数据结构与算法

第9章文件管理和外排序

主要内容

- 9.1 主存储器和外存储器
- 9.2 文件的组织和管理
- 9.3 外排序
- 9.4 文件管理和外排序知识点总结

9.1 主存储器和外存储器

- 计算机存储器主要有两种：
  - 主存储器(primary memory或者main memory，简称“内存”，或者“主存”)
    - 随机访问存储器(Random Access Memory, 即RAM)
    - 高速缓存(cache)
    - 视频存储器(video memory)
  - 外存储器(peripheral storage或者secondary storage，简称“外存”)
    - 硬盘
    - 软盘
    - 磁带

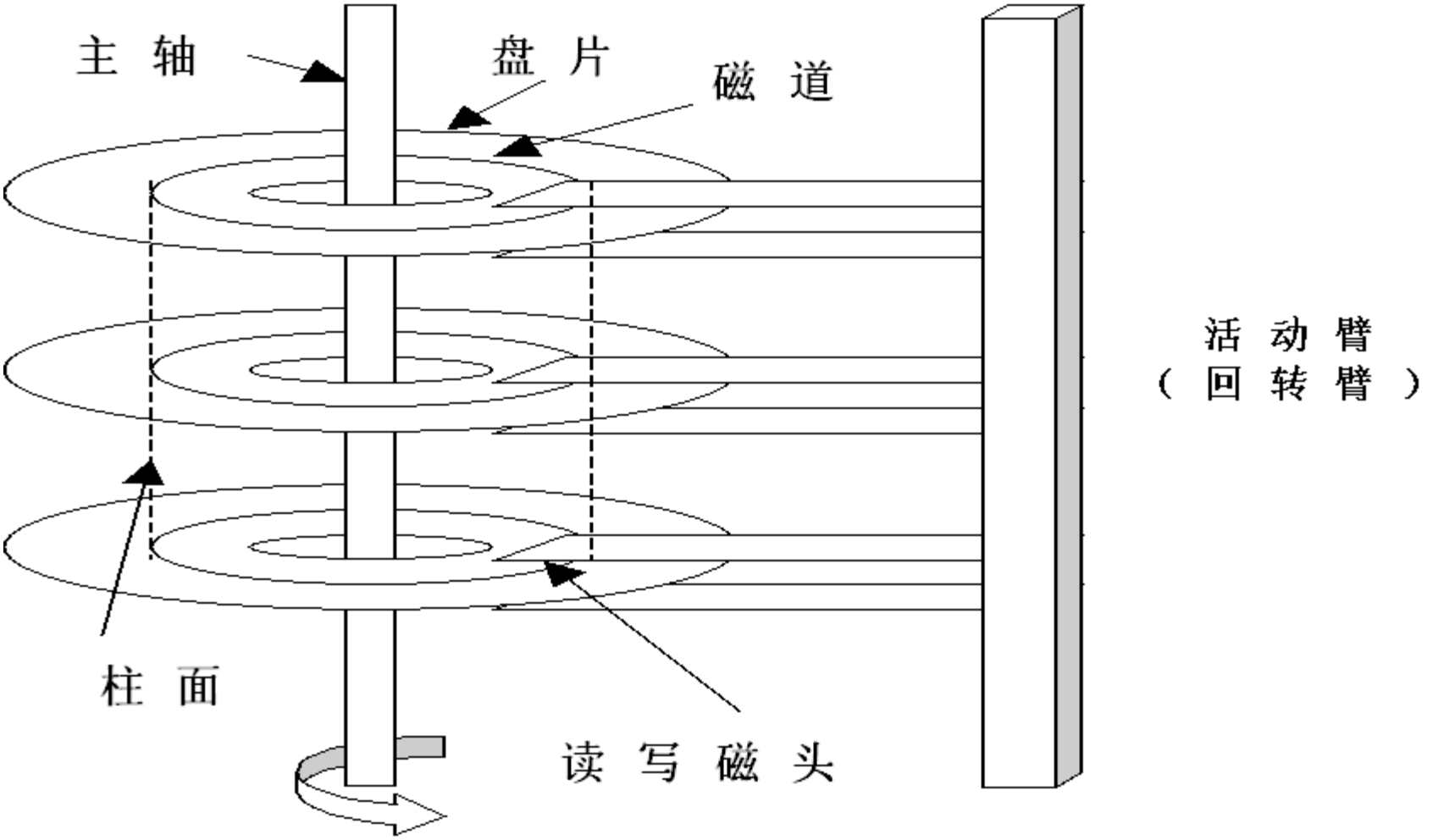
外存的优缺点

- 优点：价格低、信息不易失、便携性
- 缺点：存取速度慢

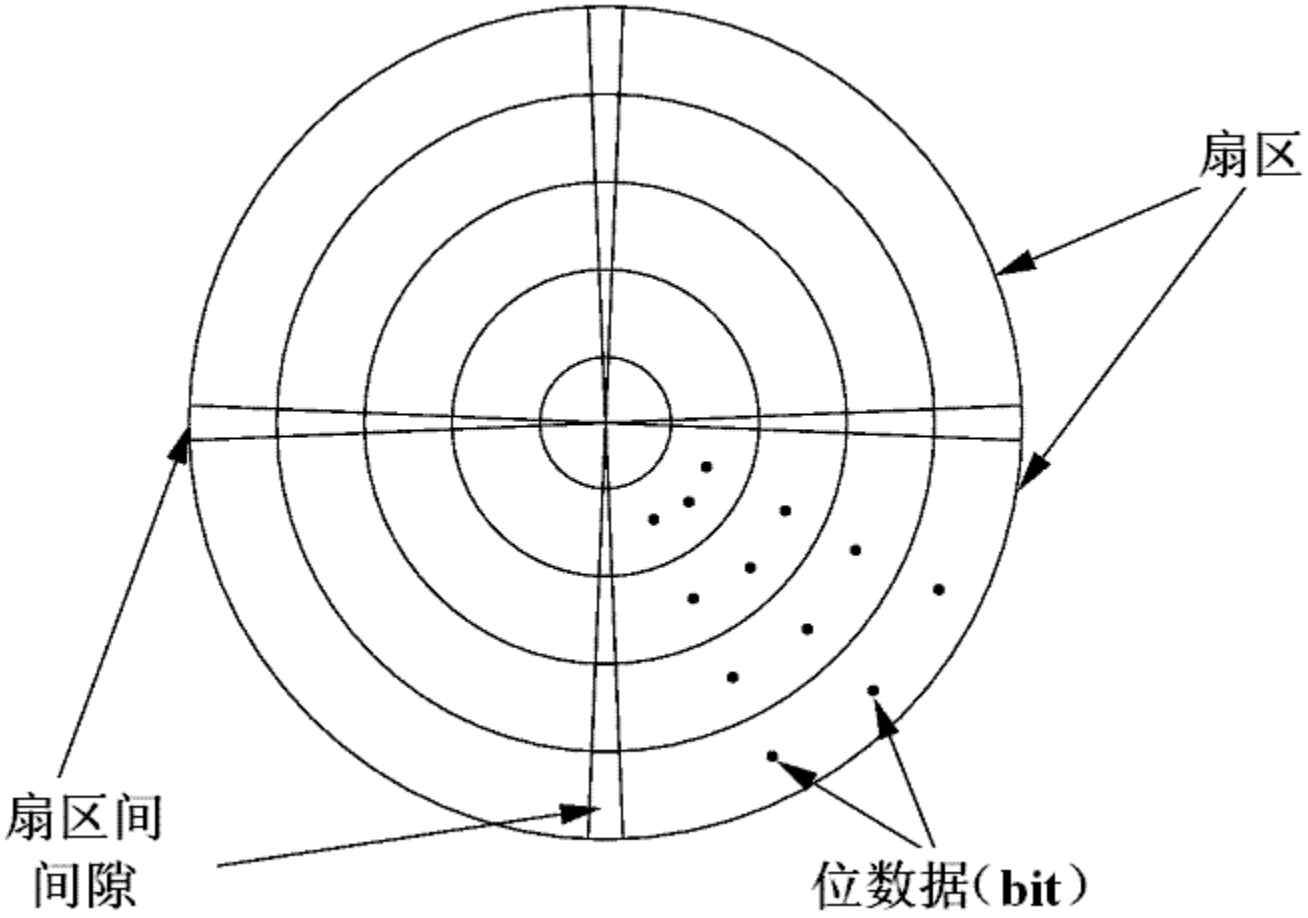
  一般的内存访问存取时间的单位是纳秒（1纳秒 = 10^{-9}秒），而外存一次访问时间则以毫秒（1毫秒 = 10^{-3}秒）或秒为数量级。

- 牵扯到外存的计算机程序应当尽量减少外存的访问和存取次数，从而减少程序执行的时间

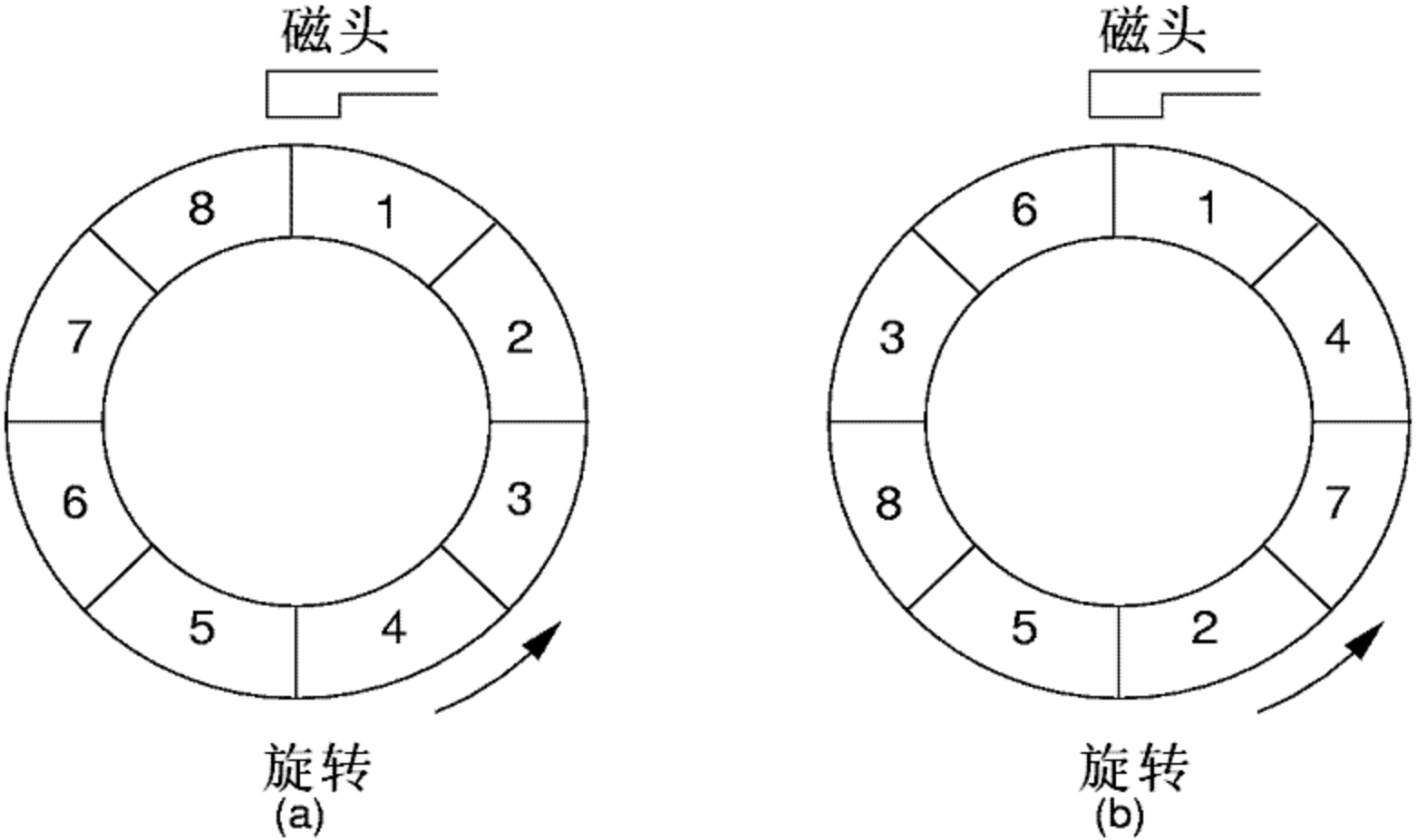
磁盘的物理结构



磁盘盘片的组织



磁盘磁道的组织（交错法）



（a）没有扇区交错；（b）以3为交错因子

磁盘存取步骤

- 选定某个盘片组
- 选定某个柱面
  - 这需要把磁头移动到该柱面，这个移动过程称为寻道(seek)
- 确定磁道
- 确定所要读写的数据在磁盘上的准确位置
  - 这段时间一般称为旋转延迟(rotational delay 或者rotational latency)
- 真正进行读写

内存的优缺点

- 优点：访问速度快
- 缺点：造价高、存储容量小和断电后丢失数据
- CPU直接与主存沟通，对存储在内存地址的数据进行访问时，所需要的时间可以看作是一个很小的常数

外存数据访问方式

- 外存空间被划分成长度固定的存储块，称为页(page)，每一页包含一定数量的数据单元
- 作为外存的基本存储单位，数据以页块为单位进行存取，这样可以减少外存的定位次数，从而减少外存读写的时间耗费

9.2 文件的组织和管理

- 文件(file)是存储在外存上的数据结构，是由大量性质相同的记录(record)组成的集合。
- 所谓记录，就是具有独立逻辑意义的数据块，是文件的基本数据单位。
- 最简单的记录可以是字符或者二进制序列，复杂的记录通常可以由若干字段或域(field)的数据项组成。

9.2 文件的组织和管理

按照记录的类型，文件可以分成两种：

- 操作系统的文件

  操作系统的文件是一组连续的字符序列，这种序列没有明显的结构。用户也可以将文件中的信息划分成若干个逻辑记录，以便存取和使用。

- 数据库文件

  数据库文件是有结构的记录集合，其中每一条记录都由一个或多个数据项组成，而每个数据项是不可再分的基本数据单元。

9.2 文件的组织和管理

如表9.1所示为一个数据库文件，每个学生的信息组成一个记录，每一条记录由6个数据项构成。

姓名	学号	性别	出生年月	所在院系	入学时间
贾由	00646125	男	1988.5	数学	2006.9.1
陈醒	00648308	男	1989.7	计算机	2006.9.1
吴轲	00648230	男	1988.3	计算机	2006.9.1
王子琪	00648139	女	1988.2	计算机	2006.9.1
……	……	……	……	……	……

9.2 文件的组织和管理

按照记录信息长度，文件可以分成

- 定长文件

  如果文件中每一条记录均含有相同的信息长度，那么这种文件称为定长文件。通常定长文件处理起来比较方便。

- 不定长文件

  若文件中的记录不是相等长度的，则称为不定长文件。

9.2 文件的组织和管理

按照关键码(key)的个数，文件可以分成：

- **单关键码文件**

  单关键码文件是指文件的记录中只有一个标识关键码

- **多关键码文件**

  多关键码记录文件中，记录除了有一个主关键码以外还允许有若干个次关键码

9.2 文件的组织和管理

- 文件的操作有实时和批量两种处理方式
  - 实时操作要求有较短的应答响应时间，在接受指令后尽可能快地完成检索或修改任务。
  - 批量的文件处理则允许较长反馈时间。用户可以根据需求选择不同的文件处理方式。

9.2 文件的组织和管理

- 9.2.1 文件组织
- 9.2.2 C++的流文件

9.2.1 文件组织

- 文件逻辑组织有以下三种形式：顺序结构的定长记录、顺序结构的变长记录和按关键码存取的记录。
- 文件的物理结构可以有多种多样的组织方式。常见的物理组织结构有：
  - 顺序文件
  - 散列文件
  - 索引文件
  - 倒排文件

9.2.2 C++的流文件

- 文件流是以外存文件为输入输出对象的数据流。
- 文件流与文件不是同一个概念，文件流不是由若干个文件组成的流。文件流本身不是文件，而只是以文件为输入输出对象的流。

9.2.2 C++的流文件

- 标准输入输出流类包括istream，ostream和iostream类
  - istream是通用输入流和其它输入流的基类，支持输入
  - ostream是通用输出流和其它输出流的基类，支持输出
  - iostream是通用输入输出流和其它输入输出流的基类，支持输入输出
- 3个用于文件操作的文件类
  - ifstream类，从istream类派生，用来支持从磁盘文件的输入
  - ofstream类，从ostream类派生，用来支持向磁盘文件的输出
  - fstream类，从iostream类派生，用来支持对磁盘文件的输入和输出

9.2.2 C++的流文件

下面是fstream类的一些主要成员函数：

```
#include <fstream.h>                    // fstream = ifstream+ofstream
void fstream::open(char*name, openmode mode);      // 打开文件进行处理
fstream::read(char*ptr, int numbytes);     // 从文件当前位置读入一些字节
fstream::write(char*ptr, int numbtyes);    // 向文件当前位置写入一些字节
// seekg和seekp：在文件中移动当前位置，以便在文件中的任何位置读出或写入字节
fstream::seekg(int pos);                   // 输入时用于设置读取位置
fstream::seekg(int pos, ios::curr);
fstream::seekp(int pos);                   // 设置输出时的写入位置
fstream::seekp(int pos, ios::end);
void fstream::close();                     // 处理结束后关闭文件
```

9.2.2 C++的流文件

文件常见的三个基本操作：

- 把文件指针设置到指定位置
- 从文件的当前位置读取字节
- 向文件中的当前位置写入字节，都是围绕文件指针进行的

- 程序员对磁盘文件进行输入输出时，都需要管理标志文件当前位置的文件指针。

9.3 外排序

- 9.3.1 置换选择排序
- 9.3.2 二路外排序
- 9.3.3 多路归并——选择树

9.3 外排序

- 如果数据存放在外存文件中，则需要考虑外存特点，采用外存文件排序技术，简称外排序(external sort)。
- 需要根据内存的大小，将外存中的数据文件划分成若干段，每次把其中一段读入内存并用内排序方法进行排序。这些已排序的段或有序的子文件称为顺串或归并段(run)。

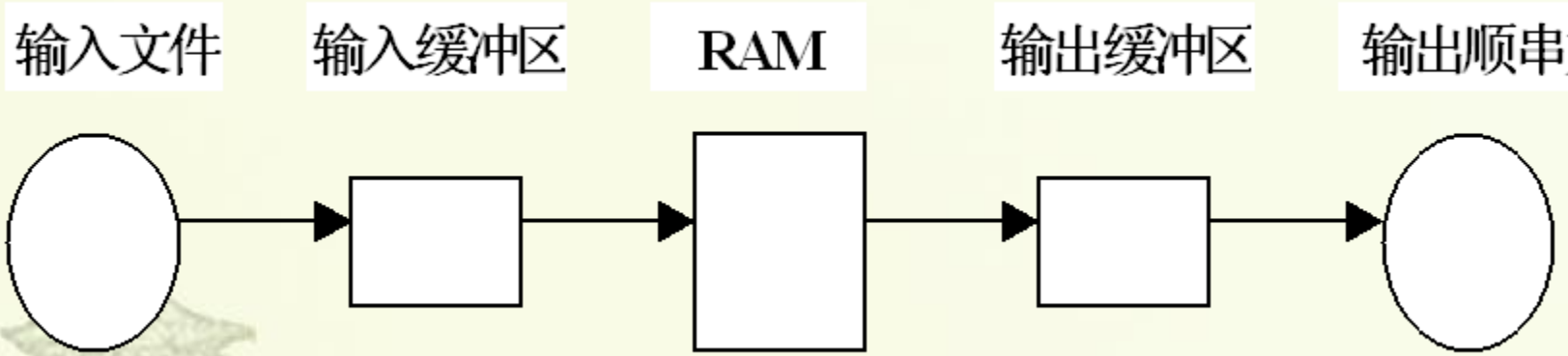
9.3 外排序

- 外排序通常由两个相对独立的阶段组成：
  - 文件形成尽可能长的初始顺串
  - 逐趟归并顺串，最后形成对整个数据文件的排列文件
- 外排序所需要的时间由三部分组成：
  - 内部排序所需要的时间
  - 外存信息读写所需要的时间
  - 内部归并所需要的时间
- 减少外存信息的读写次数是提高外部排序效率的关键

9.3 外排序

- 对同一个文件而言，进行外排序所需读写外存的次数与归并趟数有关系
- 假设有m个初始顺串，每次对k个顺串进行归并，归并趟数为$[\log_k m]$
- 为了减少归并趟数，可以从两个方面着手：
  - 减少初始顺串的个数m
  - 增加归并的顺串数量k

9.3.1 置换选择排序



- 算法的处理过程为：从输入文件读取一定数量的记录进入输入缓冲区；然后向内存工作区放入待排序记录并进行排序；记录被处理后，写到输出缓冲区；当输出缓冲区写满的时候，把整个缓冲区写回到外存文件。当输入缓冲区为空时，再次从外存文件中读取下一块记录。

9.3.1 置换选择排序

置换选择产生一个顺串的算法如下：

1、初始化堆：

（1） 从磁盘读M个记录放到数组RAM中；

（2） 设置堆尾指标LAST = M - 1；

（3） 建立一个最小堆。

2、重复以下步骤，直到堆为空（即LAST < 0）：

（1）把具有最小关键码值的记录(根结点)送到输出缓冲区；

9.3.1 置换选择排序

（2）设R是输入缓冲区中的下一条记录。判断R的关键码值是否大于刚刚输出的关键码值，

① 如果是，那么把R放到根结点。

② 否则，

(a) 使用数组中LAST位置的记录代替根结点；

(b) 把R放到LAST位置；

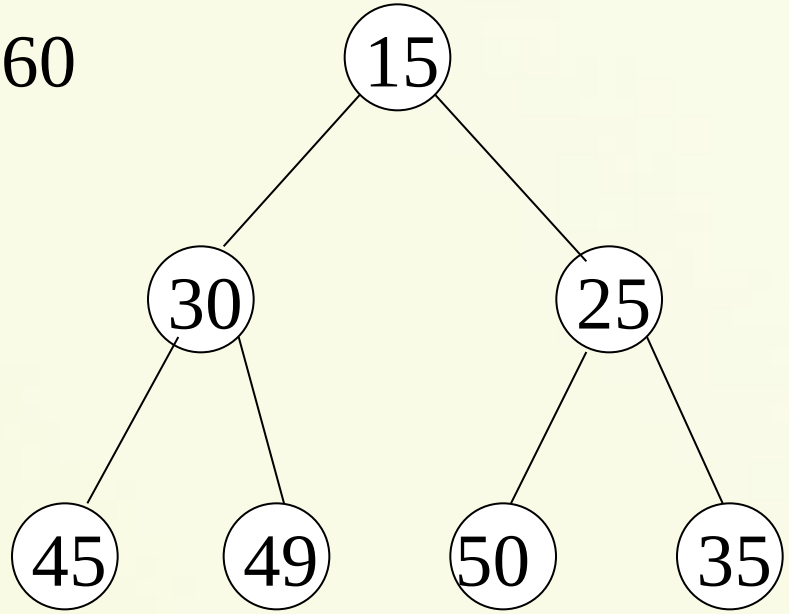
(c) 设置LAST = LAST - 1。

（3）重新排列堆，筛出根结点。

9.3.1 置换选择排序

置换选择示例

27 16 60



9.3.1 置换选择排序

【算法9.1】 置换选择算法

```
// 参数A是从外存读入数据后所存放的数组，n是数组元素的个数，in和out是输入和输出文件名
template<class T>
void replacementSelection(T *A, int n, const char *in, const char *out) {
    T mval;                                  // 存放最小堆的最小值
    T r;                                     // 存放从输入缓冲区中读入的元素
    FILE *inputFile;                         // 输入文件句柄
    FILE *outputFile;                        // 输出文件句柄
    Buffer<T> input;                         // 输入缓冲区
    Buffer<T> output;                        // 输出缓冲区
    initFiles(inputFile, outputFile, in, out);   // 初始化输入输出文件
```

9.3.1 置换选择排序

```
initMinHeapArry(inputFile, n, A);
// 从输入文件读入n个数据初始化堆数组A
MinHeap<T> H(A, n);                          // 建立最小堆

initInputBuffer(input, inputFile);
// 初始化输入缓冲区，读入一部分数据
for (int last = (n-1); last >= 0;) {                    // 堆不为空
    mval = H.heapArray[0];          // 获得堆的最小值
    sendToOutputBuffer(input, output, inputFile,
outputFile, mval);   // 把mval送到输出缓冲区
    input.read(r);                          // 从输入缓冲区读入一个记录r
    if (!less(r, mval)) {
// 如果r值大于等于输出值，r放到堆的根结点
        H.heapArray[0] = r;
    }
```

9.3.1 置换选择排序

```
            else {                                  // r不能入堆
                    H.heapArray[0] = H.heapArray[last];
            // 用堆中last位置的记录代替根结点
                    H.heapArray[last] = r;// 把r放到last位置
                    H.setSize(last);                // 堆规模减小1

last--;                                             // 设置
    LAST = LAST - 1
            }
            if (last != 0) {                        // 重新排列堆
                    H.SiftDown(0);    // 从根结点开始向下筛选
            }
    }
    endUp(output, inputFile, outputFile);           // 处理
    输出缓冲区，关闭输入和输出文件
}
```

9.3.1 置换选择排序

置换选择排序算法得到的顺串长度并不相等。平均情况下，置换选择排序算法可以形成长度为2M的顺串。

9.3.2 二路外排序

处理方法：即首先把数据文件划分成若干段，用有效的内排序方法对文件的各段进行初始排序以形成顺串；然后把这些顺串逐趟合并，直至变为一个顺串为止。

9.3.2 二路外排序

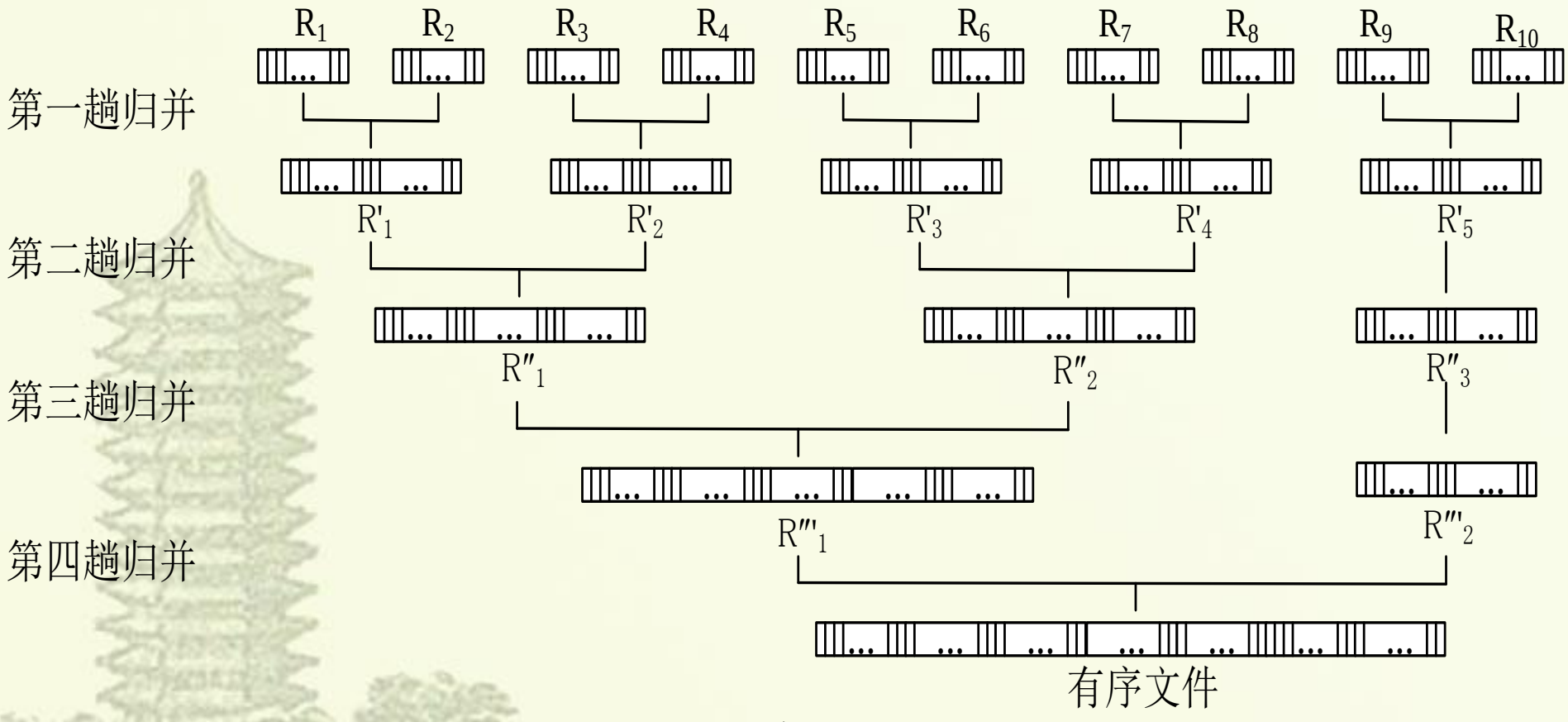


图 9.3　二路归并过程

9.3.2 二路外排序

- 为一个待排文件创建尽可能大的初始顺串，可以大大减少扫描遍数和外存读写次数。
- 归并顺序的安排也能影响读写次数，把初始顺串长度作为权，其实质就是Huffman树最优化问题。

9.3.3 多路归并——选择树

- k路归并是每次将k个顺串合并成一个排好序的顺串。一般情况下，对m个初始顺串进行k路归并时归并趟数为$\log_k m$。增加每次归并的顺串数量k可以减少归并趟数。
- 选择树是完全二叉树，有两种类型：赢者树和败者树。

赢者树

- 假设用完全二叉树的公式化描述方法来定义赢者树，采用数组作为存储结构。
  - 选手或叶结点用数组L[1...n]表示，内部结点用数组B[1...n-1]表示
  - 数组B中实际存放的是数组L的索引
- 赢者树的一个优点是，如果一个选手L[i]的分数值改变了，可以很容易地修改这棵赢者树。只需要沿着从L[i]到根结点的路径修改二叉树，而不必改变其它比赛的结果。

赢者树

【代码9.2】 赢者树的类定义

```
template<class T>
class WinnerTree  {
private:
    int MaxSize;            // 最大选手数
    int n;                  // 当前选手数
    int LowExt;   // 最底层外部结点数
    int offset;
                  // 最底层外部结点之上的结点总数
    int *B;
                  // 赢者树数组，存储数组L的索引
    T *L;
                        // 叶结点数组
```

赢者树

```
    // 在内部结点B[p]处开始从右分支向上比赛
    void Play(int p, int lc, int rc, int(*winner)(T A[], int b, int
    c));
public:
    WinnerTree(int Treesize = MAX);
                                    // 构造函数
    ~WinnerTree(){delete [] B;}
                                    // 析构函数
    void Initialize(T A[], int size,int (*winner)(T A[], int b, int
    c)); // 初始化赢者树
    int Winner();                                   // 返回最终胜者的
    索引
    void RePlay(int i, int(*winner)(T A[], int b, int c));
                        // 外部L[i]改变后重构赢者树
};
```

赢者树

数组B中存储的是L的下标

当前进行比较的8个关键码

等待进行比较的8个顺串

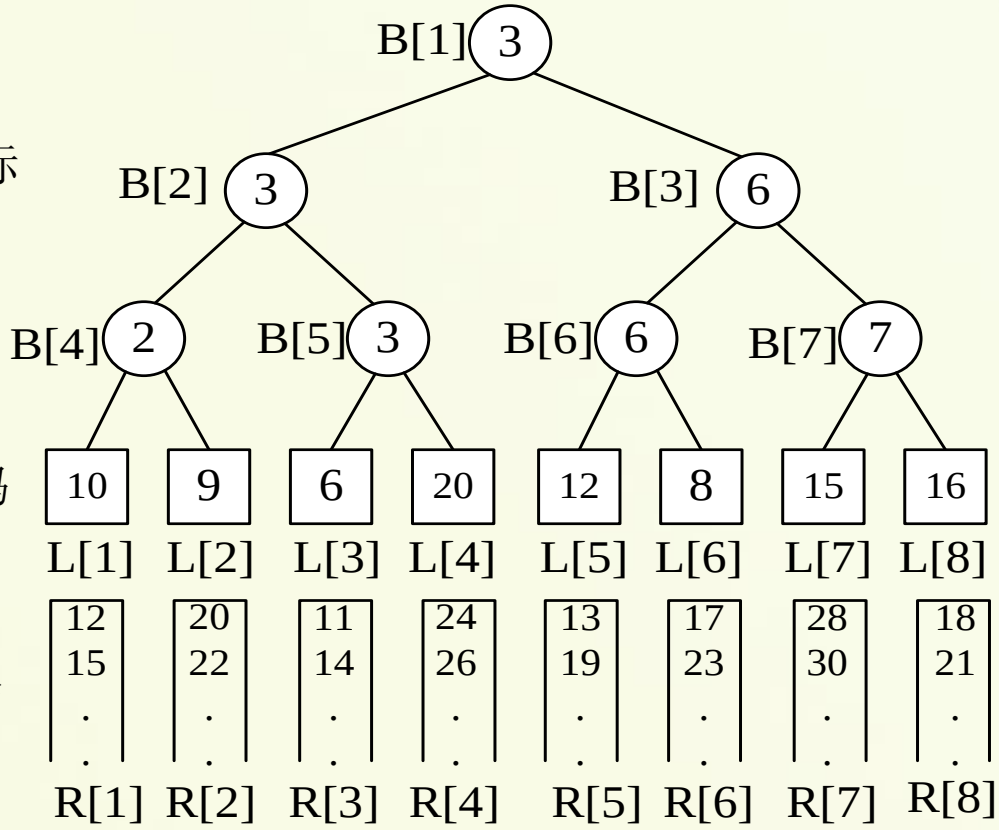


图9.5　8路合并的赢者树

赢者树

数组B中存储的是L的下标

当前进行比较的8个关键码

等待进行比较的8个顺串

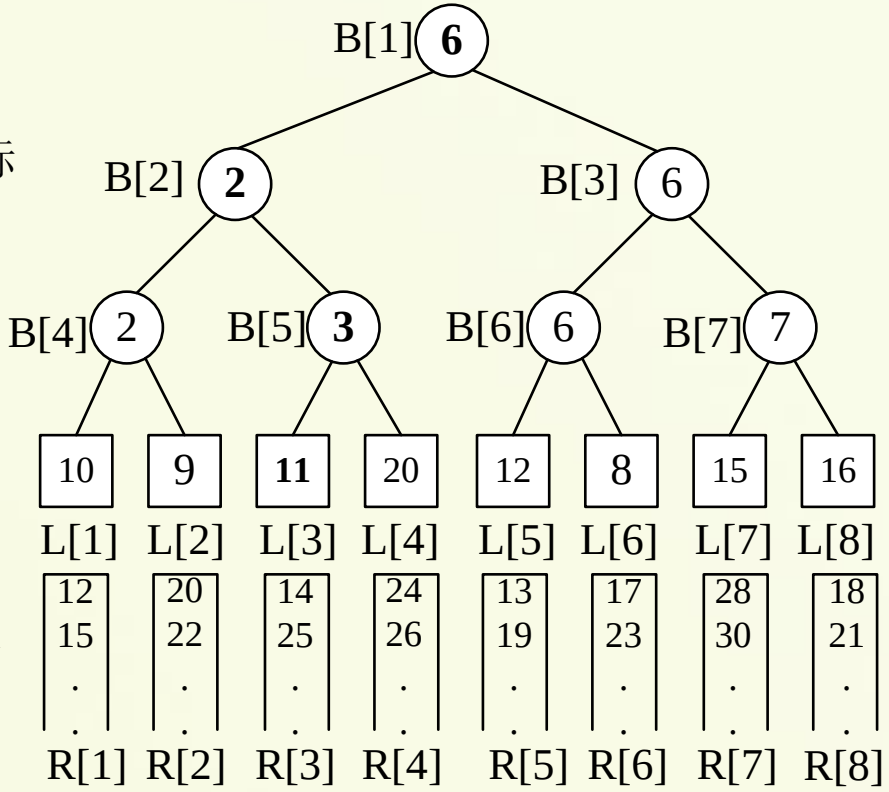


图9.6　重构后的赢者树

败者树

- 败者树是赢者树的一种变体。在败者树中，用父结点记录其左右子结点进行比赛的败者，而让获胜者去参加更高阶段的比赛。
- 另外，根结点处加入一个结点来记录整个比赛的胜者。
- 采用败者树是为了简化重构的过程。

败者树

重构过程如下：

- 将新进入选择树的结点与其父结点进行比赛
  - 把败者的下标存放在父结点中
  - 而胜者再与上一级的父结点比较
- 比赛沿着到根结点的路径不断进行，直到结点B[1]处
  - 把败者的索引放在结点B[1]
  - 把胜者的索引放到结点B[0]

败者树

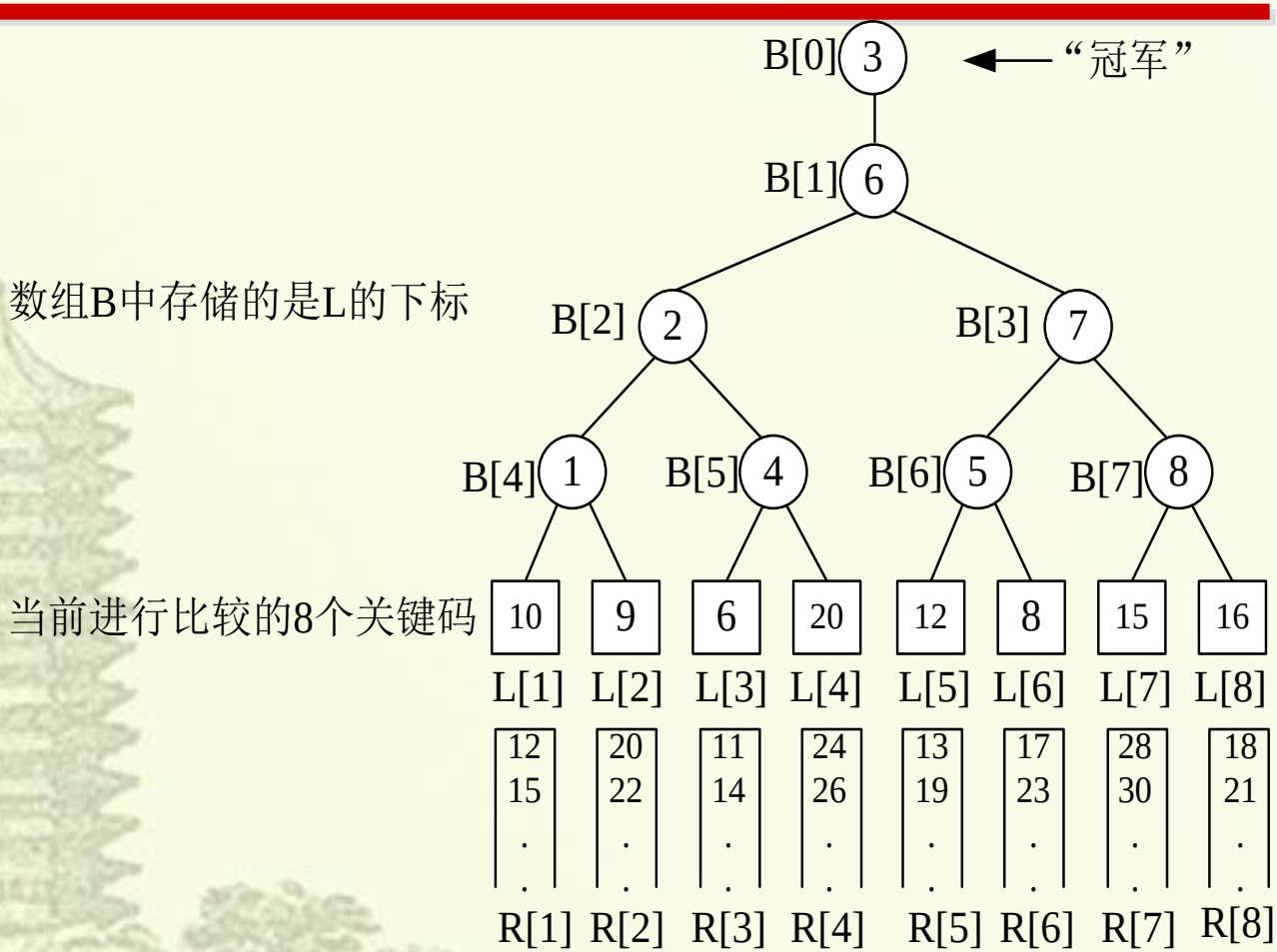


图9.7　8路合并的败者树示例

败者树

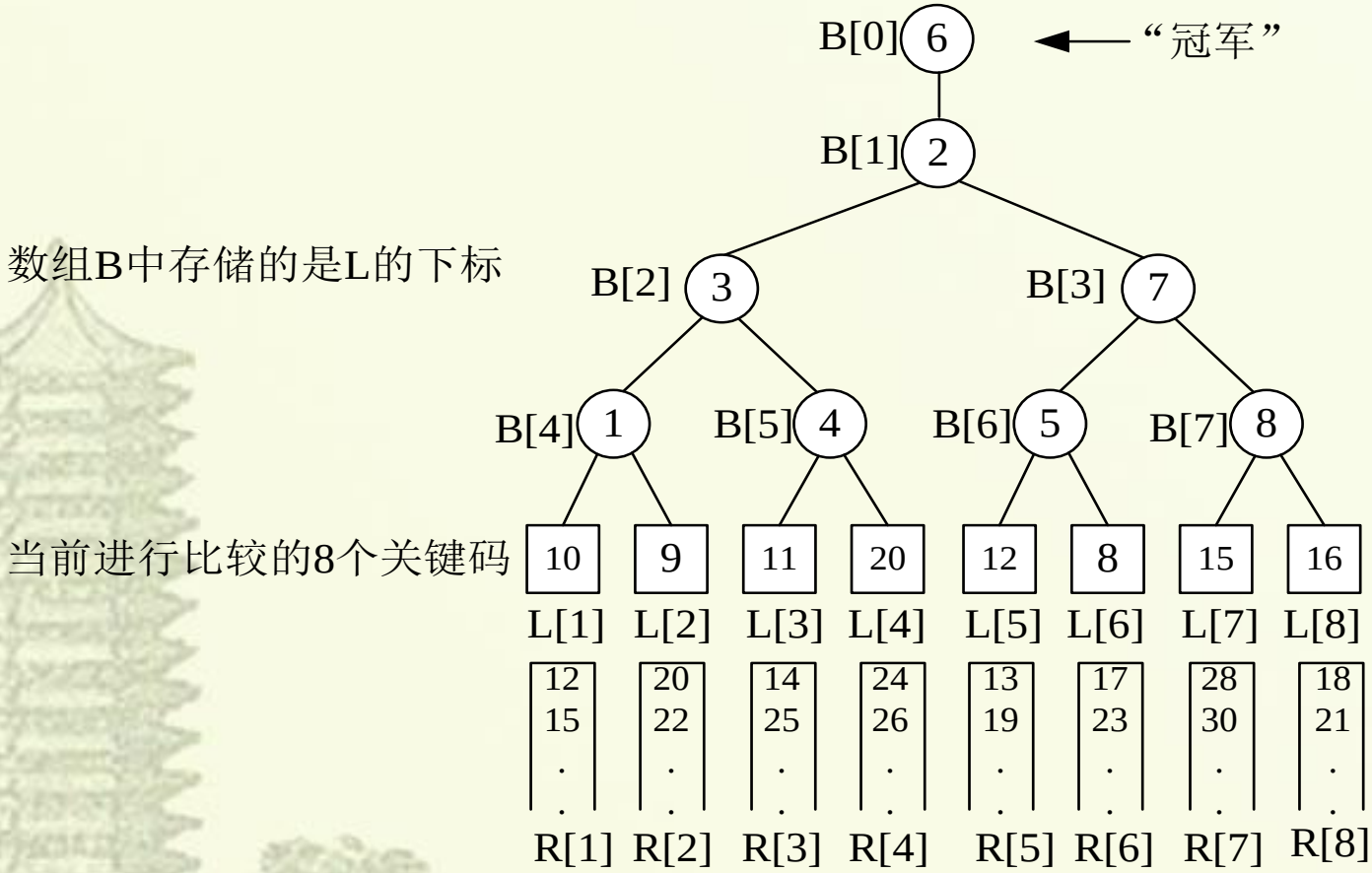


图9.8 重构后的8路合并败者树

败者树

- 利用败者树实现多路归并排序算法，首先初始化一棵败者树，取得最终胜者的索引。
  - 把胜者送入输出缓冲区中，如缓冲区已满，就把输出缓冲区的数据传输到磁盘文件中
  - 然后再从输入缓冲区中读入一个新的关键码，重新从叶结点到根结点调整败者树选择下一个胜者
  - 如果输入缓冲区为空，则从输入文件读取一批数据进输入缓冲区；如果对应输入文件已经没有数据了（本顺串数据处理完毕），就在新的竞赛者位置置入一个较大的数

多路归并的效率

假设对k个顺串进行归并：

- 原始方法：找到每一个最小值的时间是O(k)，产生一个大小为n的顺串的时间复杂度是O (k·n)
- 败者树方法：对k个顺串进行归并，初始化包含k个选手的败者树，需要时间O(k)，把最小值输出到缓冲区后，读入一个新值并重构败者树的时间为O(log k)，那么产生一个大小为n的顺串的总时间缩短为O (k+n·log k)，近似于O (n·log k)
- 显然败者树归并效率更高

9.4 文件管理和外排序知识点总结

- 主存储器和外存储器的特点
- 文件的组织形式
- 外排序思想
- 置换选择排序
- 二路外排序
- 选择树（赢者树和败者树）

The End