8
11
44
16
15 16
20 28
28 35
15 24
15 50
16 :
100 110
18 27
顺串1
顺串2
顺串8

上面的选择树进行的内部排序的时间将随着k的增大会稍微增加一些,这是因为k越大选择树越大,兄弟结点之间的比较、孩子结点与父结点之间的比较就越多。

也就是说,当输出根结点后,进行选择树的再构过程中,既要查找兄弟结点，又要查找父结点。为了减少这种重构的代价,可以使用败方树进行优化。

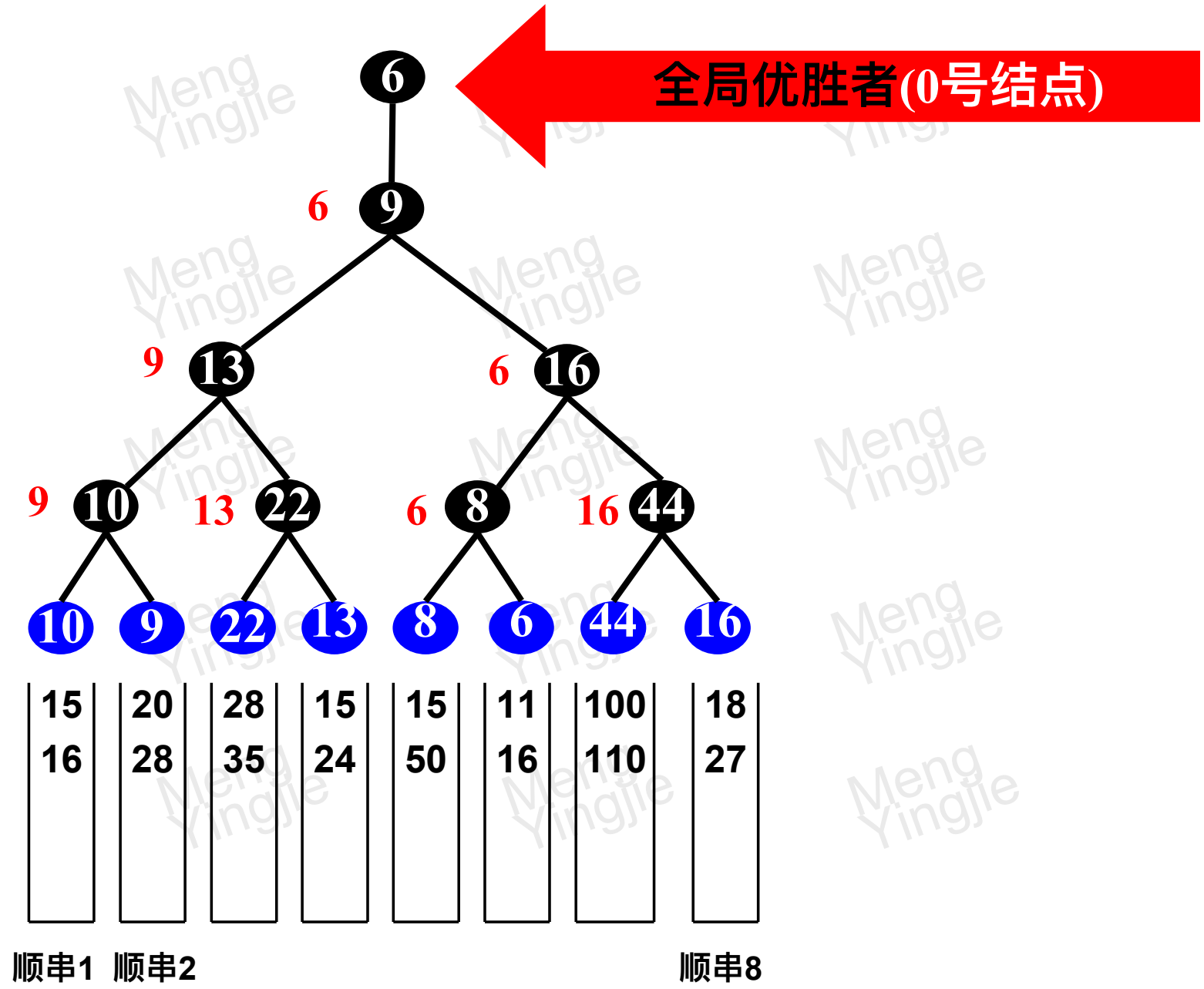
败方树可以通过对选择树的改造获得：

- 叶结点仍旧表示顺串的记录；
- 将选择树中非叶结点存储胜者改为存储败者(将胜者作为下一轮的比较结点)；
- 增加一个0号结点，以便保存全局优胜者.

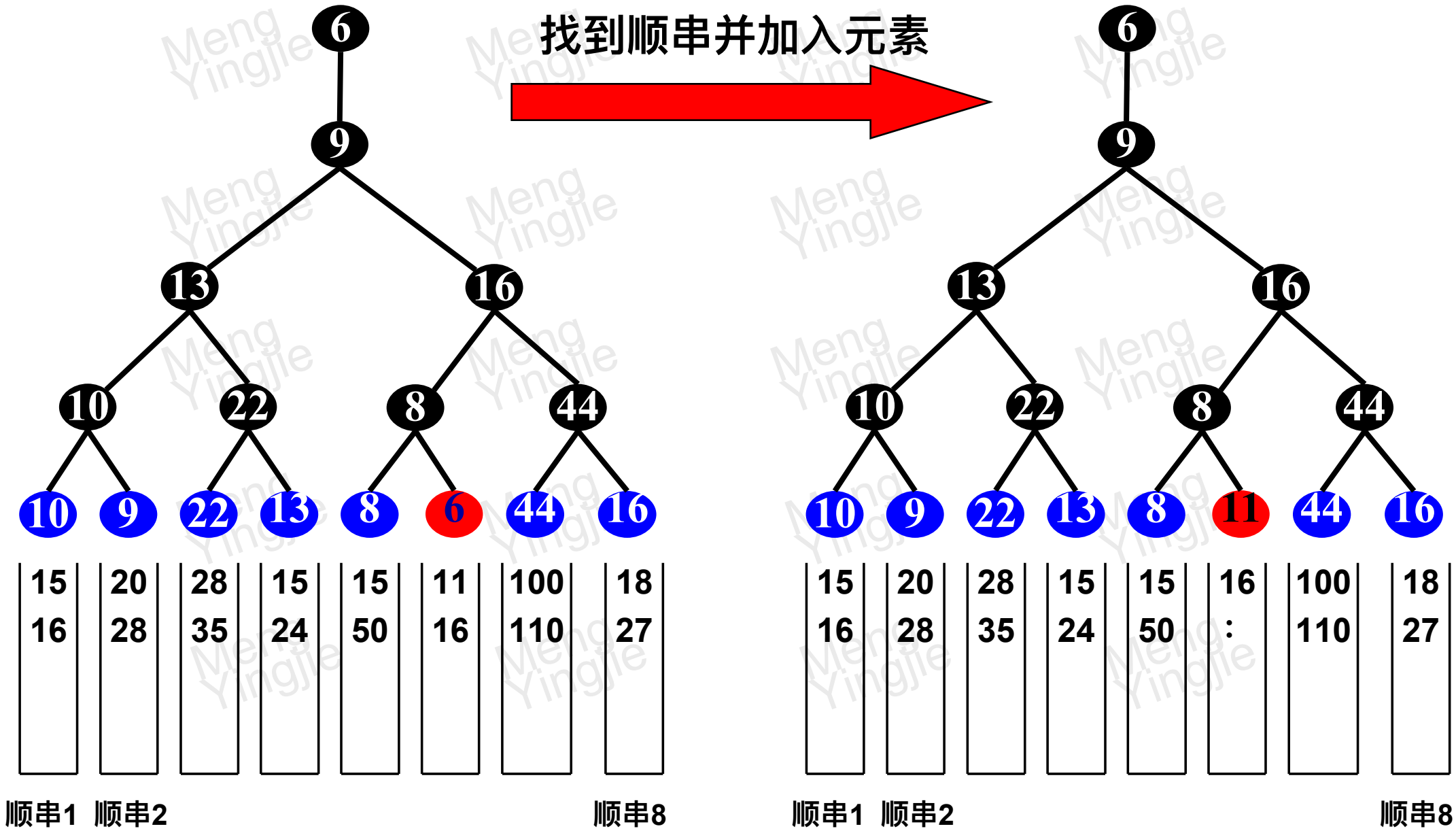
经这样改造得到的结构称为败方树(或败者树)。

在败方树中所进行的优化就是，对结构调整时不再需要进行兄弟结点间的比较，因此可以改善因k的越大而内部处理时间会增大的不足。

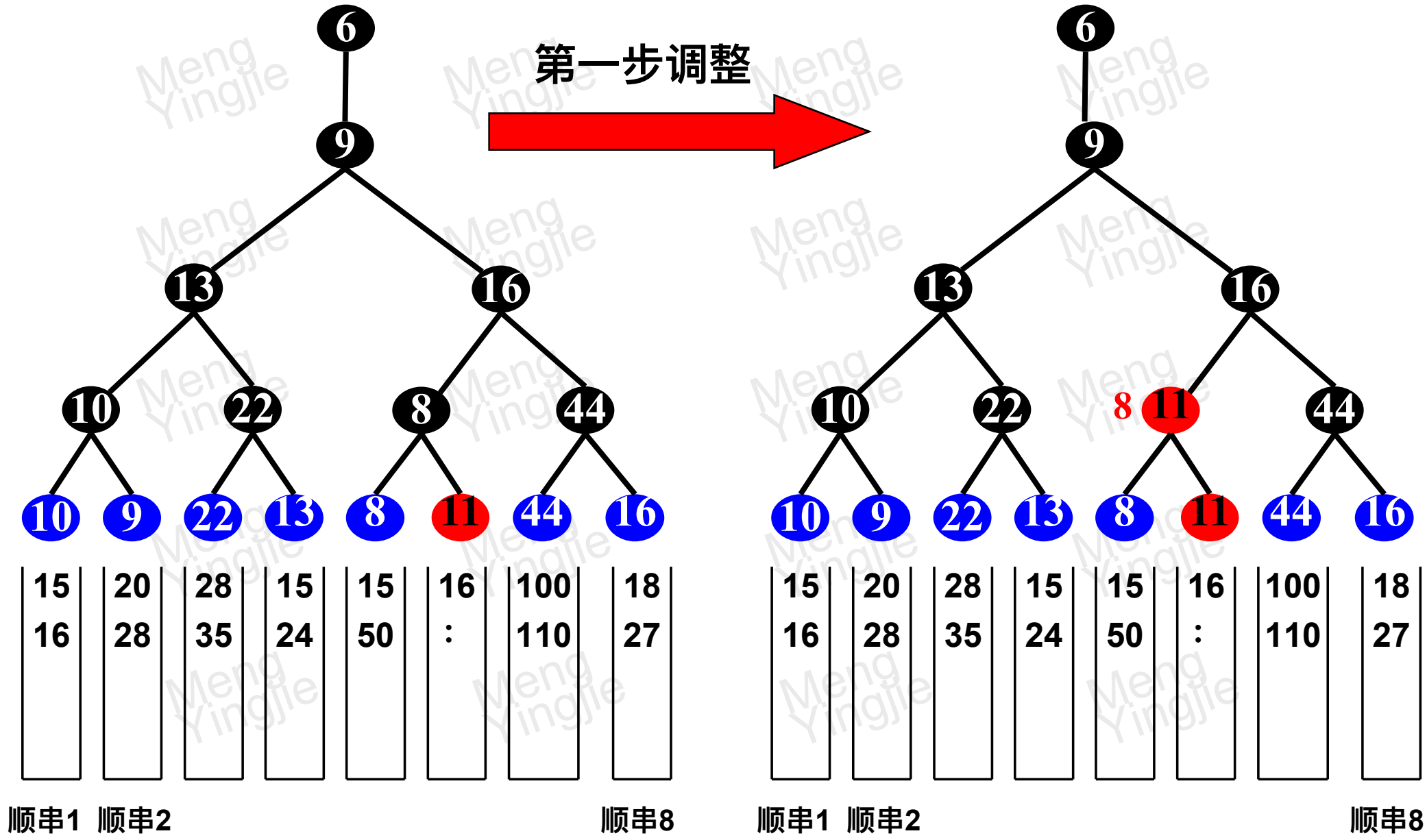
例，败方树的构造:


全局优胜者(0号结点)
6
6 9
9 13 6 16
9 10 13 22 6 8 16 44
10 9 22 13 8 6 44 16
15 16
20 28
28 35
15 24
15 50
11 16
100 110
18 27
顺串1
顺串2
顺串8

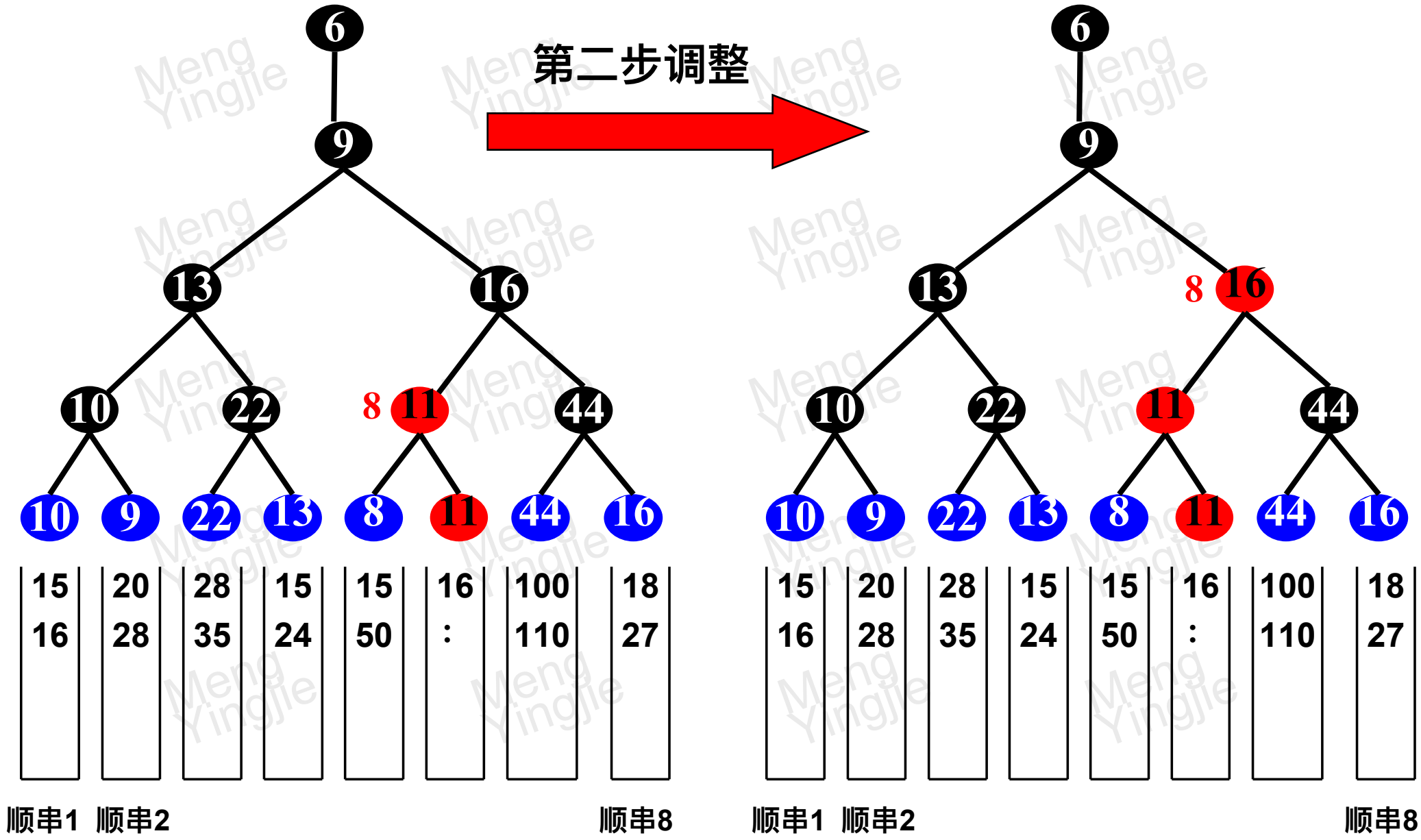
例，败方树的调整:


找到顺串并加入元素
6
9
13
16
10
22
8
44
10
9
22
13
8
6
44
16
15 16
20 28
28 35
15 24
15 50
11 16
100 110
18 27
顺串1
顺串2
顺串8
6
9
13
16
10
22
8
44
10
9
22
13
8
11
44
16
15 16
20 28
28 35
15 24
15 50
16 :
100 110
18 27
顺串1
顺串2
顺串8

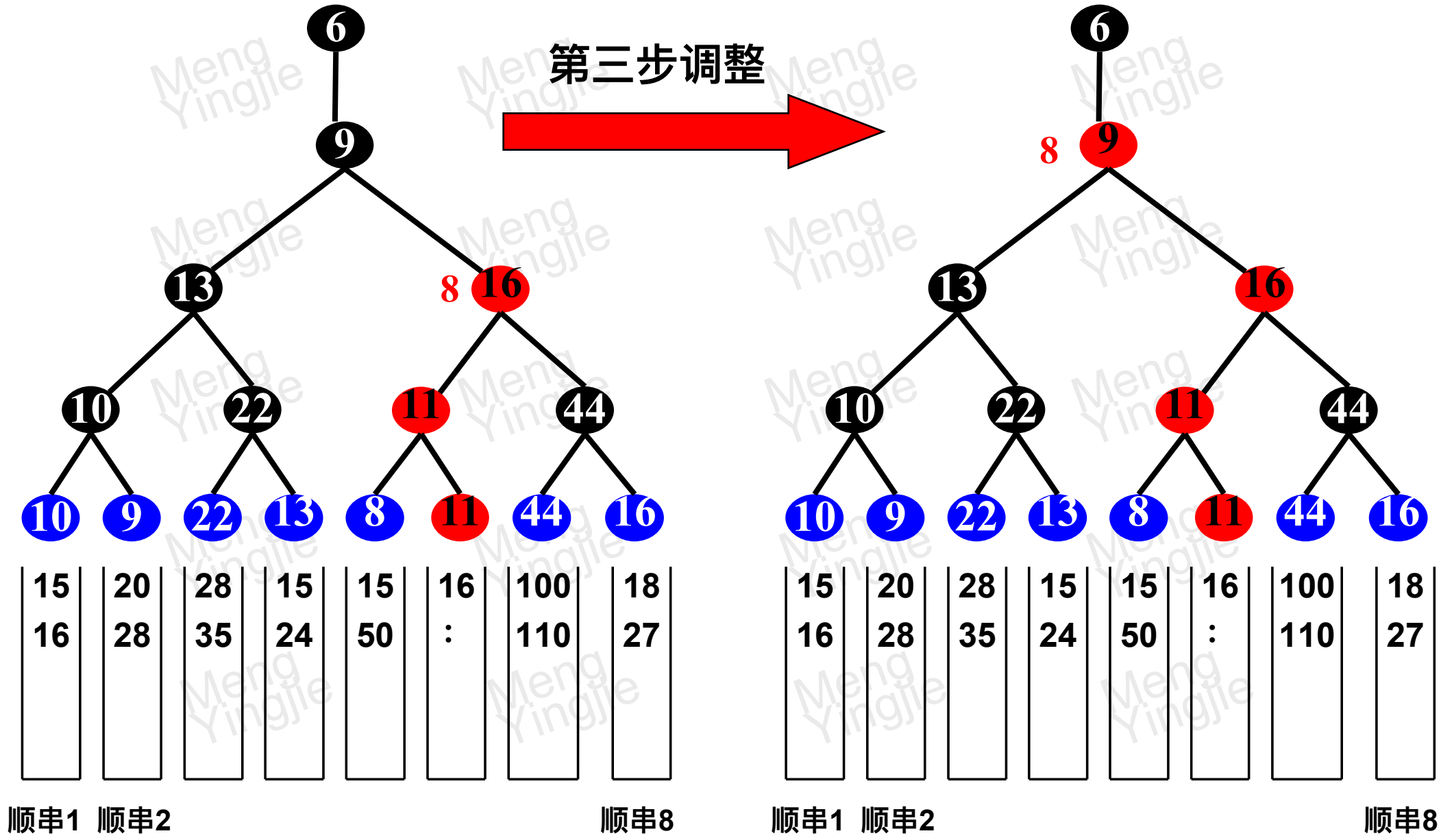
例，败方树的调整:


6
9
13
16
10
22
8
44
10
9
22
13
8
11
44
16
15 16
20 28
28 35
15 24
15 50
16 :
100 110
18 27
顺串1
顺串2
顺串8
第一步调整
6
9
13
16
10
22
8 11
44
10
9
22
13
8
11
44
16
15 16
20 28
28 35
15 24
15 50
16 :
100 110
18 27
顺串1
顺串2
顺串8

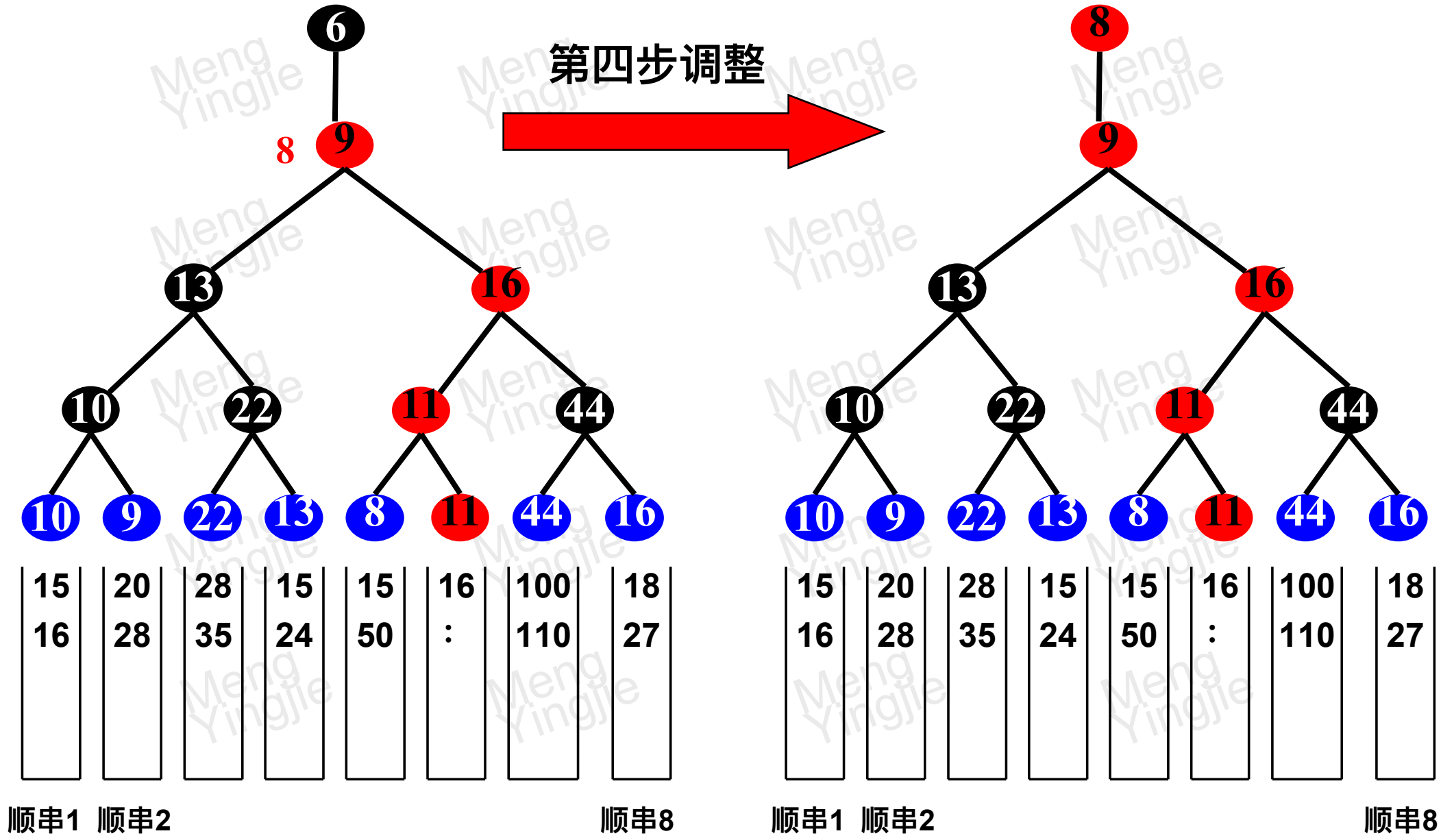
例，败方树的调整:


第二步调整
6
9
13
16
10
22
8
11
44
10
9
22
13
8
11
44
16
15 16
20 28
28 35
15 24
15 50
16 :
100 110
18 27
顺串1
顺串2
顺串8
6
9
13
8
16
10
22
11
44
10
9
22
13
8
11
44
16
15 16
20 28
28 35
15 24
15 50
16 :
100 110
18 27
顺串1
顺串2
顺串8

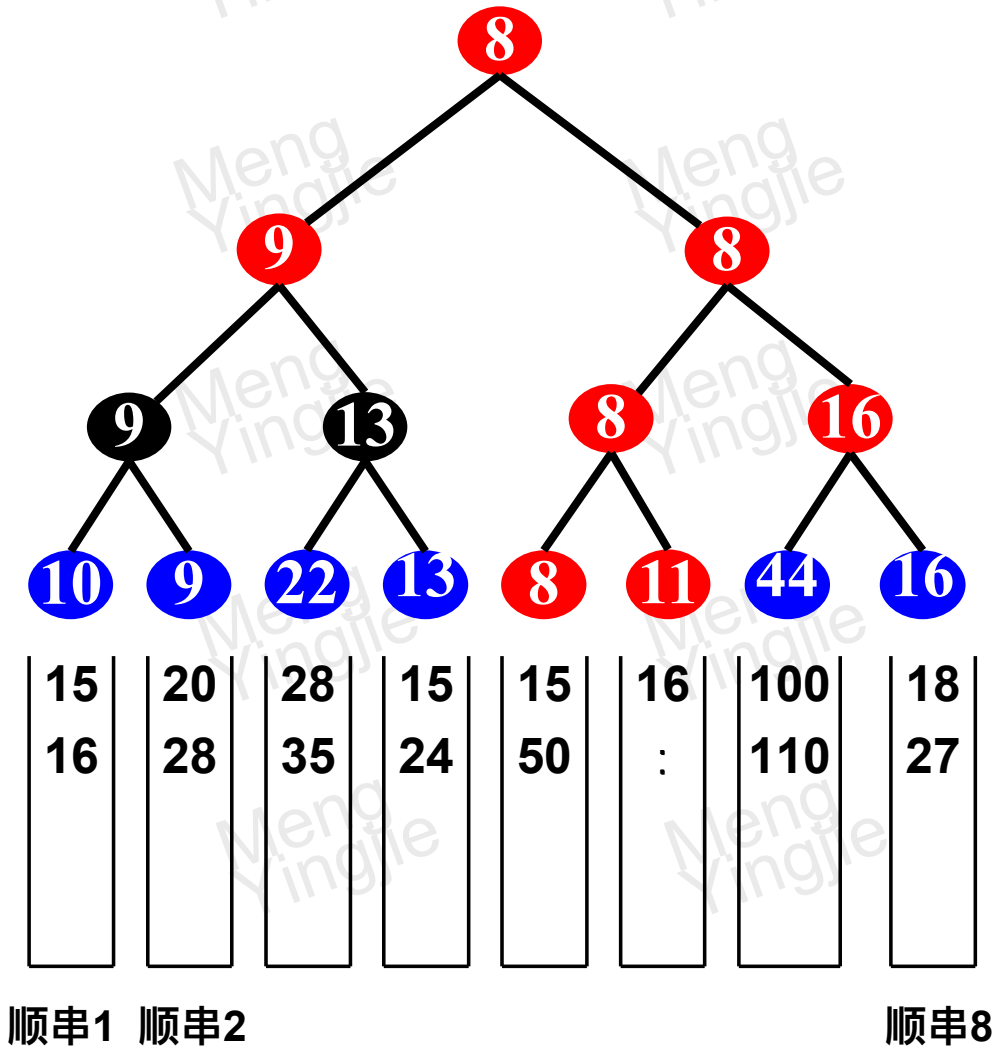
例，败方树的调整:

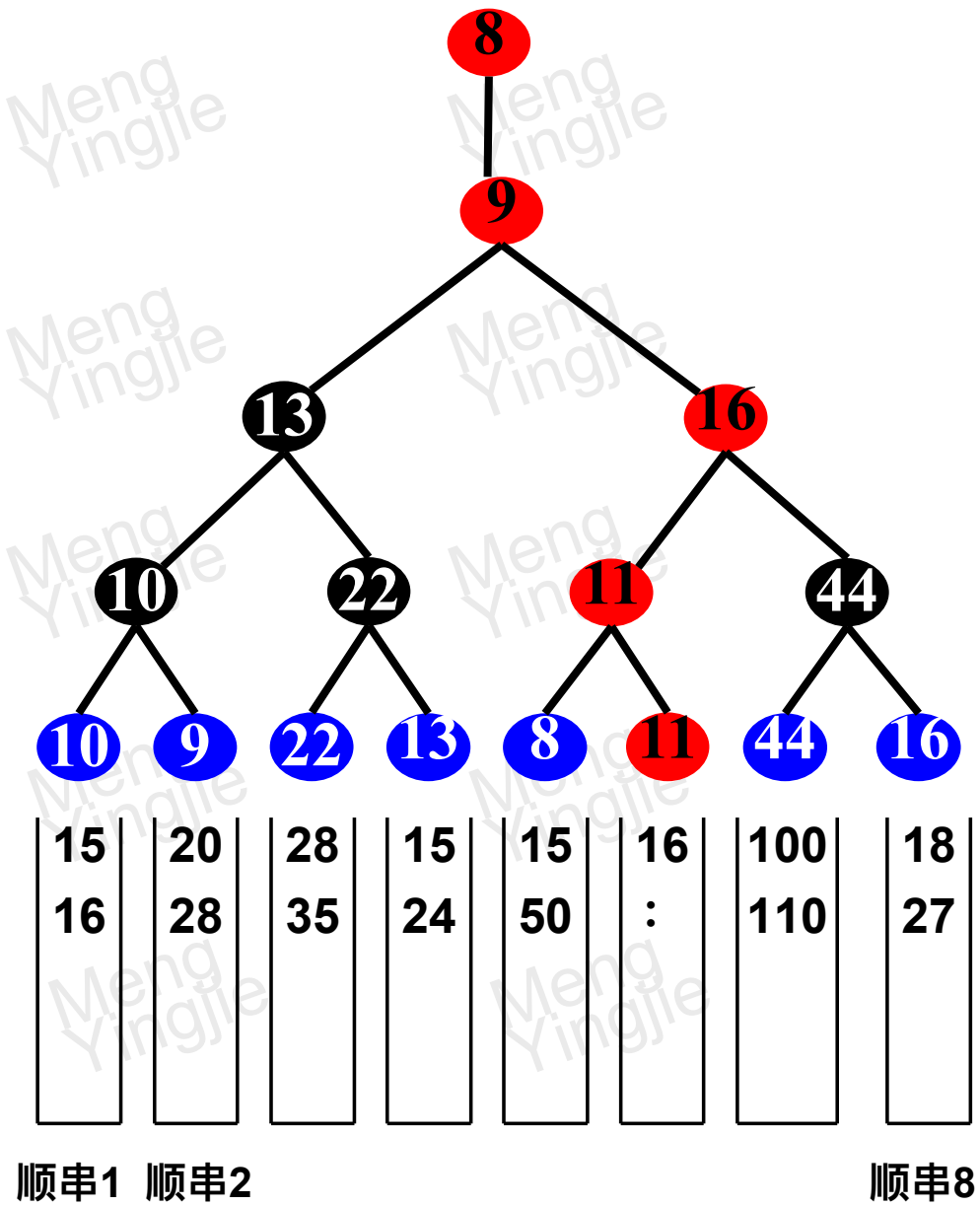

第三步调整
6
9
13
8
16
10
22
11
44
10
9
22
13
8
11
44
16
15 16
20 28
28 35
15 24
15 50
16 :
100 110
18 27
顺串1
顺串2
顺串8
6
8
9
13
16
10
22
11
44
10
9
22
13
8
11
44
16
15 16
20 28
28 35
15 24
15 50
16 :
100 110
18 27
顺串1
顺串2
顺串8

例，败方树的调整:


6
8
9
13
16
10
22
11
44
10
9
22
13
8
11
44
16
15 16
20 28
28 35
15 24
15 50
16 :
100 110
18 27
顺串1
顺串2
顺串8
第四步调整
8
9
13
16
10
22
11
44
10
9
22
13
8
11
44
16
15 16
20 28
28 35
15 24
15 50
16 :
100 110
18 27
顺串1
顺串2
顺串8

例，选择树与败方树的调整比较:





本节结束