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.jpk.pku.edu.cn/pkujpk/course/sjjg

第9章 文件管理和外排序

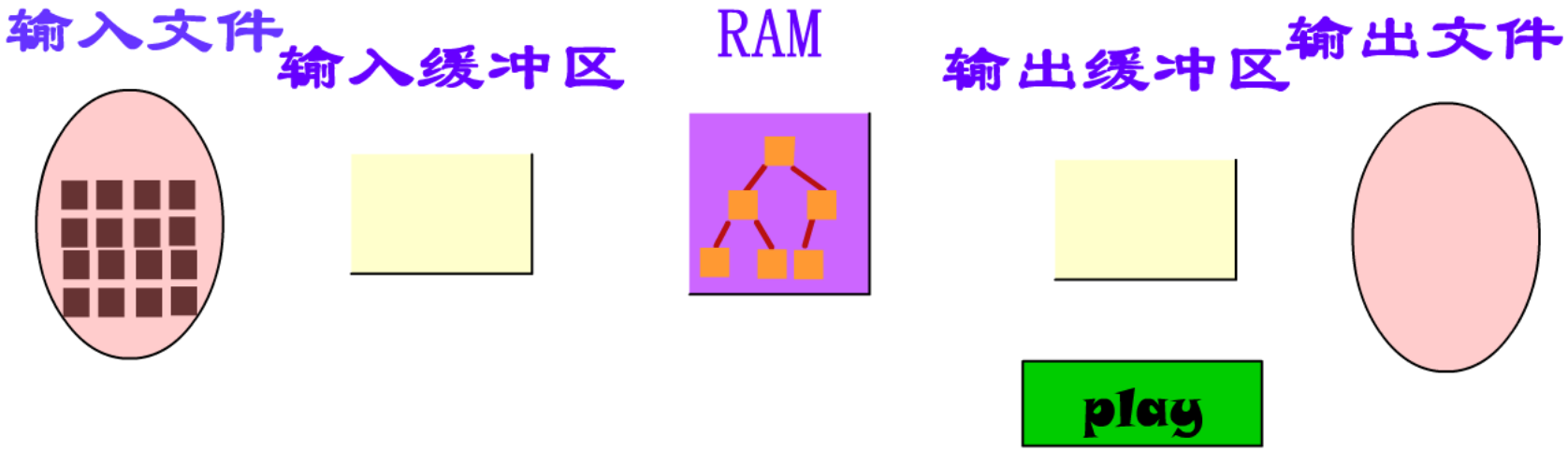
- 9.1 主存储器和外存储器
- 9.2 文件的组织和管理
- 9.3 外排序
  - 9.3.1 置换选择排序
  - 9.3.2 二路外排序
  - 9.3.3 多路归并——选择树

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16插入29

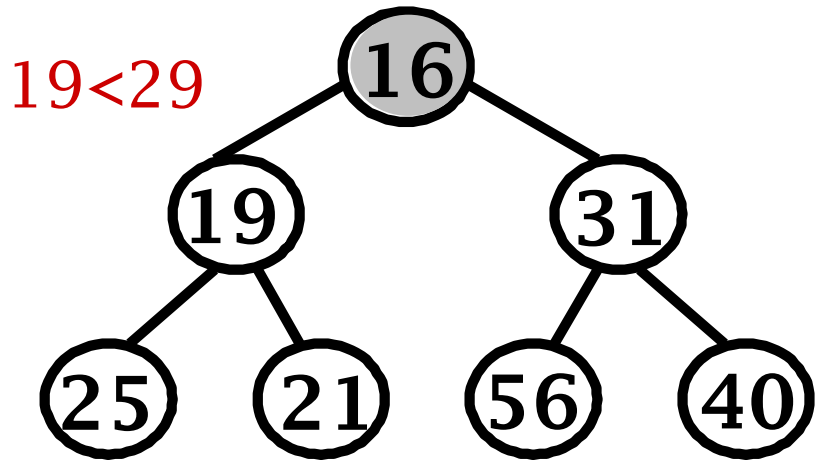
重新排列堆

>16

输入

29
14
35
13

存储



19<29

21<25<29

输出

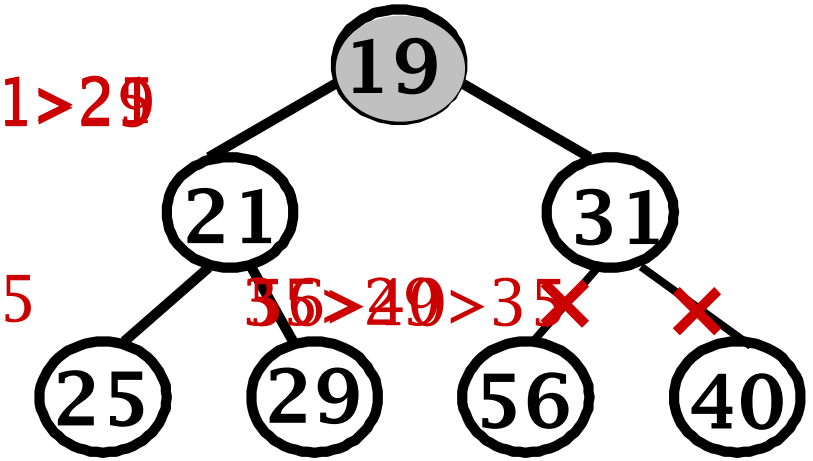
12

置换选择示例

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);
initMinHeapArry(inputFile, n, A);     // 建堆
MinHeap<Elem> H(A, n, n);
initInputBuffer(input, inputFile);
```

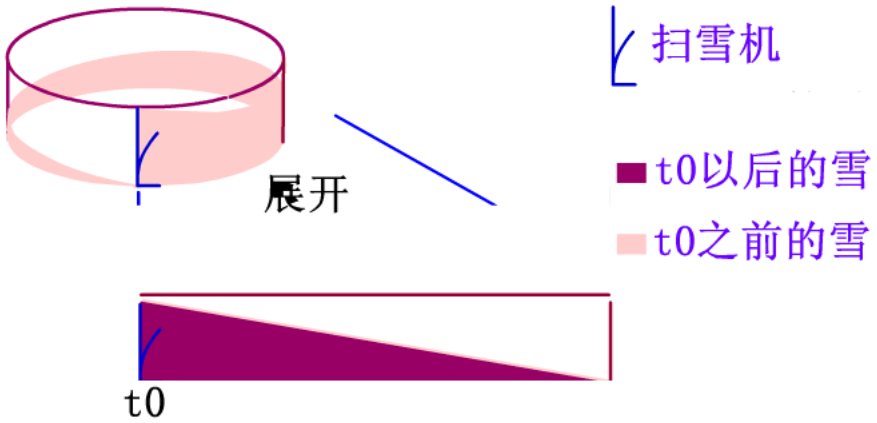
9.3 外排序

```
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 {                                    // last记录代替根结点，r放到last位置
        H.heapArray[0] = H.heapArray[last];
        H.heapArray[last] = r;
        H.setSize(last--);
    }
    H.SiftDown(0);                            // 调整根结点
}                                             // for
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

扫雪机模型

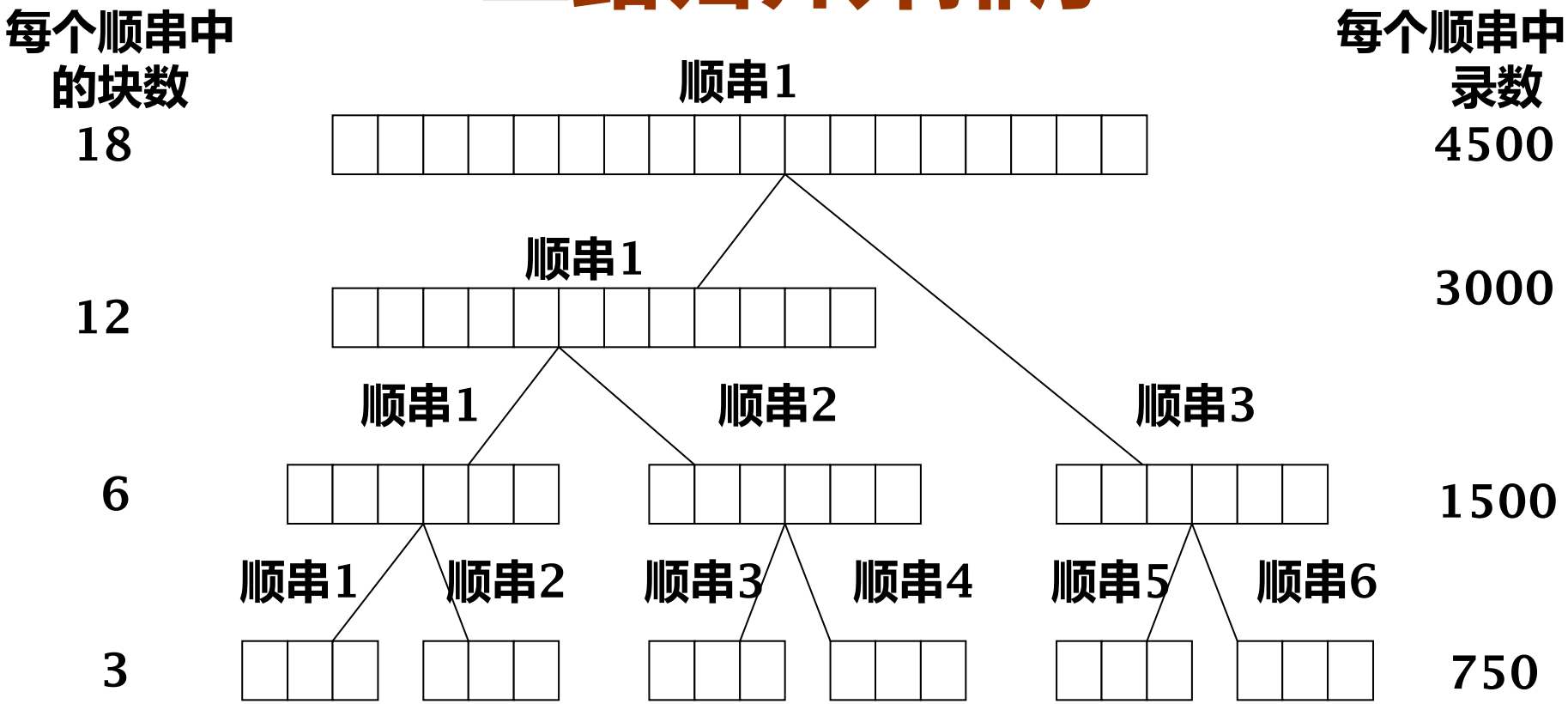


click me

二路外排序

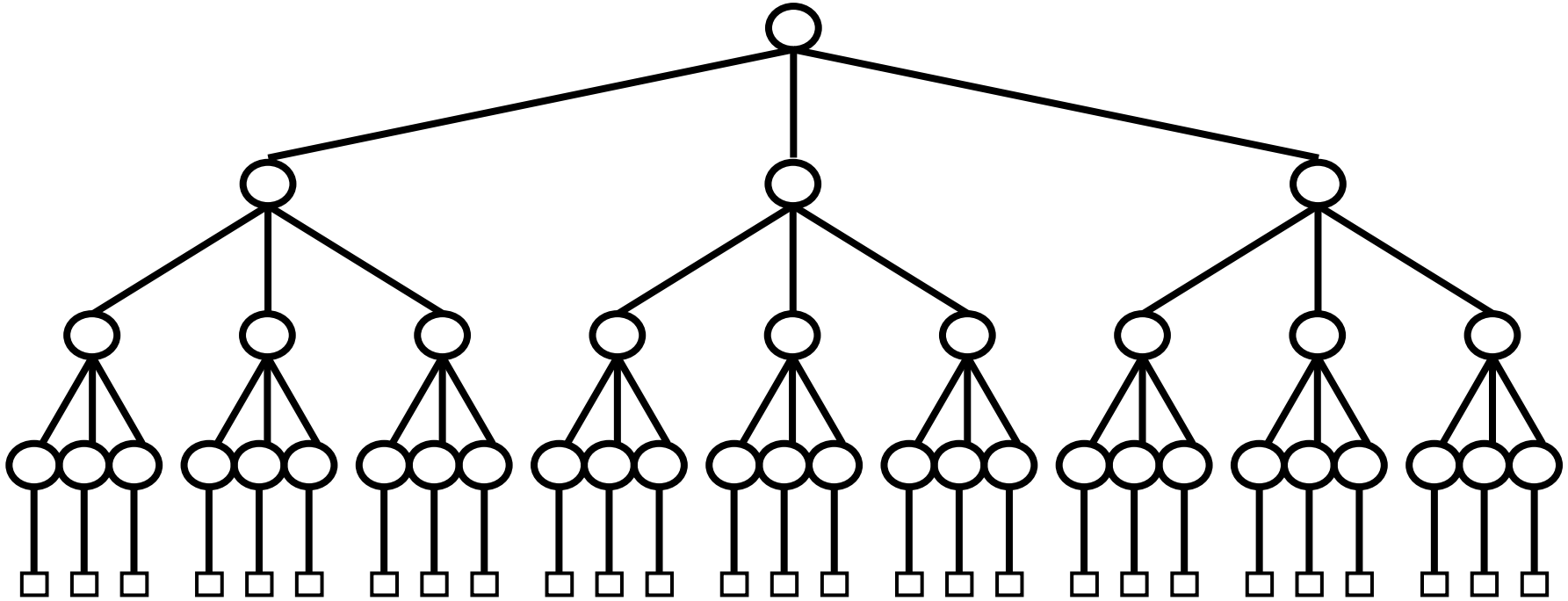
- 归并原理：把第一阶段所生成的顺串加以合并(例如通过若干次二路合并)，直至变为一个顺串为止，即形成一个已排序的文件
- 为一个待排文件创建尽可能大的初始顺串，可以大大减少扫描遍数和外存读写次数
- 归并顺序的安排也能影响读写次数，把初始顺串长度作为权，其实质就是 Huffman 树最优化问题

二路归并外排序



读写各：3*6 + 6*2 + （12 +6）= 48 次

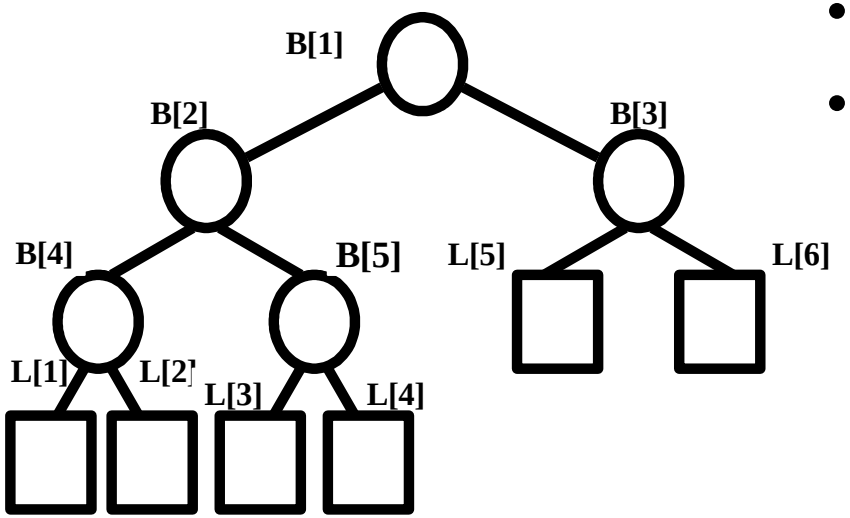
多路归并



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

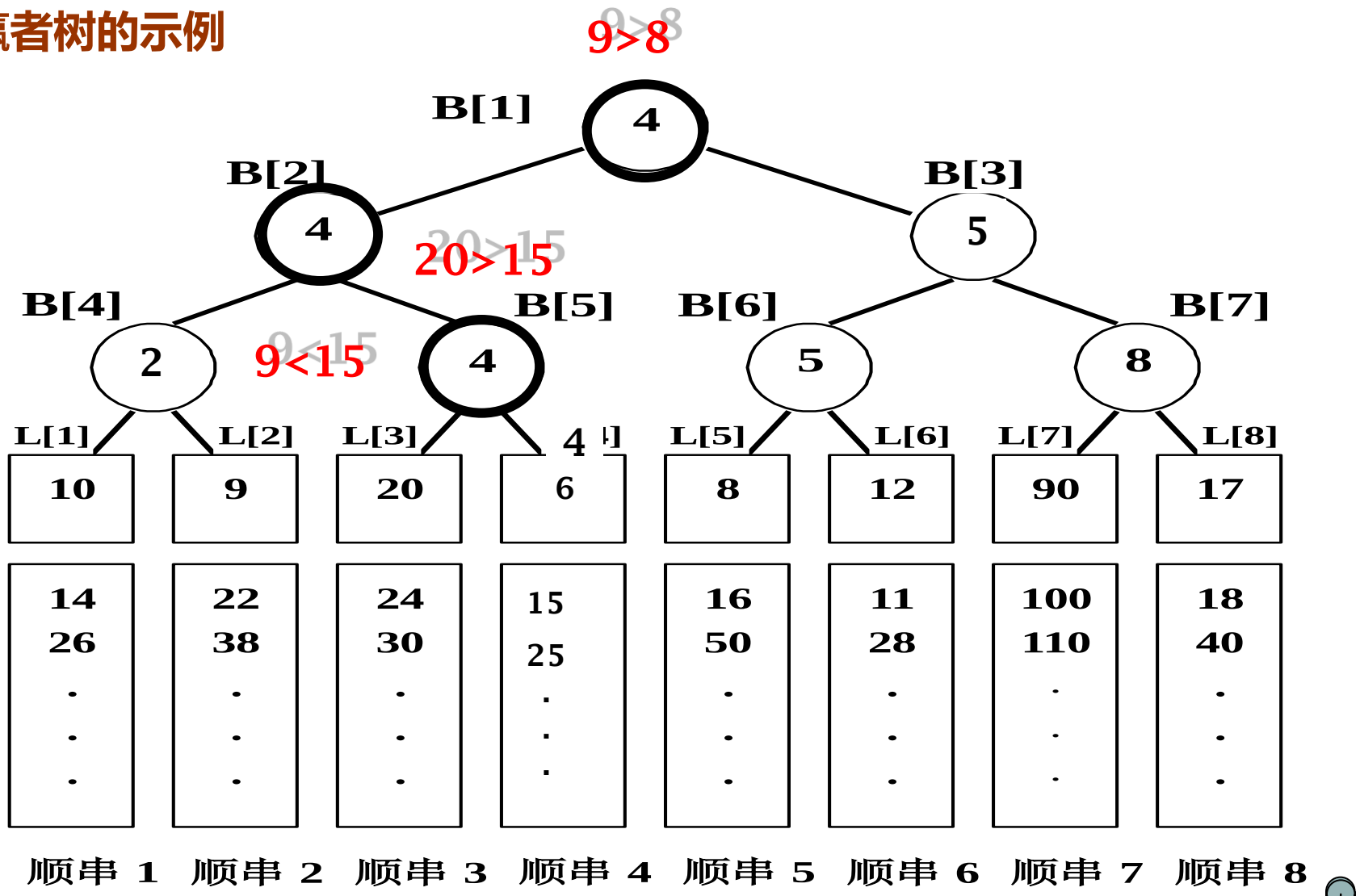


- n=6, LowExt=4, Offset=7
- LowExt + Offset = 2n-1

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：

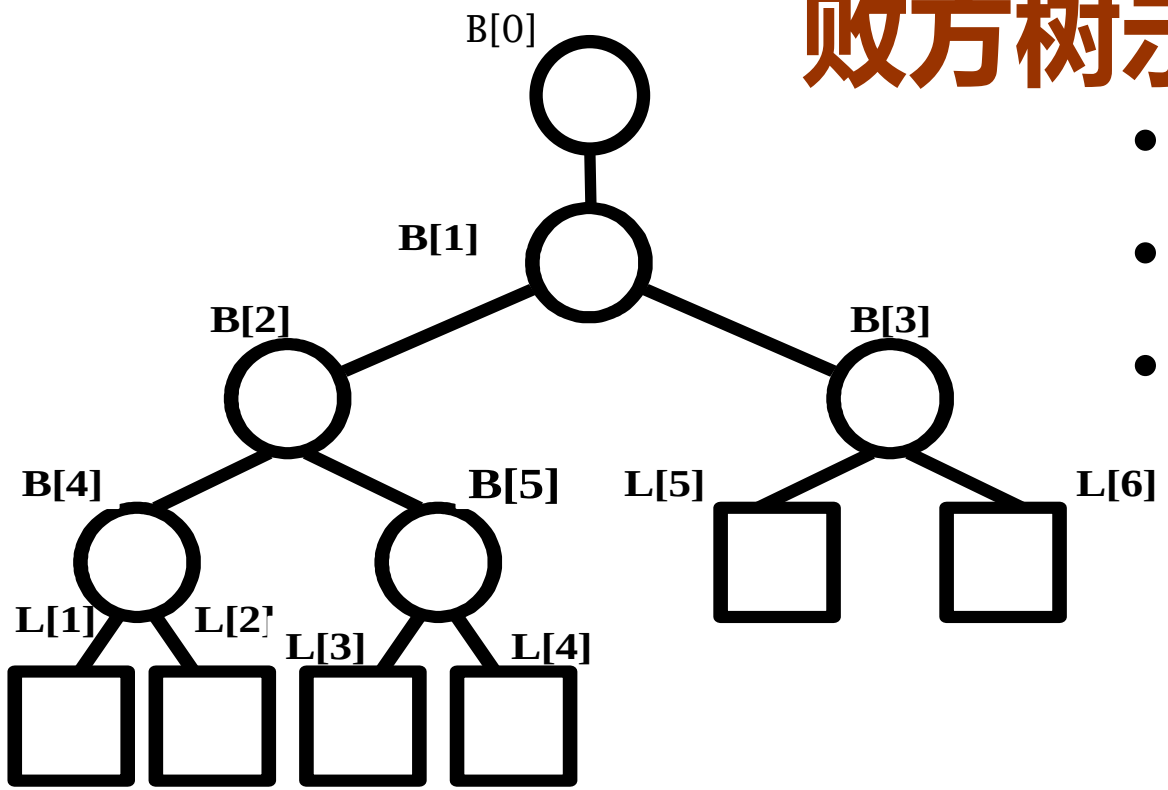
$$\begin{cases} (i+offset)/2 & i \le LowExt \\ (i-LowExt+n-1)/2 & i > LowExt \end{cases}$$

赢者树的示例



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。

败方树示例

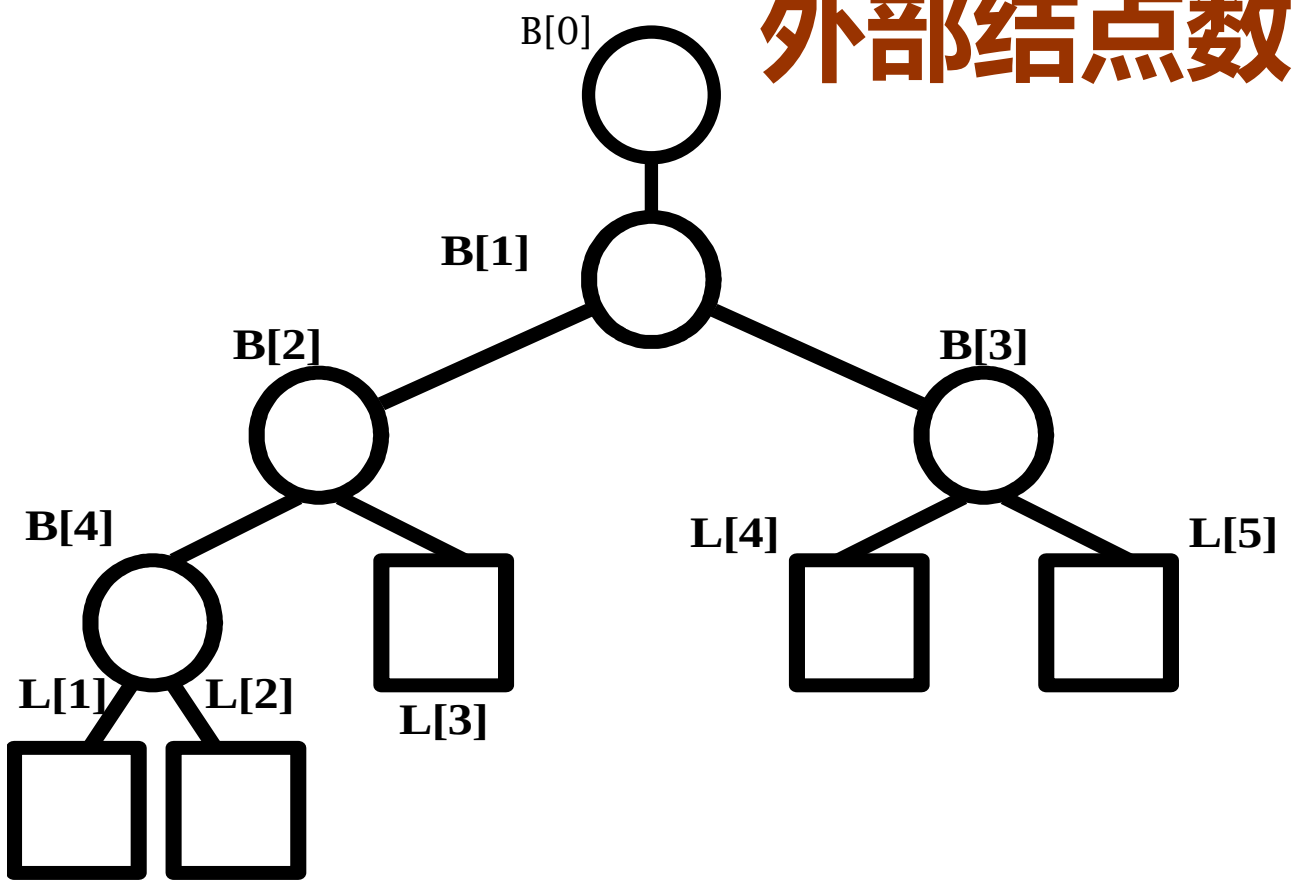


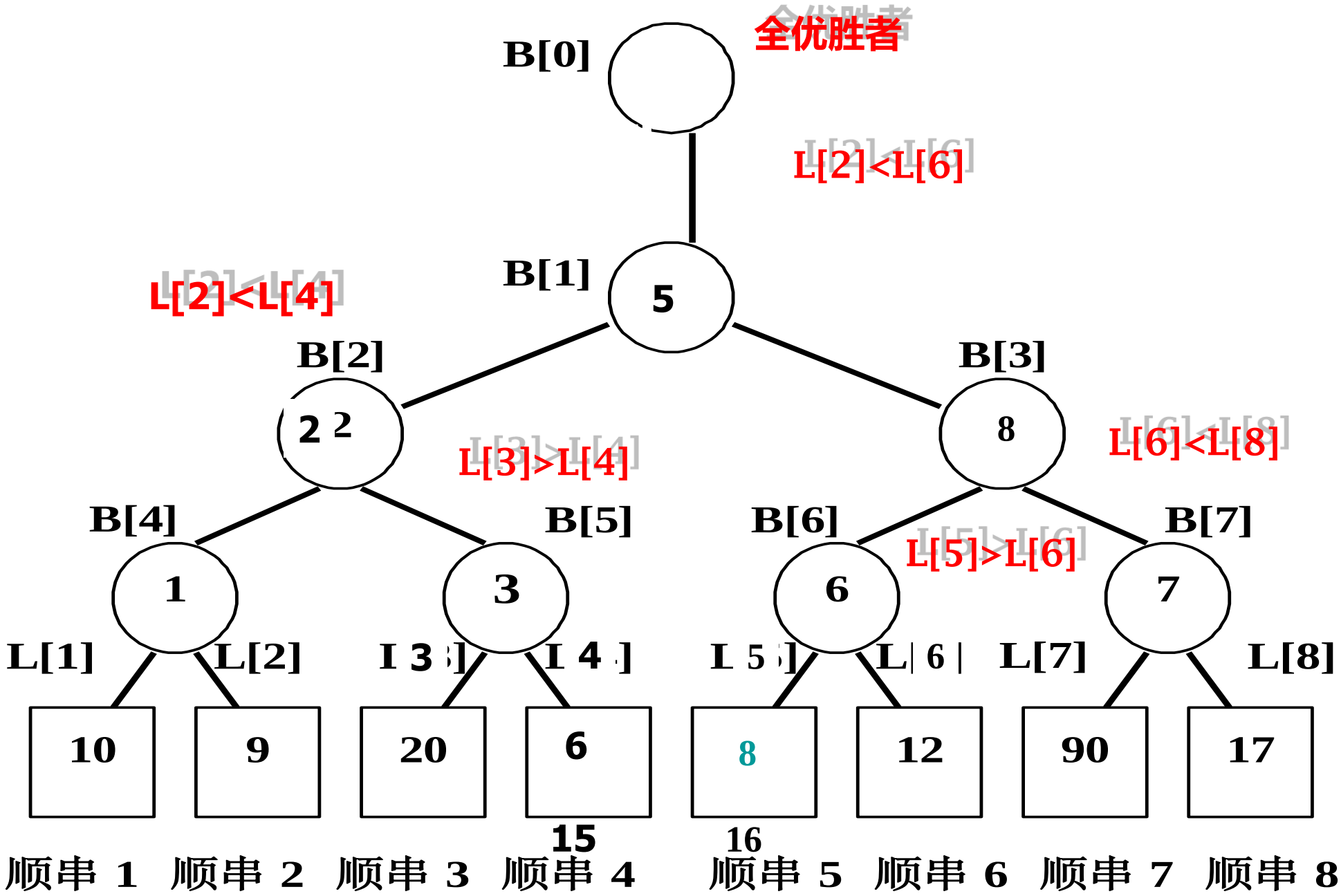
- n=6
- LowExt=4
- Offset=7

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

9.3 外排序

外部结点数n为奇数




全优胜者
B[0]
L[2]<L[6]
B[1]
5
L[2]<L[4]
B[2]
B[3]
2 2
8
L[3]>L[4]
L[6]<L[8]
B[4]
B[5]
B[6]
B[7]
L[5]>L[6]
1
3
6
7
L[1]
L[2]
L[7]
L[8]
10
9
20
6
8
12
90
17
15
16
顺串 1
顺串 2
顺串 3
顺串 4
顺串 5
顺串 6
顺串 7
顺串 8

败方树 ADT

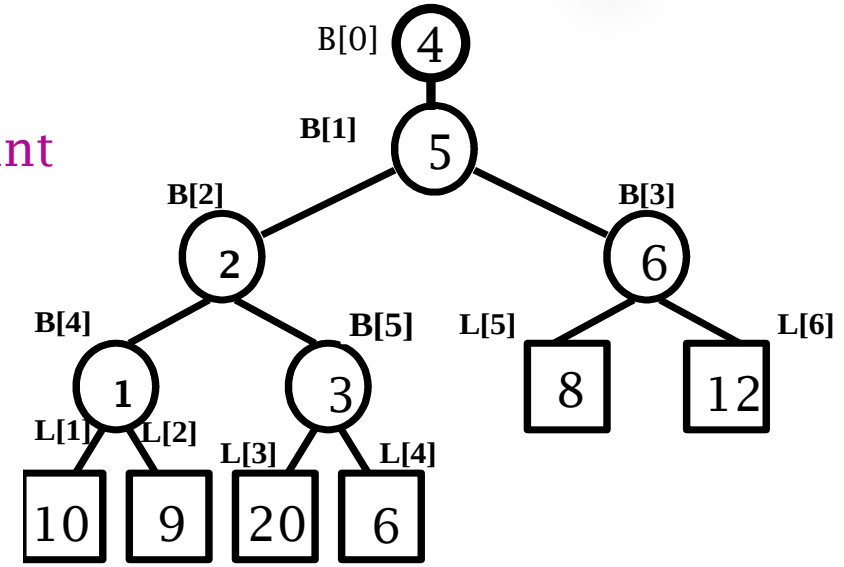
```
template<class T>
class LoserTree{
private:
    int MaxSize;          // 最大选手数
    int n;                // 当前选手数
    int LowExt;           // 最底层外部结点数
    int offset;           // 最底层外部结点之上的结点总数
    int * B;              // 败方树数组，实际存放的是下标
    T * L;                // 元素数组
    void Play(int p,int 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;
}
```

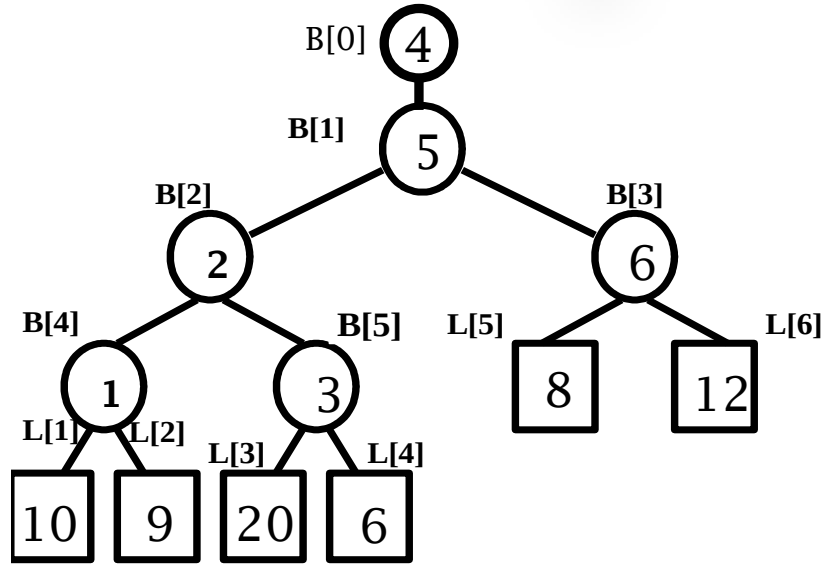
9.3 外排序

```
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);
}}
```



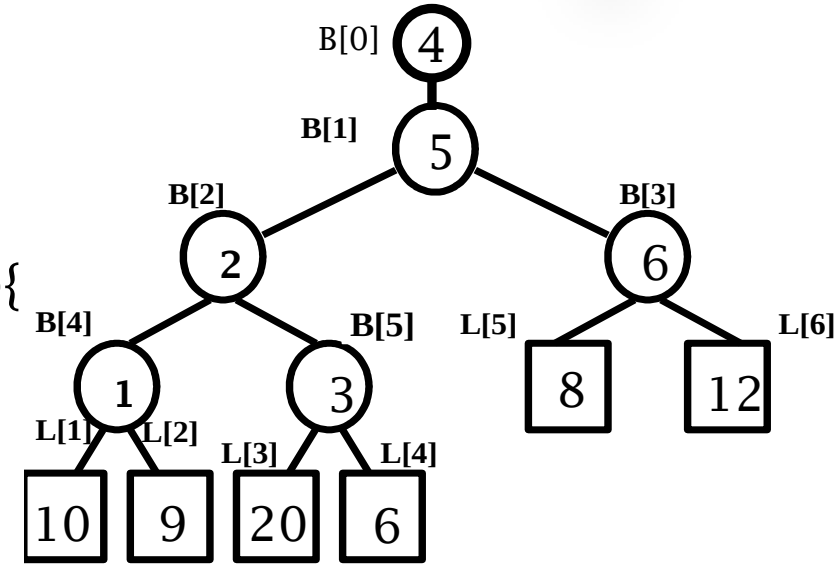
Play 比赛

```
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;
}
```



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

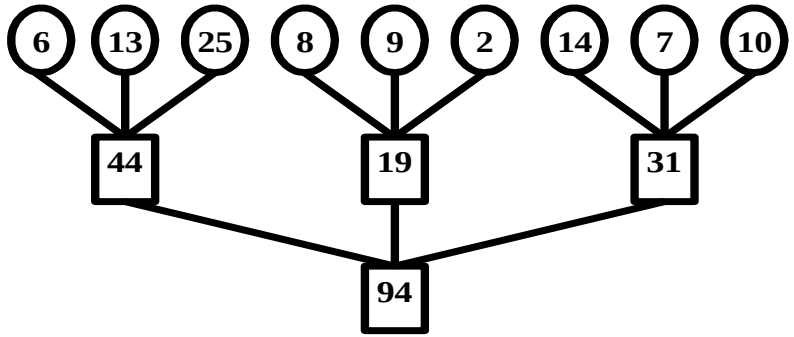
- 对同一个文件而言，进行外排序所需读写外存的次数与归并趟数有关系
- 假设有 m 个初始顺串，每次对 k 个顺串进行归并，归并趟数为 $\lceil \log_k m \rceil$
- 为了减少归并趟数，可以从两个方面着手：
  - 减少初始顺串的个数 m
  - 增加归并的顺串数量 k

9.3 外排序

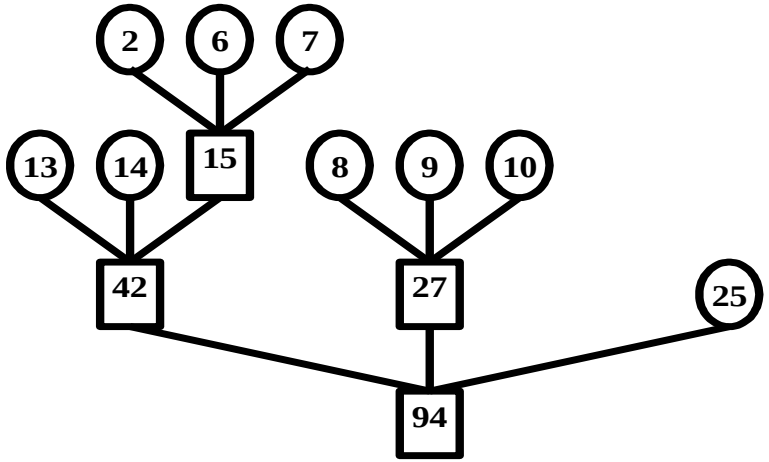
假设对 k 个顺串进行归并，归并后长 n

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

最佳归并树



(a)一棵普通的归并树



(b)最佳归并树

读写总次数次376

读写总次数次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