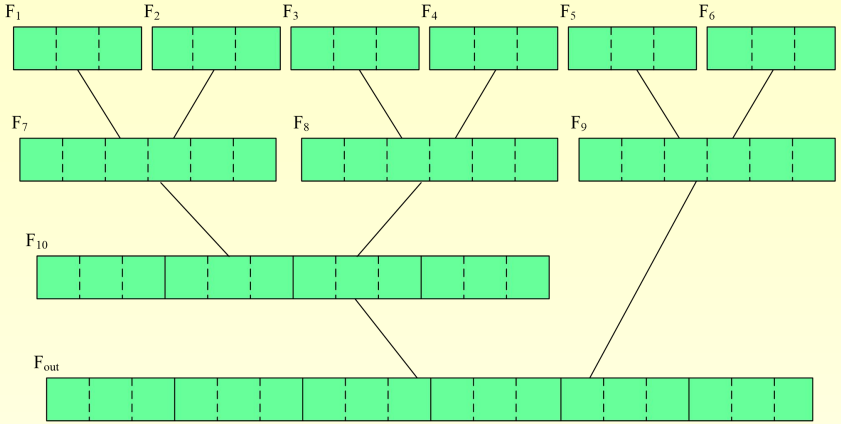
本节主题：

多路平衡归并

由二路归并看复杂度

- 前提
  - 初始归并段有m个
- 采用二路平衡归并
  - 归并树有$\lceil \log_2 m \rceil+1$层
  - 要对数据进行$\lceil \log_2 m \rceil$遍扫描
- 采用k路平衡归并
  - 归并树有$\lceil \log_k m \rceil+1$层
  - 要对数据进行$\lceil \log_k m \rceil$遍扫描



k-路归并排序算法

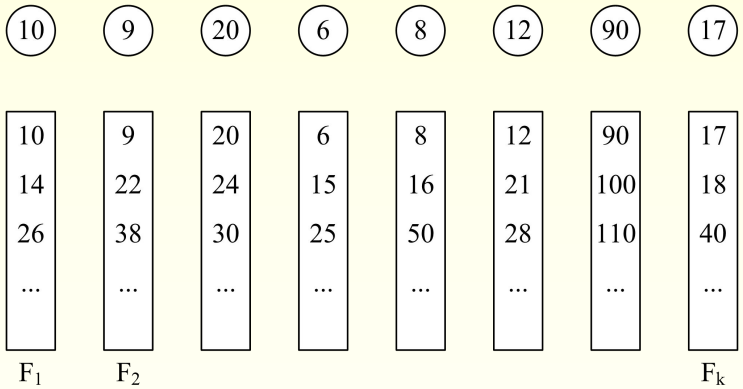
- 前提
  - 各路缓冲区中存储了有序的归并段
  - 在每一路记录当前正在处理元素的位置，从而确定当前该路的最小记录
  - 每一路的最小记录，为排序中要考察的当前记录
- 归并排序算法
  - 从各路缓冲区中，找出关键字最小值的记录
  - 将最小记录送到输出缓冲区
  - 改变该路的位置指针
  - 重复上面的步骤，直到各路数据全部为空

- k个记录中选择最小者，需要顺序比较k-1次
- 每趟归并u个记录需要做(u-1)*(k-1)次比较
- 初始归并段有m个，需要 $s=\lceil \log_k m \rceil$趟归并
- s趟归并总共需要的比较次数为

$$s*(u-1)*(k-1)=\lceil \log_k m \rceil*(u-1)*(k-1)$$
$$=\lceil \log_2 m \rceil*(u-1)*(k-1) / \lceil \log_2 k \rceil$$

  - $(k-1) / \lceil \log_2 k \rceil$在k增大时趋于无穷大

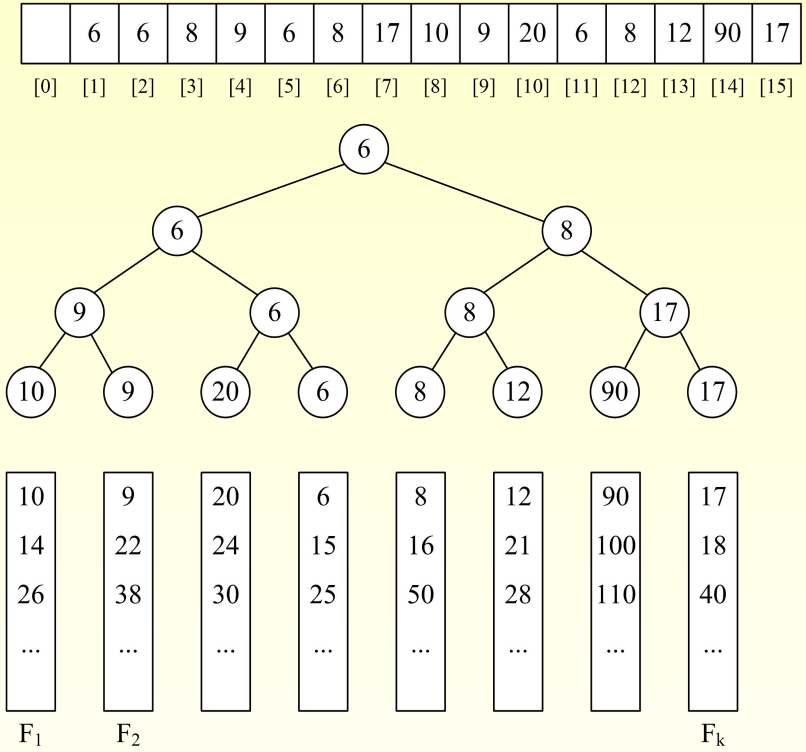
不盲目扩大k



利用胜者树的多路归并

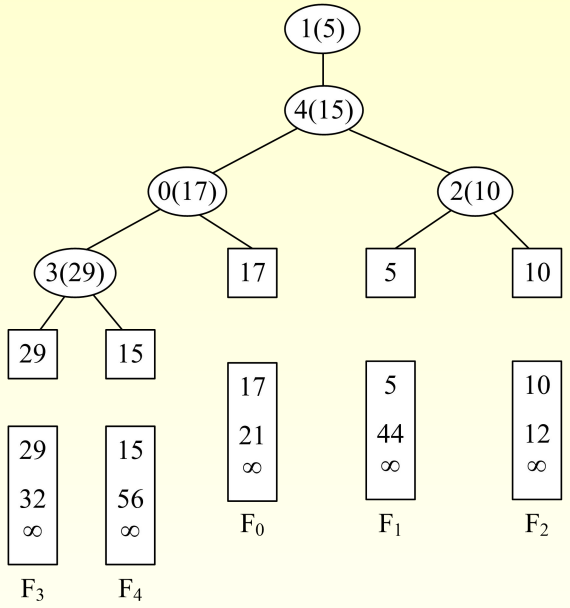
- 目标
  - 减少获得k路中最小值的比较次数
- 策略
  - 读入各归并段，构造胜者树
  - 归并胜者树根节点
  - 归并后调整胜者树
  - 某归并段取尽时，将∞写入胜者树对应的叶节点
  - 所有归并段取尽时，读入下一组待归并记录，重建胜者树
- 效果
  - 除建胜者树需要中k-1次比较，其余仅需要$\lceil \log_2 k \rceil$次比较

$$s*(u-1)*(k-1)=\lceil \log_k m \rceil*(u-1)*\mathbf{(k-1)}$$
$$=\lceil \log_2 m \rceil*(u-1)*\mathbf{(k-1)}\ /\lceil \log_2 k \rceil$$



败者树——利用败者树的k路平衡归并

- 败者树
  - 一棵有k个叶子节点的完全二叉树
  - 叶子节点存储记录
  - 分支节点存放关键字及对应的段号
    格式：段号(关键字)
  - 双亲节点记录败者（关键字较大者），而让胜者参加更高一级的比较。
- 建立初始败者树
  - 取每个输入有序段的第一个记录作为败者树的叶子结点
  - 两两叶结点进行比较，在双亲结点中记录比赛的败者(关键字较大者)，而让胜者去参加更高一层的比赛，如此在根结点之上胜出的“冠军”是关键字最小者。



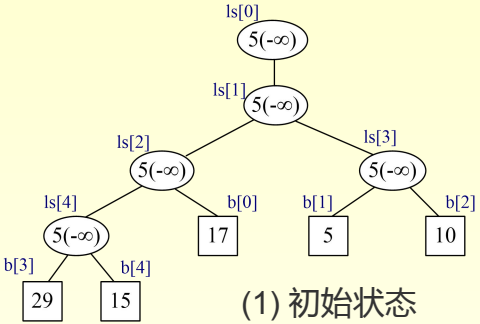
建立败者树

ls[0]: 存放冠军节点　ls[1]~ls[4]: 存放分支节点
b[0]~b[1]: 存放叶子节点

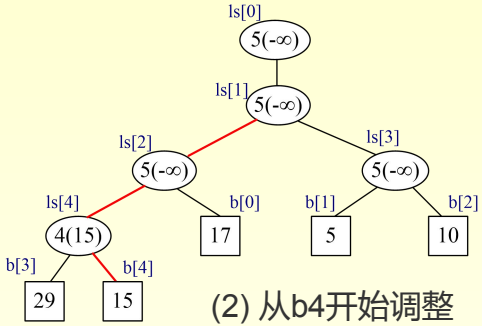
5(-∞)
含最小关键字的虚拟段

例如，设有5个初始归并段，它们中各记录的关键字分别是（∞是段结束标志）：

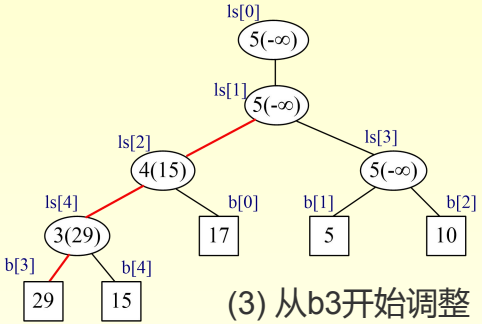
R0:{17,21,∞} R1:{5,44,∞}　R2:{10,12,∞} R3:{29,32,∞} R4:{15,56,∞}



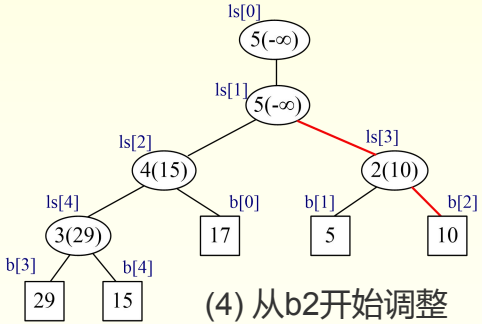
(1) 初始状态



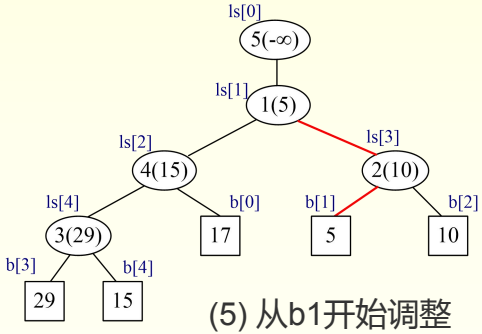
(2) 从b4开始调整



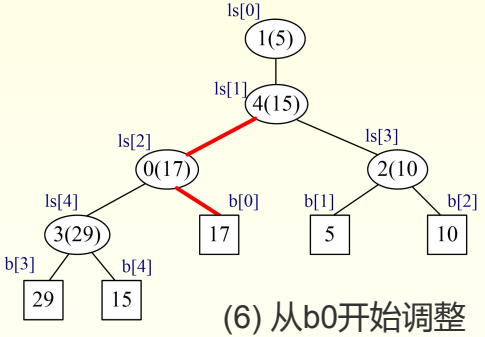
(3) 从b3开始调整



(4) 从b2开始调整



(5) 从b1开始调整



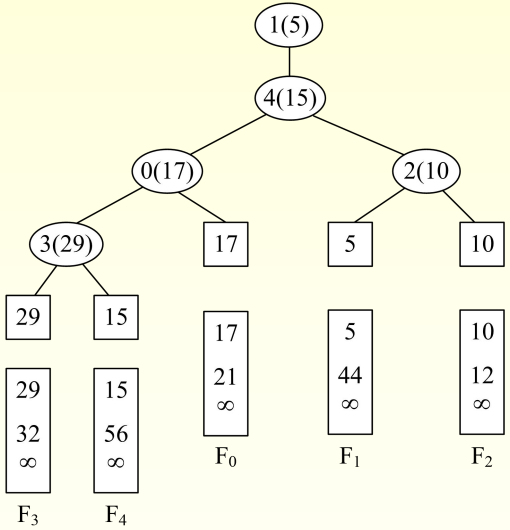
(6) 从b0开始调整

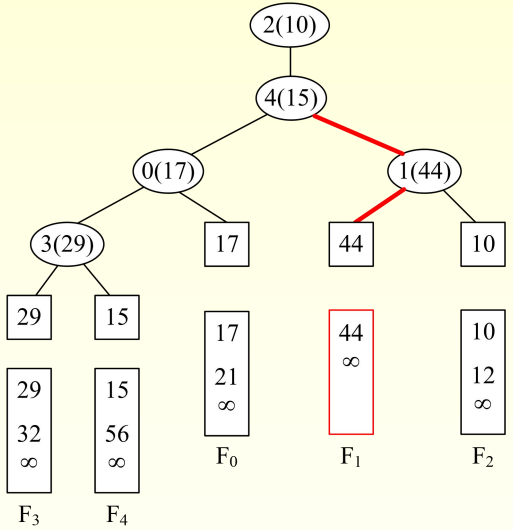
对k个输入有序段进行k路平衡归并的方法

(1)胜出的记录写至输出归并段，在对应的叶结点处，补充其输入有序段的下一个记录。若该有序段变空，则补充一个大关键字(比所有记录关键字都大，设为k_{max})的虚记录。

(2) 调整败者树，选择新的关键字最小的记录：从补充记录的叶结点向上和双亲结点的关键字比较，败者留在该双亲结点，胜者继续向上，直至树根的双亲。

(3) 若胜出的记录关键字等于k_{max},则归并结束；否则转(1)继续。





性能分析

- 效果
  - 从上例看到，k路平衡归并的败者树的深度为$\lceil \log_2 k \rceil$，在每次调整找下一个具有最小关键字记录时，最多做$\lceil \log_2 k \rceil$次关键字比较。
  - 因此，利用败者树在k个记录中选择最小者，只需要进行$O(\lceil \log_2 k \rceil)$次关键字比较，这时归并总共需要的比较次数为：

$$
\begin{aligned}
s*(u-1)*\lceil \log_2 k \rceil &= \lceil \log_k m \rceil *(u-1)*\lceil \log_2 k \rceil \\
&= \lceil \log_2 m \rceil *(u-1)*\lceil \log_2 k \rceil / \lceil \log_2 k \rceil \\
&= \lceil \log_2 m \rceil *(u-1)
\end{aligned}
$$

- 结论
  - 关键字比较次数与k无关，总的内部归并时间不会随k的增大而增大。
  - 只要内存空间允许，增大归并路数k，将有效地减少归并树的深度，从而减少读写磁盘次数，提高外排序的速度。