



数据结构与算法（九）

张铭 主讲

采用教材：张铭，王腾蛟，赵海燕 编写
高等教育出版社，2008.6（“十一五”国家级规划教材）

<http://www.jpku.pku.edu.cn/pkujpk/course/sjjg>



第9章 文件管理和外排序

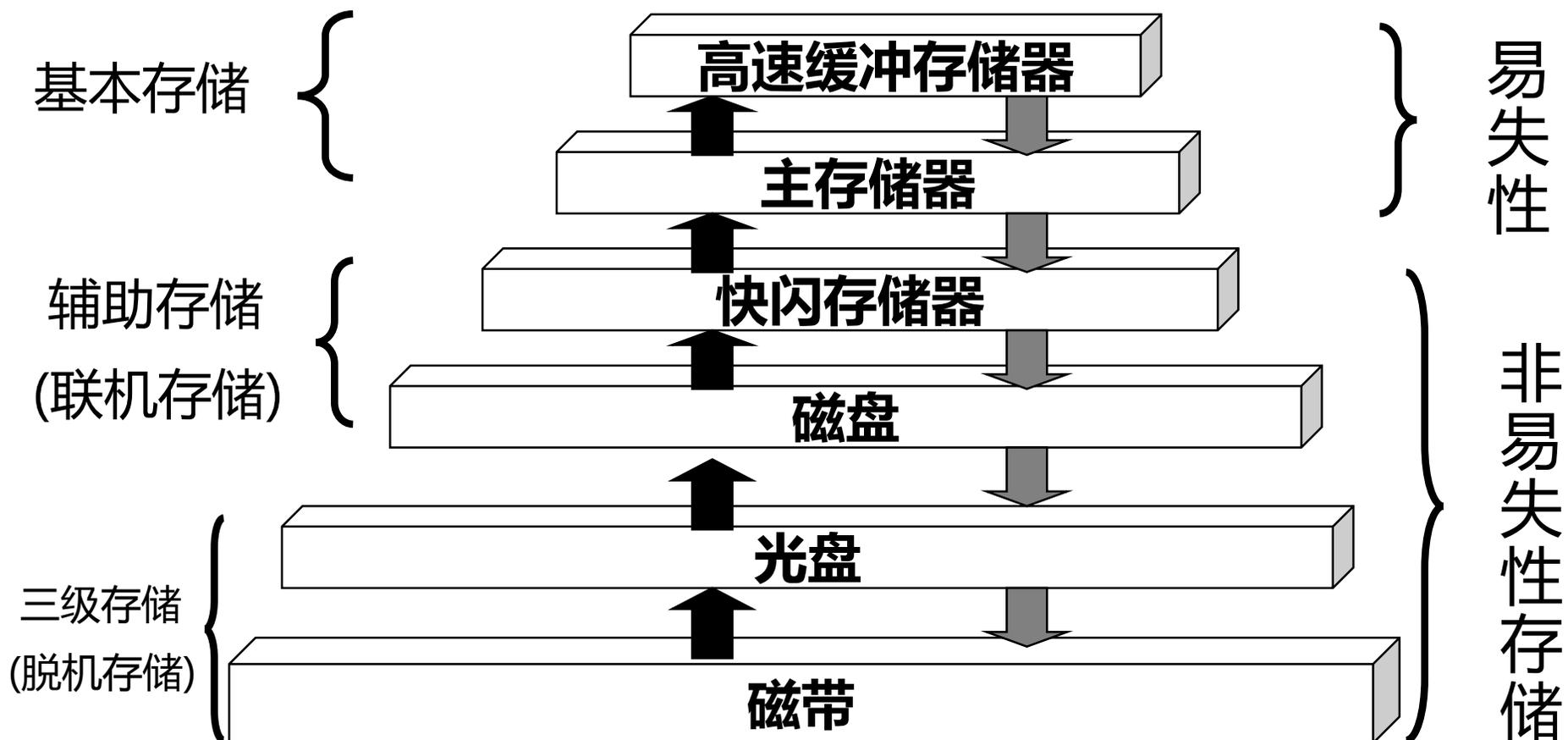
- 9.1 主存储器和外存储器
- 9.2 文件的组织和管理
 - 9.2.1 文件组织
 - 9.2.2 C++ 的流文件
- 9.3 外排序



主存储器和外存储器

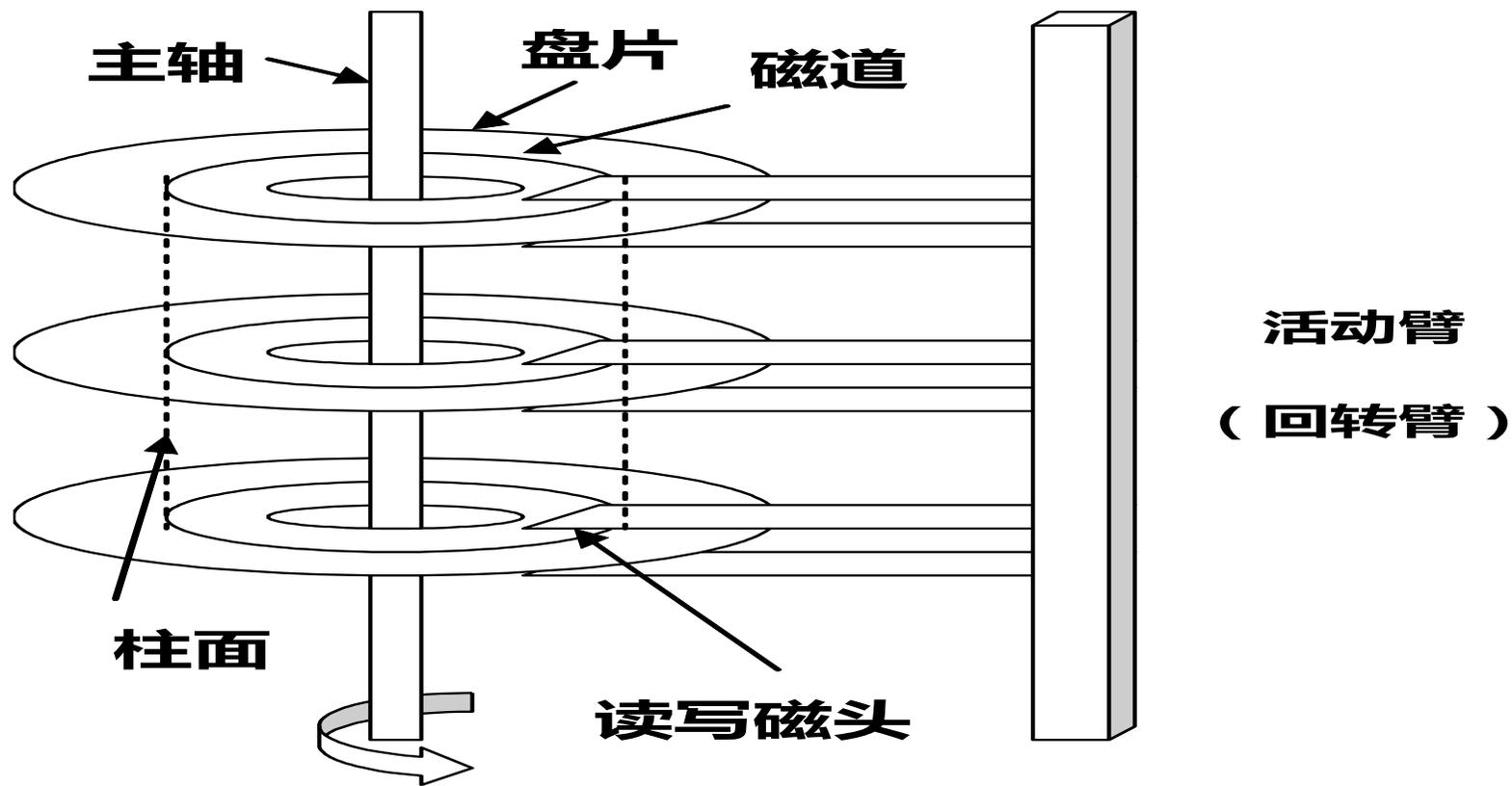
- 计算机存储器主要有两种：
 - 主存储器 (primary memory 或者 main memory , 简称 “内存” , 或者 “主存”)
 - 随机访问存储器 (Random Access Memory, 即 RAM)
 - 高速缓存 (cache)
 - 视频存储器 (video memory)
 - 外存储器 (peripheral storage 或者 secondary storage , 简称 “外存”)
 - 硬盘 (几百G - 几百T , 10^{12} B)
 - 磁带 (几个P , 10^{15} B)

物理存储介质概览

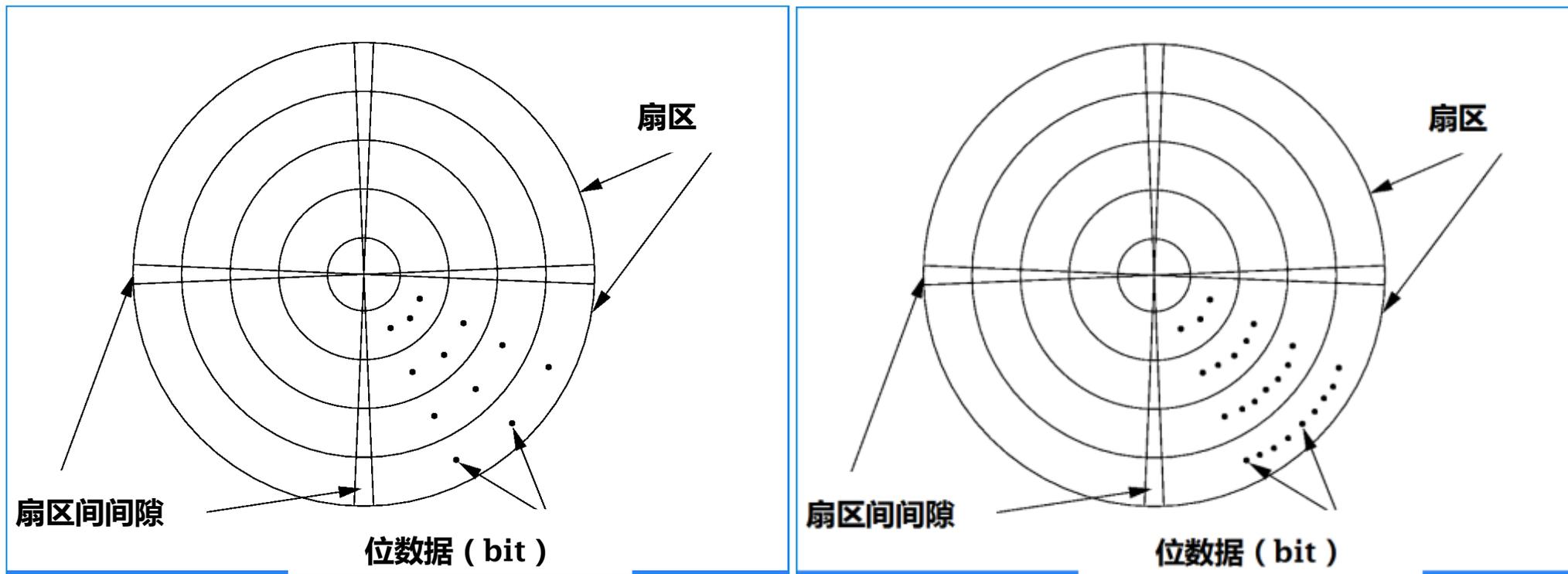


9.1 主存储器和外存储器

磁盘的物理结构



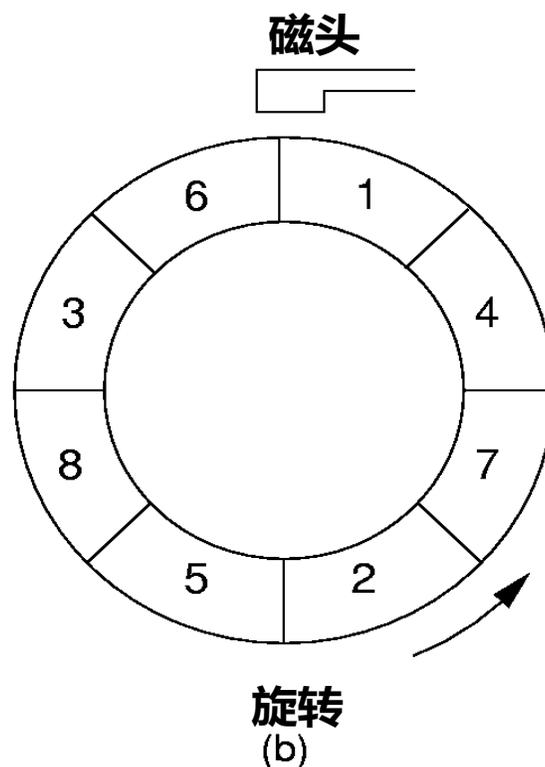
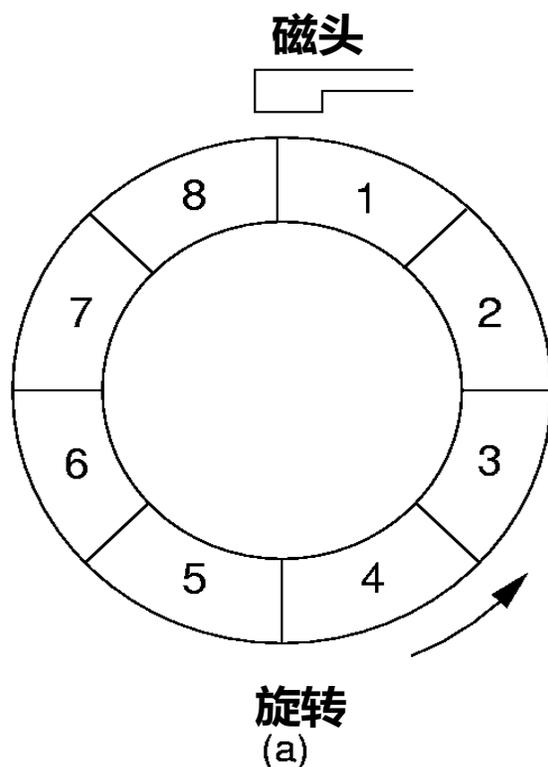
磁盘盘片的组织



9.1 主存储器和外存储器

磁盘磁道的组织（交错法）

• 每页 512 字节 或 1024 字节



(a) 没有扇区交错；

(b) 以 3 为交错因子



内存的优缺点

- 优点：访问速度快
- 缺点：造价高，存储容量小，断电丢数据
- 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)
 - 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 **减少外存的访问次数**，从而减少程序执行的时间

9.1 主存储器和外存储器

- KB (kilo byte) 10^3B (页块)
- MB (mega byte) 10^6B (高速缓存)
- GB (giga) 10^9B (内存、硬盘)
- TB (tera) 10^{12}B (磁盘阵列)
- PB (peta) 10^{15}B (磁带库)
- EB = 10^{18}B ; ZB = 10^{21}B ; YB = 10^{24}B
- Googol 是 10 的 100 次方



文件的逻辑结构

- 文件是记录的汇集
 - 一个文件的各个记录按照某种次序排列起来，各记录间就自然地形成了一种线性关系
- 因而，文件可看成是一种线性结构



文件的组织和管理

- 逻辑文件(logical file)
 - 对高级程序语言的编程人员而言
 - 连续的字节构成记录，记录构成逻辑文件
- 物理文件(physical file)
 - 成块地分布在整个磁盘中
- 文件管理器
 - 操作系统或数据库系统的一部分
 - 文件的记录无结构，数据库文件是结构型记录
 - 把逻辑位置映射为磁盘中具体的物理位置



文件组织

- 文件逻辑组织有三种形式：
 - 顺序结构的定长记录
 - 顺序结构的变长记录
 - 按关键码存取记录
- 常见的物理组织结构：
 - 顺序结构——顺序文件
 - 计算寻址结构——散列文件
 - 带索引的结构——带索引文件
 - 倒排是一种特殊的索引



9.2 文件的组织和管理

文件上的操作

- 检索：在文件中寻找满足一定条件的记录
- 修改：是指对记录中某些数据值进行修改。若对关键码值进行修改，这相当于删除加插入
- 插入：向文件中增加一个新记录
- 删除：从文件中删去一个记录
- 排序：对指定好的数据项，按其值的大小把文件中的记录排成序列，较常用的是按关键码值的排序



C++ 的标准输入输出流类

- 标准输入输出流类
 - 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，支持从磁盘文件的输入
 - ofstream 类，从 ostream 类派生，支持向磁盘文件的输出
 - fstream 类，从 iostream 类派生，支持对磁盘文件的输入和输出



fstream类的主要成员函数

文件指针 **定位** ; 在当前文件指针位置 **读取** ; 向当前文件指针位置 **写入**

```
#include <fstream.h> // fstream = ifstream + ofstream
void fstream::open(char*name, openmode mode);
// 打开文件
fstream::read(char*ptr, int numbytes); // 从文件当前位置读入字节
fstream::write(char*ptr, int numbytes); // 向文件当前位置写入字节
// 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
```



缓冲区和缓冲池

- 目的：减少磁盘访问次数的
- 方法：缓冲（buffering）或缓存（caching）
 - 在内存中保留尽可能多的块
 - 可以增加待访问的块已经在内存中的机会
- 存储在一个缓冲区中的信息经常称为一页（page），往往是一次 I/O 的量
- 缓冲区合起来称为缓冲池（buffer pool）



替换缓冲区块的策略

- 新的页块申请缓冲区时，把最近最不可能被再次引用的缓冲区释放来存放新页
 - “先进先出”（FIFO）
 - “最不频繁使用”（LFU）
 - “最近最少使用”（LRU）



思考

1. 查询内存、硬盘、磁带、高速缓存等设备每字节的价格
2. 查询当前主流硬盘的性能指标
 - 容量 (G)
 - 磁盘旋转速度 (rpm)
 - 交错因子
 - 寻道时间
 - 旋转延迟时间



第9章 文件管理和外排序

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



磁盘文件的排序

- 对外存设备上(文件)的排序技术
- 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 (run)
 - 处理顺串，最后形成对整个数据文件的排列文件

9.3 外排序

置换选择排序



9.3 外排序

置换选择示例

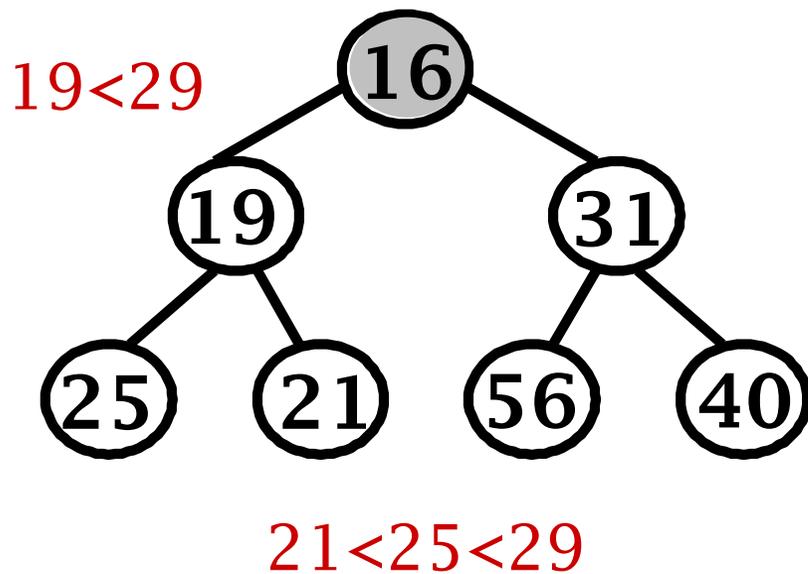
输出堆插入堆
~~输出堆插入堆~~
 > 16

输入

存储

输出

29
14
35
13



12

9.3 外排序

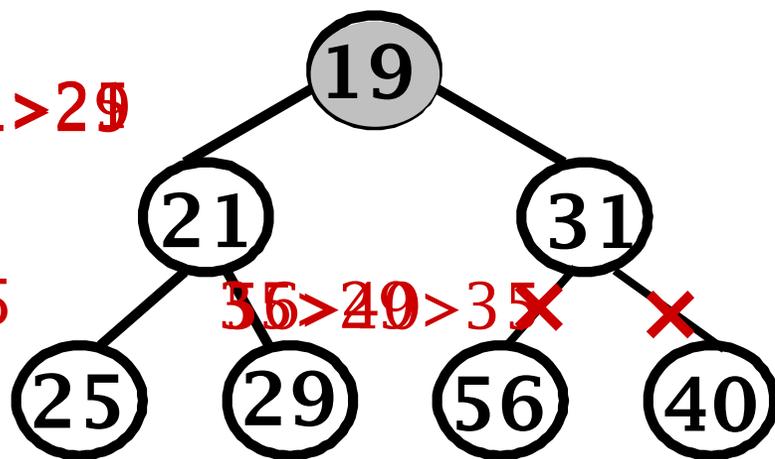
置换选择示例

输入新数据值

输入

14
35
13

存储



输出

16
12



置换选择算法的实现

```
// 模板参数 Elem 代表数组中每一个元素的类型
// A 是从外存读入 n 个元素后所存放的数组
// in 和 out 分别是输入和输出文件名
template <class Elem>
void replacementSelection(Elem * A, int n, const char * in, const char * out) {
    Elem mval; // 存放最小值堆的最小值
    Elem r; // 存放从输入缓冲区中读入的元素
    FILE * inputFile; // 输入、输出文件句柄
    FILE * outputFile;
    Buffer<Elem> input; // 输入、输出buffer
    Buffer<Elem> output; // 初始化输入输出文件
    initFiles(inputFile, outputFile, in, out);
    initMinHeapArray(inputFile, n, A); // 建堆
    MinHeap<Elem> H(A, n, n);
    initInputBuffer(input, inputFile);
```



```
for(int last = (n-1); last >= 0;){
    mval = H.heapArray[0];           // 最小值
    sendToOutputBuffer(input, output,
    inputFile, outputFile, mval);
    input.read(r);                   // 从输入缓冲区读入一个记录
    if (!less(r, mval))
        H.heapArray[0] = r;         // r放到根结点
    else {
        H.heapArray[0] = H.heapArray[last];
        H.heapArray[last] = r;
        H.setSize(last--);
    }
    H.SiftDown(0);                   // 调整根结点
}
endUp(output, inputFile, outputFile);
}
```

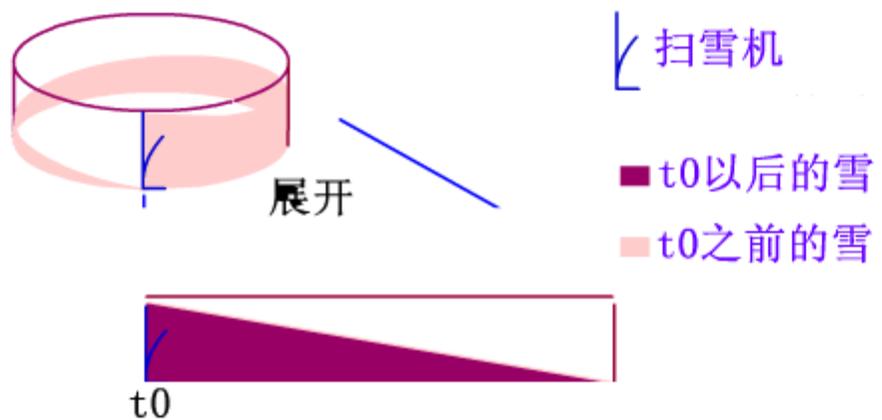


置换选择算法的效果

- 置换选择排序算法得到的顺串长度并不相等。
如果堆的大小是 M
- 一个顺串的最小长度就是 M 个记录
 - 至少原来在堆中的那些记录将成为顺串的一部分
- 最好的情况下，例如输入为正序，有可能一次就把整个文件生成为一个顺串
- 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 外排序

扫雪机模型

[click me](#)



9.3 外排序

二路外排序

- 归并原理：把第一阶段所生成的顺串加以合并(例如通过若干次二路合并)，直至变为一个顺串为止，即形成一个已排序的文件
- 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 外排序

二路归并外排序

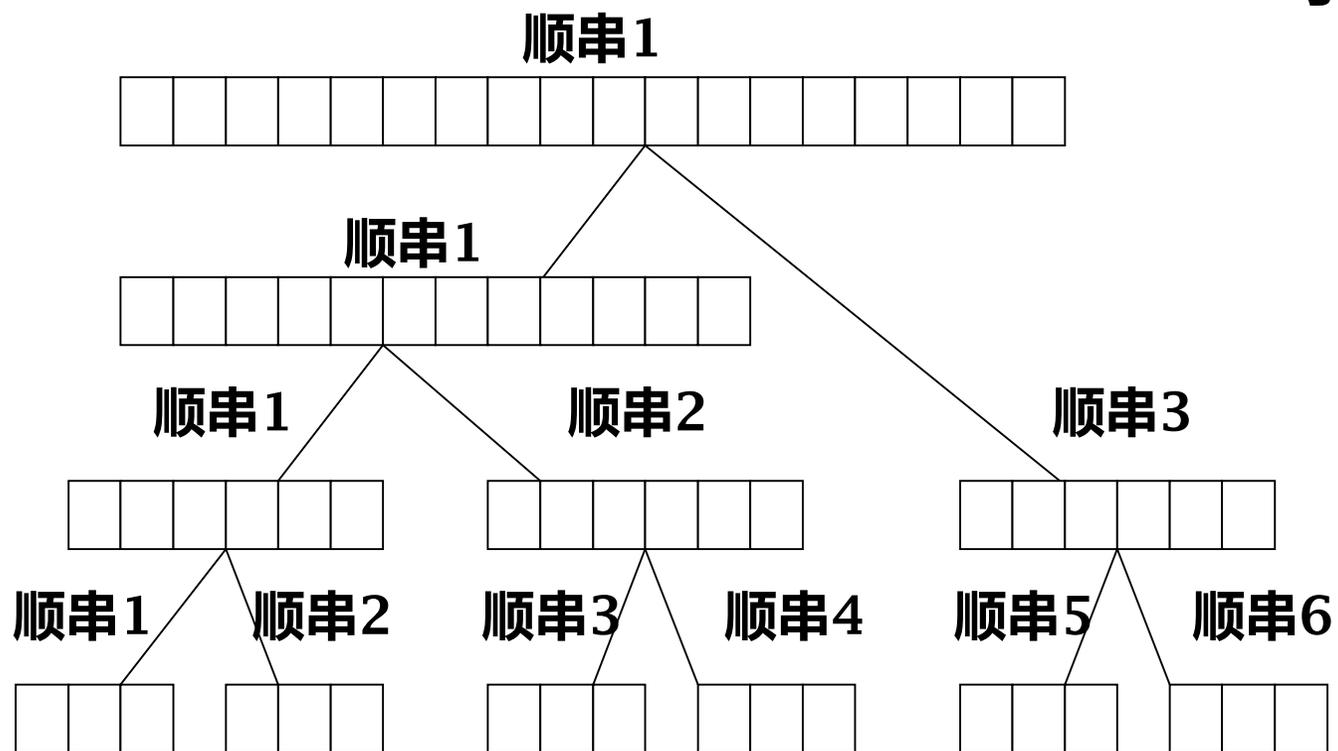
每个顺串中的块数

18

12

6

3



每个顺串中的记录数

4500

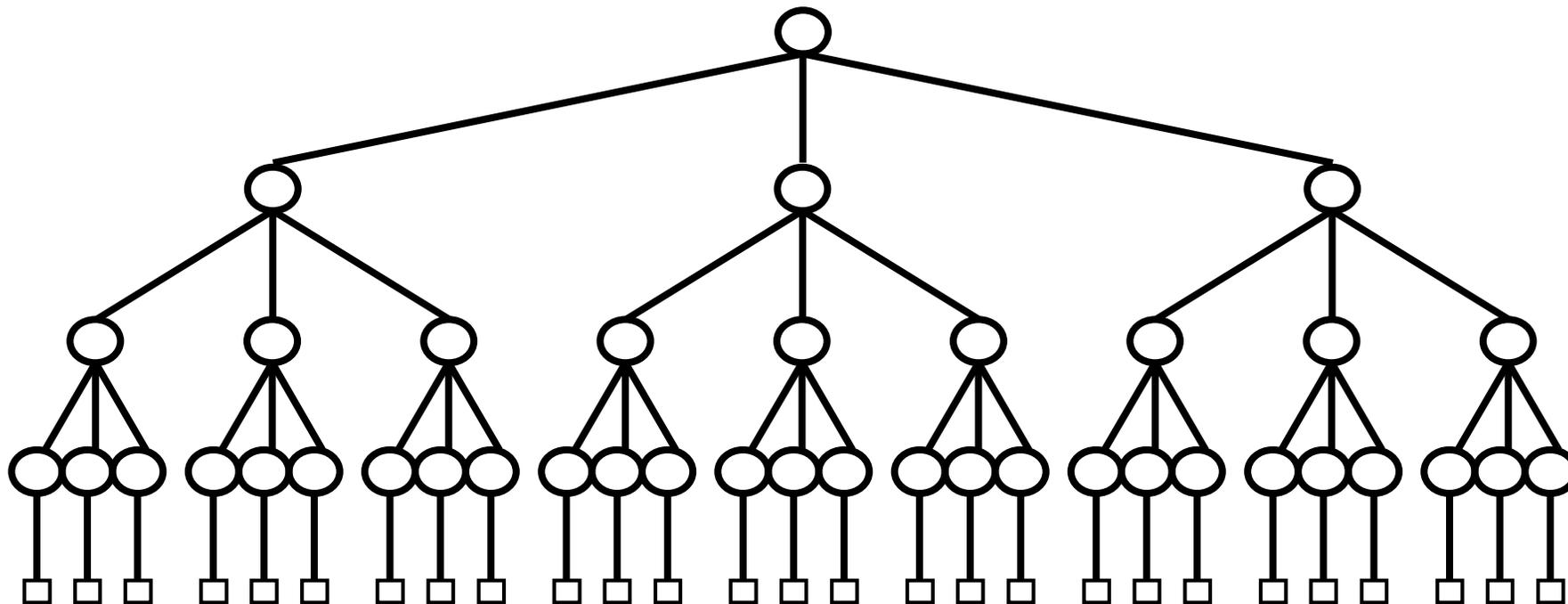
3000

1500

750

读写各： $3*6 + 6*2 + (12 + 6) = 48$ 次

多路归并

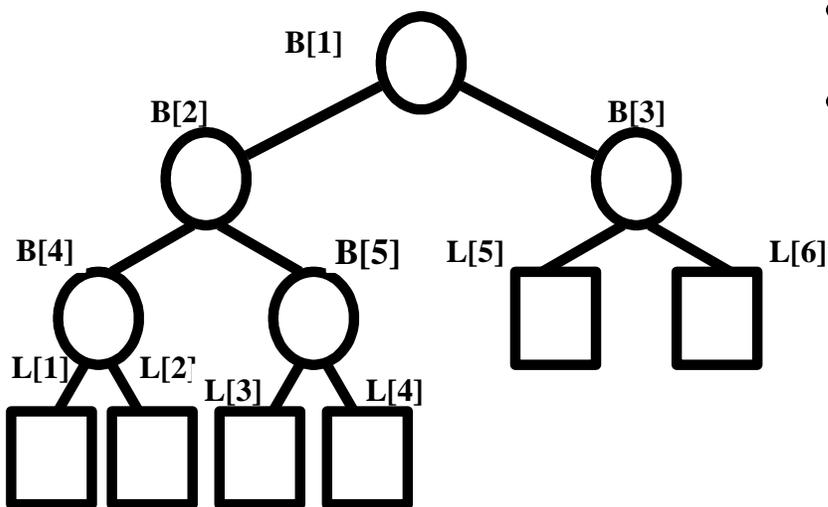




多路归并——选择树

- k 路归并是每次将 k 个顺串合并成一个排好序的顺串
- 在 k 路归并中，最直接的方法就是作 k-1 次比较来找出所要的记录，但这样做花的代价较大
- 我们采用选择树的方法来实现 k 路归并
 - 选择树是完全二叉树，有两种类型：赢者树和败方树
- 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

赢者树与数组的对应关系

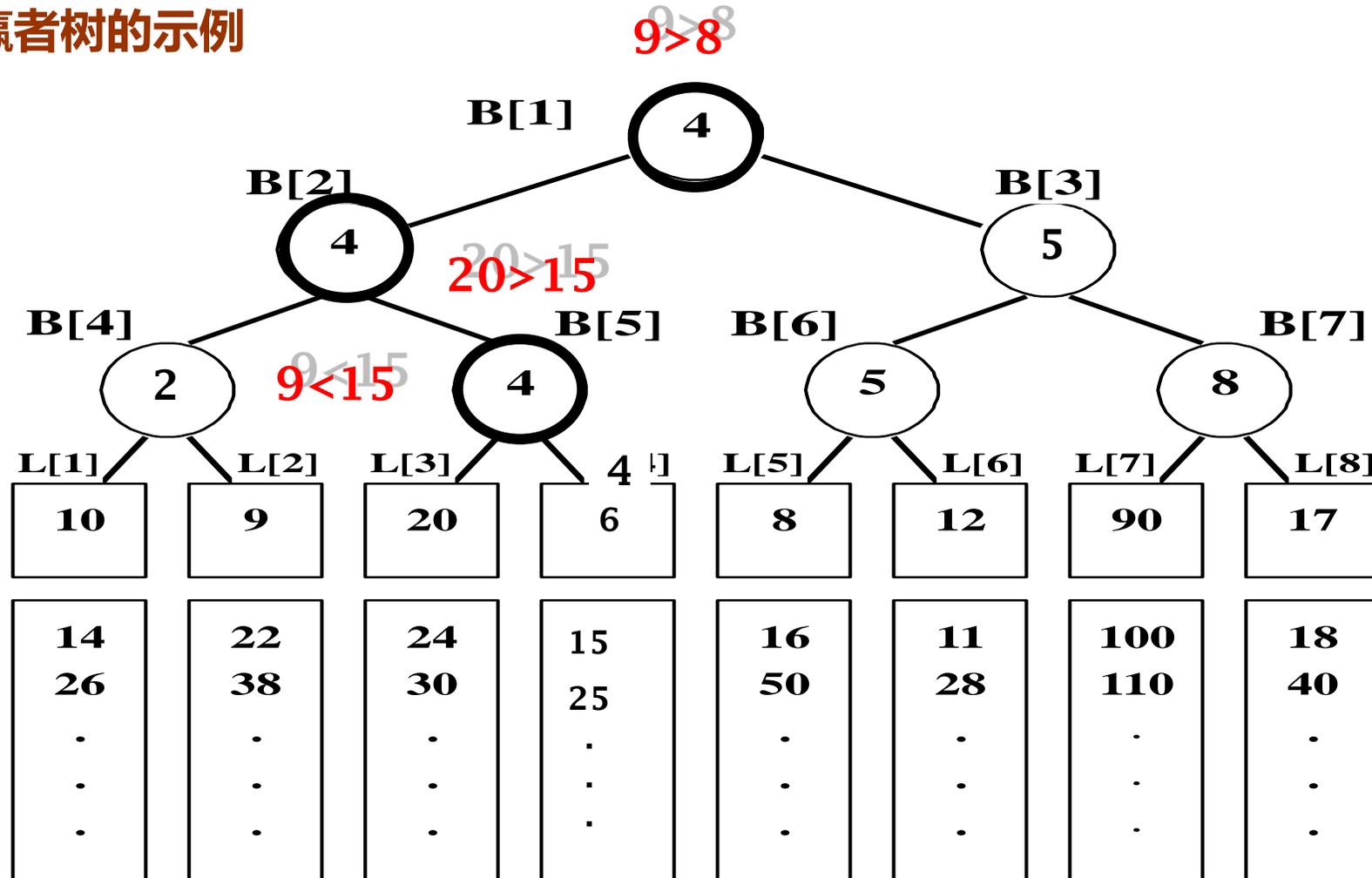


- $n=6, LowExt=4, Offset=7$
- $LowExt + Offset = 2n-1$

外部结点的数目为 n ， $LowExt$ 代表最底层的外部结点数目； $offset$ 代表最底层外部结点之上(内部+ $LowExt$ 之外的外部)所有结点数目。每一个外部结点 $L[i]$ 所对应的内部结点 $B[p]$ ， i 和 p 之间存在如下的关系：

$$\begin{cases} (i + offset) / 2 & i \leq LowExt \\ (i - LowExt + n - 1) / 2 & i > LowExt \end{cases}$$

赢者树的示例



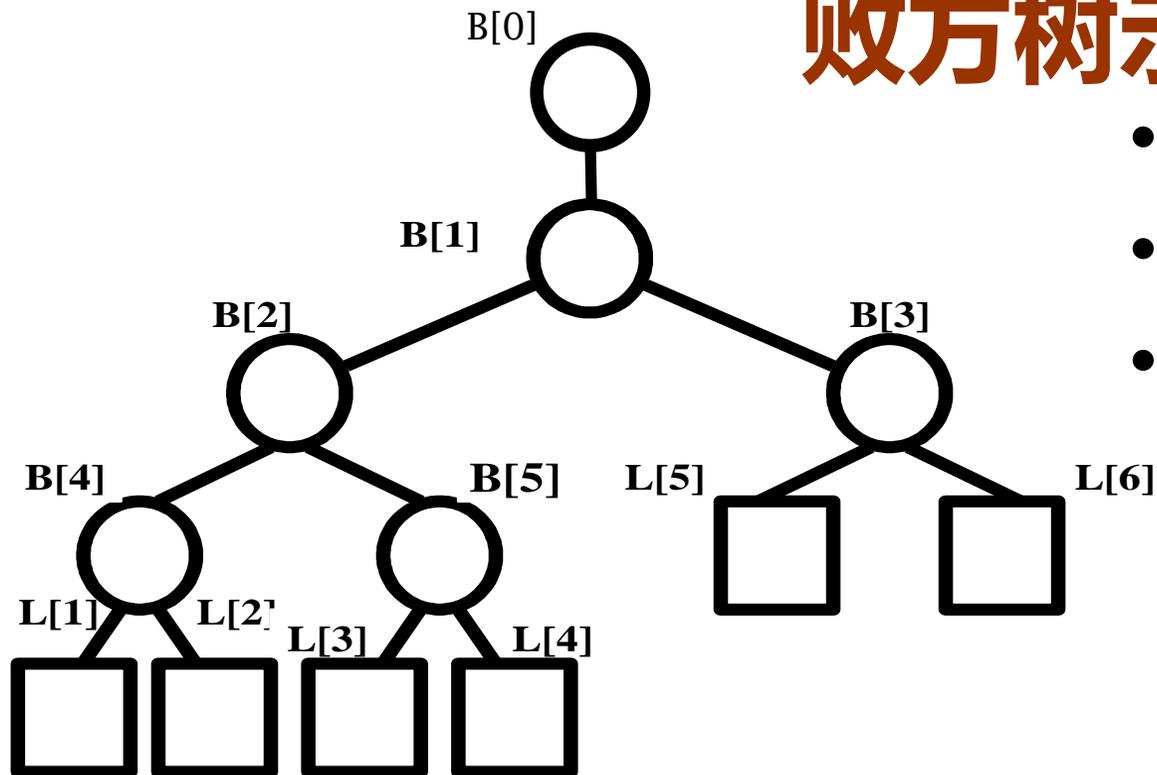
顺串 1 顺串 2 顺串 3 顺串 4 顺串 5 顺串 6 顺串 7 顺串 8

重构后的赢者树，改动的结点用较粗的框显示出来。为了重构这棵树，只须沿着从结点 L[4] 到根结点的路径重新进行比赛。

9.3 外排序

败方树示例

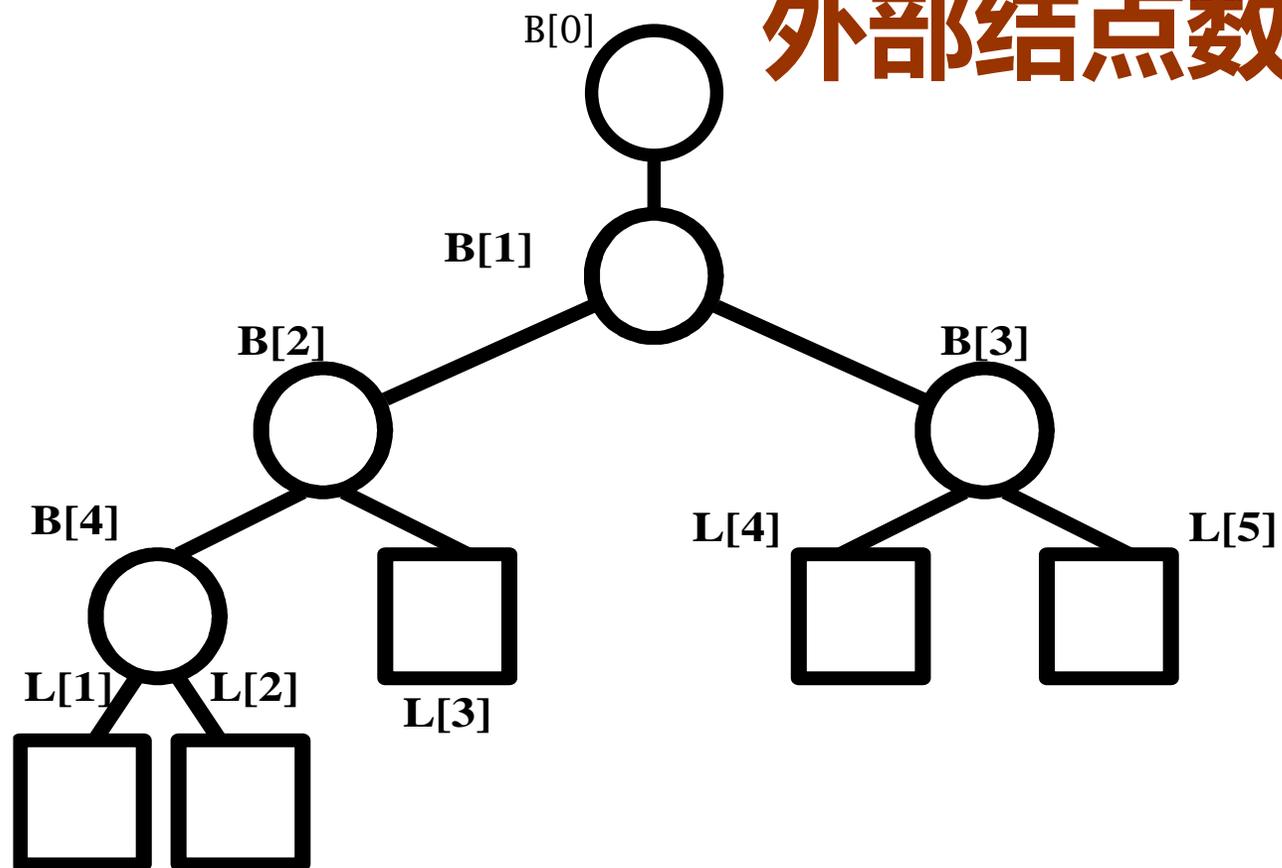
- $n=6$
- $LowExt=4$
- $Offset=7$

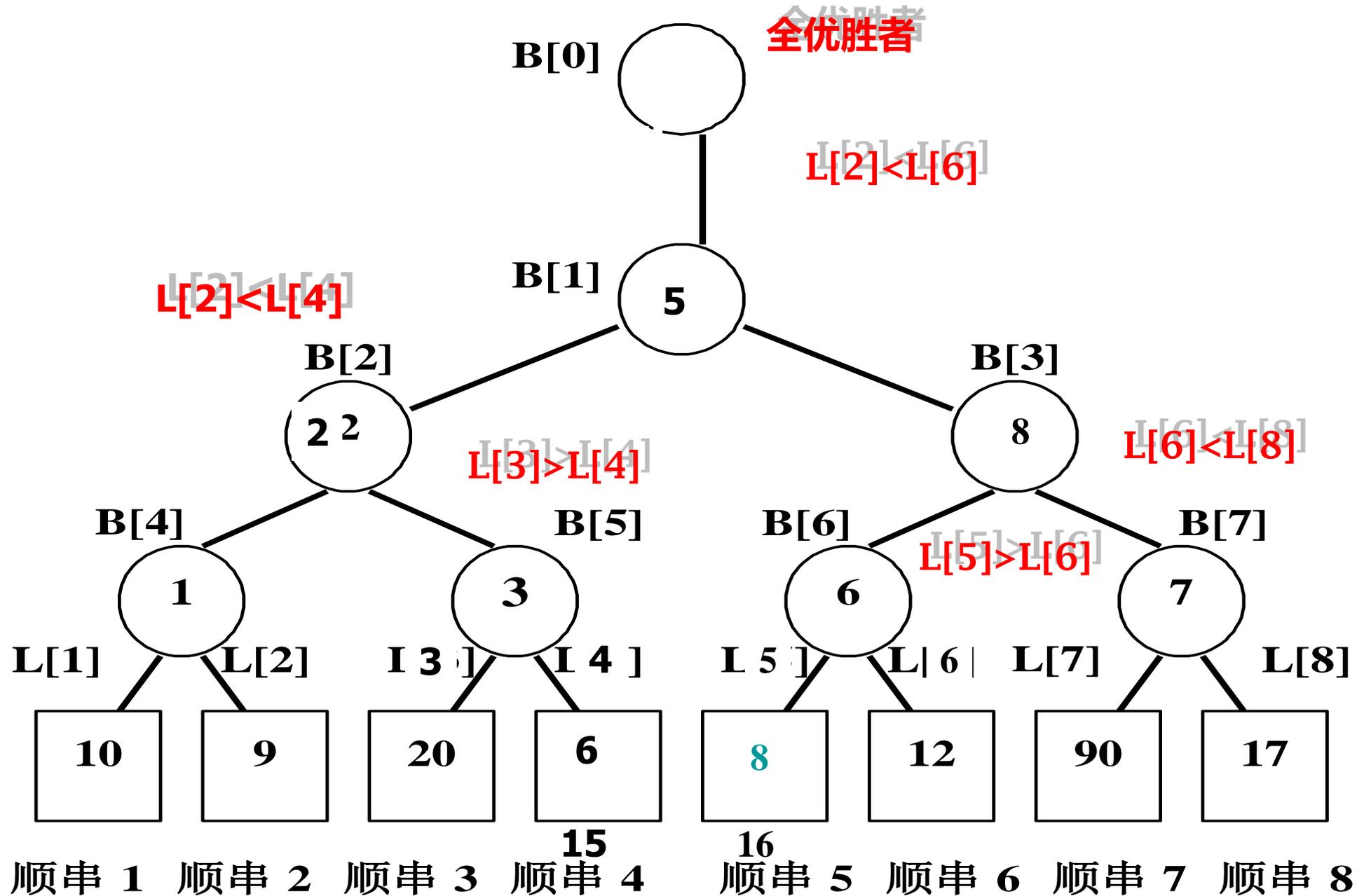


$$(i + offset) / 2 \quad i \leq LowExt$$

$$\left\{ \begin{array}{ll} (i + offset) / 2 & i \leq LowExt \\ (i - LowExt + n - 1) / 2 & i > LowExt \end{array} \right.$$

9.3 外排序

外部结点数 n 为奇数





9.3 外排序

败方树 ADT

```
template<class T>
class LoserTree{
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;              // 败方树数组，实际存放的是下标
    T * L;                // 元素数组
    void Play(int p,int lc,int rc,int(*winner)(T A[],int b,int c));
```



败方树ADT (续)

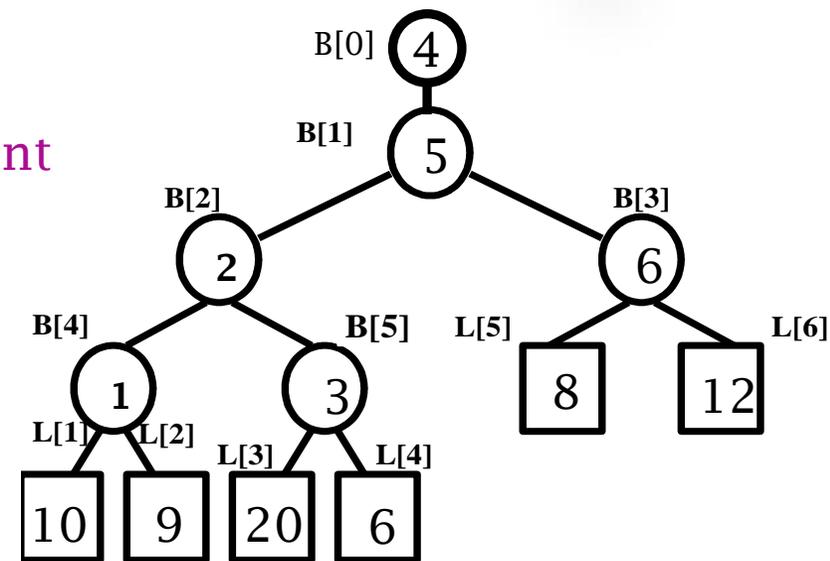
```
public:
LoserTree(int Treesize = MAX);
~LoserTree(){delete [] B;}
void Initialize(T A[], int size, int (*winner)(T A[], int b, int c),
int(*loser)(T A[], int b, int c));           // 初始化败方树
int Winner();                               // 返回最终胜者索引
void RePlay(int i, int(*winner)(T A[], int b, int c), int (*loser)(T A[], int
b, int c));                                 // 位置 i 的选手改变后重构败方树
};
// 成员函数Winner , 返回最终胜者 B[0] 的索引
template<class T>
int LoserTree<T>::Winner(){
    return (n)?B[0]:0;
}
```

9.3 外排序

```

template<class T>                                     // 初始化败方树
void LoserTree<T>::Initialize(T A[], int size, int(*winner)(T A[], int
b, int c), int(*loser)(T A[], int b, int c)) {
    if (size > MaxSize || size < 2) {
        cout<<"Bad Input!"<<endl<<endl; return; }
    n = size; L = A;                                 // 初始化成员变量
    int i,s;                                         // 计算s=2^log(n-1)
    for (s = 1; 2*s <= n-1; s+=s);
    LowExt = 2*(n-s); offset = 2*s-1;
    for (i = 2; i <= LowExt; i+=2)                 // 底层外部
        Play((offset+i)/2, i-1, i, winner, loser);
    if (n%2) {                                       // n奇数, 内部和外部比赛
        Play(n/2,B[(n-1)/2],LowExt+1,winner,loser); i = LowExt+3;
    } else i = LowExt+2;
    for (; i<=n; i+=2)                              // 剩余外部结点的比赛
        Play((i-LowExt+n-1)/2, i-1, i, winner, loser);
}

```



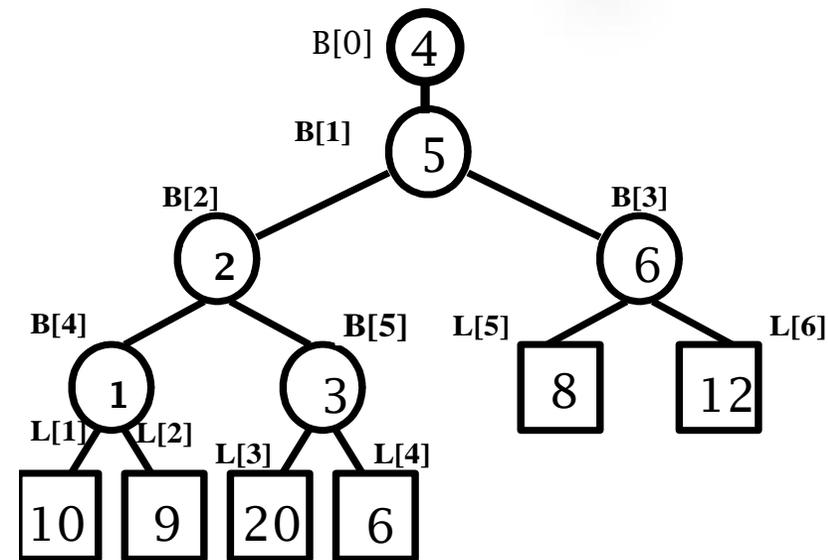


Play 比赛

```

template<class T>
void LoserTree<T>::Play(int p, int lc, int rc, int(* winner)(T
A[], int b, int c), int(* loser)(T A[], int b, int c)){
    B[p] = loser(L, lc, rc);    // 败者索引放在B[p]
    int temp1, temp2;
    temp1 = winner(L, lc, rc); // p处的胜者索引
    while(p>1 && p%2) {    // 内部右, 要沿路向上比赛
        temp2 = winner(L, temp1, B[p/2]);
        B[p/2] = loser(L, temp1, B[p/2]);
        temp1 = temp2;
        p/=2;
    } // B[p]是左孩子, 或者B[1]
    B[p/2] = temp1;
}

```



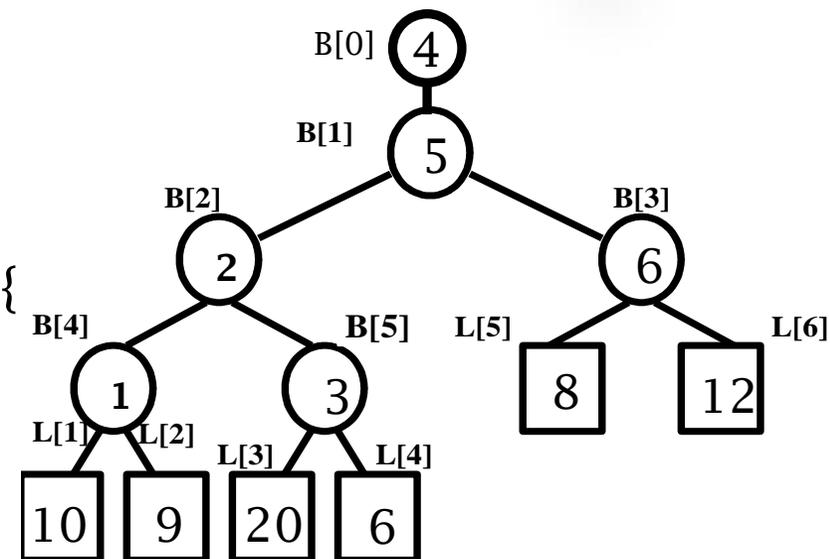
9.3 外排序

RePlay重构

```

template<class T> void LoserTree<T>::RePlay(int i, int
(*winner)(T A[], int b, int c), int (*loser)(T A[], int b, int c)){
    if (i <= 0 || i > n) {
        cout<<"Out of Bounds!"<<endl<<endl; return; }
    if (i <= LowExt) // 确定父结点的位置
        int p = (i+offset)/2;
    else p = (i-LowExt+n-1)/2;
    B[0] = winner(L, i, B[p]); B[p] = loser(L, i, B[p]);
    for(; (p/2)>=1; p/=2) { // 沿路径向上比赛
        int temp = winner(L,B[p/2], B[0]);
        B[p/2] = loser(L,B[p/2], B[0]);
        B[0] = temp;
    }
}

```





外排序效率考虑

- 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 $\lceil \log_k m \rceil$
- 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

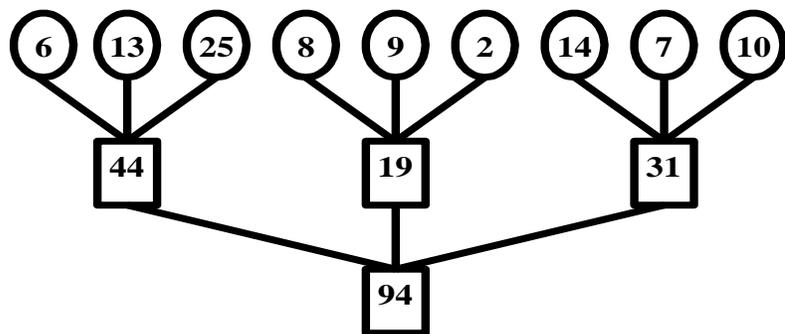


假设对 k 个顺串进行归并，归并后长 n

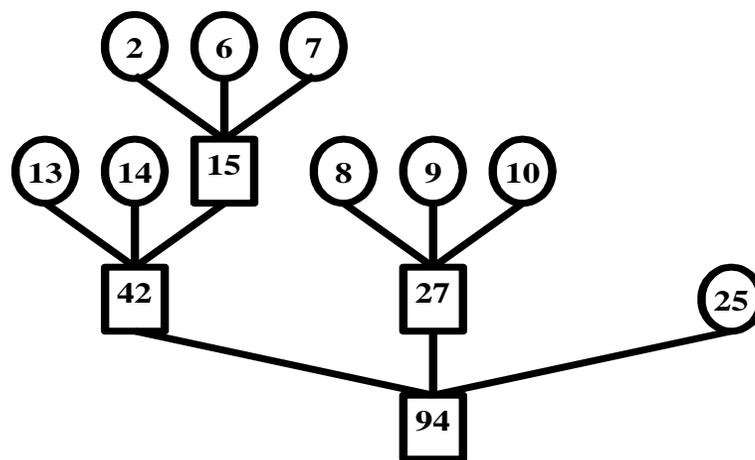
- 原始方法 $\Theta(k \cdot n)$ ，找到每一个最小值的时间是 $\Theta(k)$
- 败方树方法总时间为 $\Theta(k + n \cdot \log k)$
 - 初始化包含 k 个选手的败方树需要 $\Theta(k)$ 的时间
 - 读入一个新值并重构败方树的时间为
 - $\Theta(\log k)$

9.3 外排序

最佳归并树



(a) 一棵普通的归并树



(b) 最佳归并树

读写总次数 376

读写总次数 356



思考

- 是否可以用赢者树或败方树形成初始顺串？
- 是否可以用堆进行多路归并？



数据结构与算法

谢谢聆听

国家精品课“数据结构与算法”

<http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/>

张铭, 王腾蛟, 赵海燕

高等教育出版社, 2008. 6. “十一五”国家级规划教材