数据结构

第十八讲 多分支排序树

孙猛

http://www.math.pku.edu.cn/teachers/sunm

2017年12月11日

课程内容

- 字典与索引
- 多分树
- B树和B+树

字典与索引

- 不等长结点的问题
  - 在关于字典的讨论中，为简单起见，可将所有元素关键码的类型和属性值的类型都定义为整型。
  - 实际应用时，关键码可以为其它类型，属性值同样可以出现为多种形式。
  - 不同的值需要的空间大小不同，这样的字典就难以采用前面介绍的存储表示。

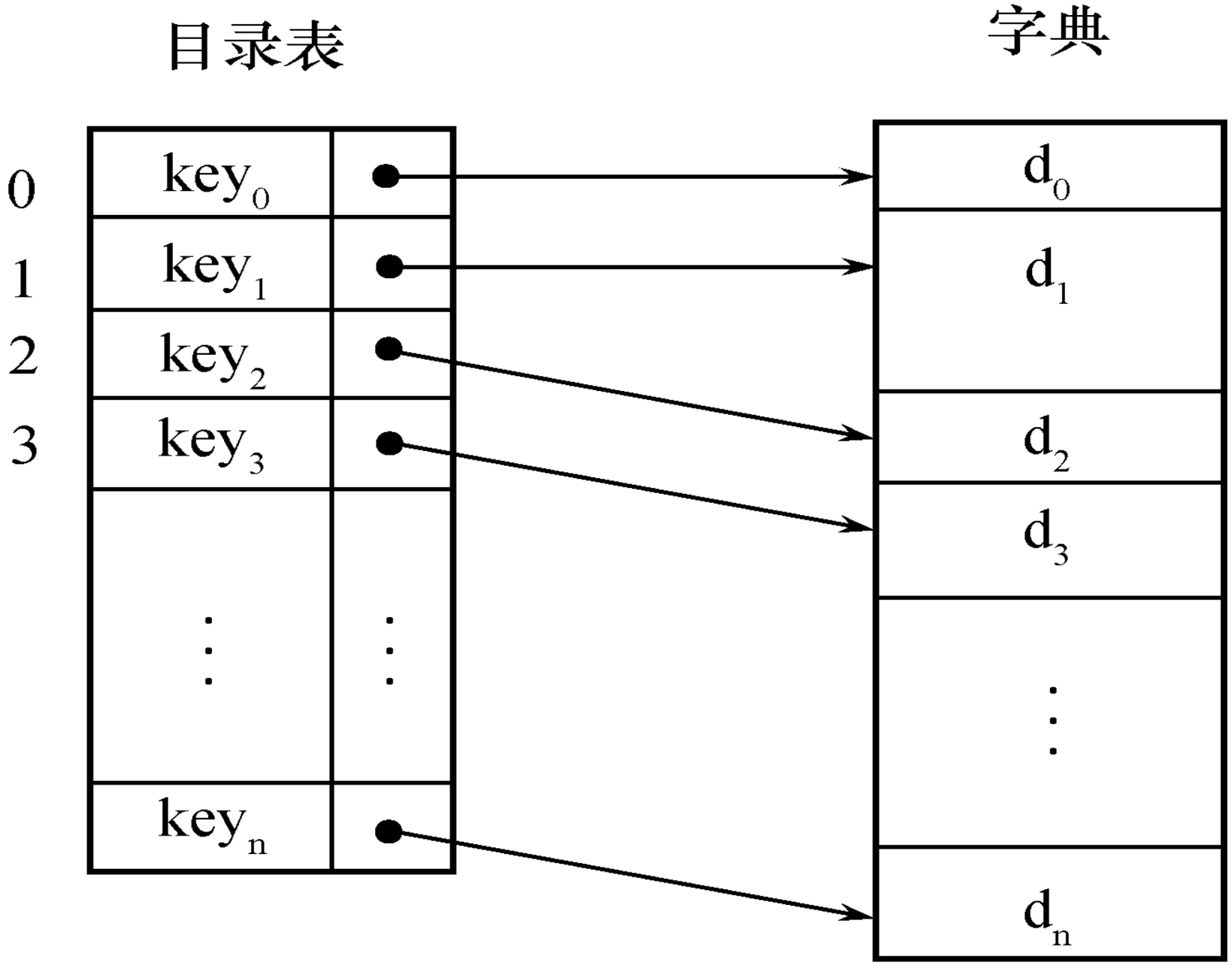
举例

- 在字典的顺序表示，如果这个字典中，元素的值需要空间的长度不等，可以另外建立一个字典的索引——通常称为目录表。

- 增加了目录表后，字典本身可以是顺序存储，也可以用其它方式存储。

(见下页)

引入了目录表（索引）后的字典



索引的作用

- 引入索引就可以将包含大量属性信息并且不等长元素的字典的处理，转换成对仅仅包含关键码到地址对应关系（简单类型并且等长的元素）的索引结构的处理。
- 在检索一个元素时，只要在目录表中找到对应的关键码，马上可以得到对应结点的存储位置；
- 在排序过程中，只要完成目录表中元素（又称索引项）的排序，而不需要移动字典本身的任何结点。

索引与索引项

- 索引是索引项的集合，
- 一个索引项是由一个结点的关键码和该结点的存储位置组成的关联。
- 索引的实质还是字典，而且是元素类型相同的等长结点的字典。
- 所有关于字典的讨论都适合于索引；所有字典的实现也可以用来组织索引。

密集索引与稀疏索引

- 每个索引项都是对应字典中一个元素，这种索引称为**密集索引**；

- 如果每个索引项对应字典中一组元素，这种索引称为**稀疏索引**。

索引文件

- 文件是记录的集合。
- 文件中的记录可以按某种顺序（例如按放入文件的先后或者按关键码的大小）定义次序，并且在外存储器上是按同样的次序存放的，则这种文件叫做顺序文件。
- 索引文件. 对于文件中的记录也可以建立索引。对于大型的文件，其索引往往也是很大的，需要存放在外存上。这样索引也是一种文件。
- 我们把索引和主文件（存放实际记录的文件）总称为索引文件。对于索引文件，其主文件如果是一个顺序文件，又可以叫作索引顺序文件。
- 所谓索引的索引就是给庞大的（通常是密集的）索引，建立另外一个辅助（通常是稀疏）的索引结构，以达到加快查找字典中特定结点的目的。

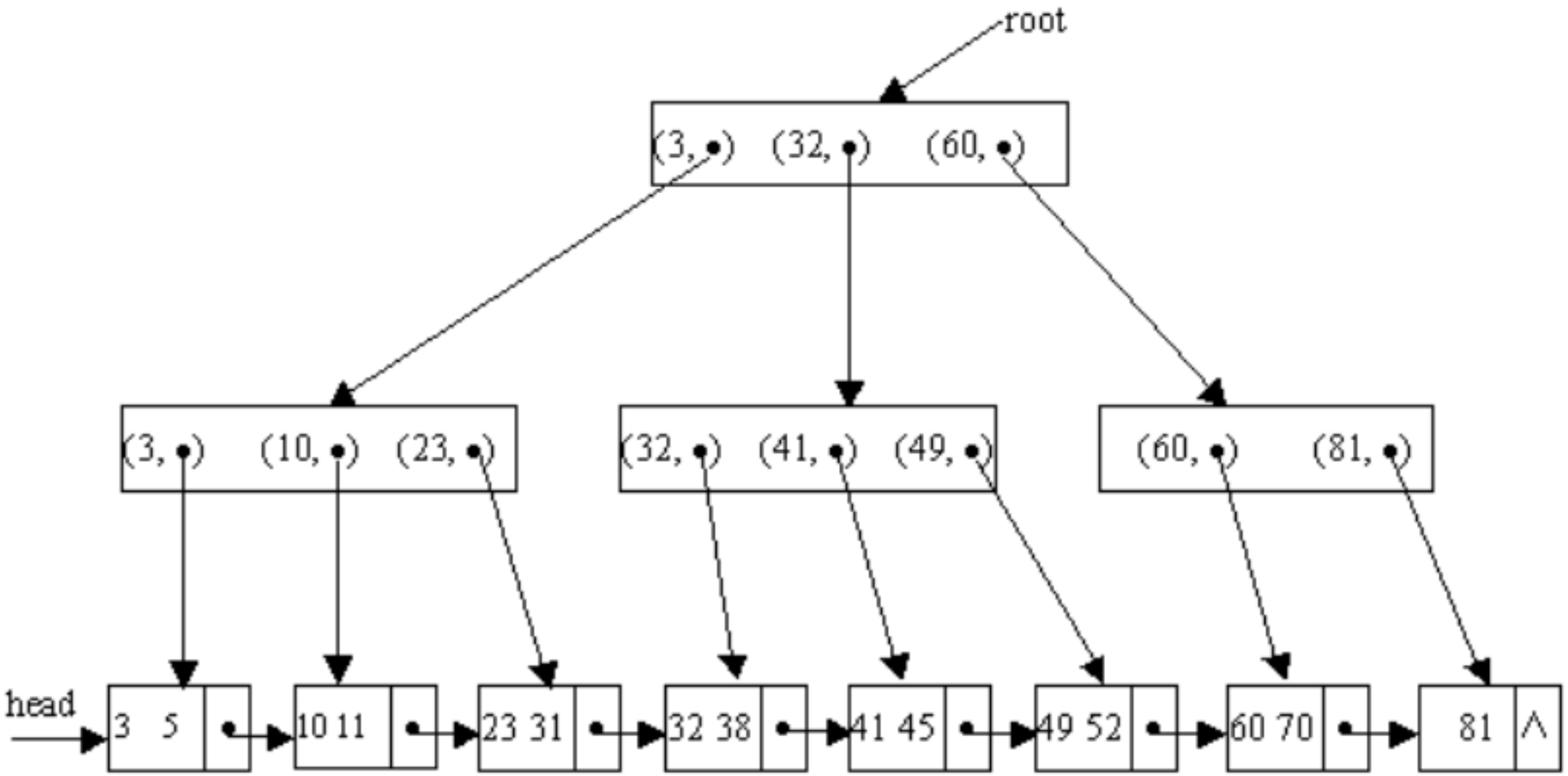
文件的处理

- 检索：在文件中查找满足一定条件的记录。
  - 最常见的是，查找关键码为一指定值的记录，此时，成功的检索只能找出唯一的记录。
- 插入：在文件中增加一个新的记录。
- 删除：删去文件中的一个记录。
  - 插入和删除操作常常需要通过检索，先找到被插入或删除记录的位置。
- 修改：把记录中某些属性的值改为新值。
  - 若被修改的属性包括关键码，则相当于删除一个老记录并插入一个新记录。
- 排序：按照关键码或者某属性的值，从小到大（或从大到小）的次序，把文件的全部记录重新进行排列。这种排序叫做外排序。

多分树

- 如果文件很大，索引顺序文件的索引可以采用多级索引结构以提高检索的效率。
- 即为主文件建立了索引之后，又针对本级索引所占的每一个页块建立一个索引项，用这些索引项构成索引的索引。
- 如果新建的这一级索引仍然占用多个页块，则再为它建立索引。这样可以得到一种多级索引结构——多分树。
- 如果每个内部结点（根除外）有m个子女，则称为m分树。

由三分树组成的多级索引结构



表示方法

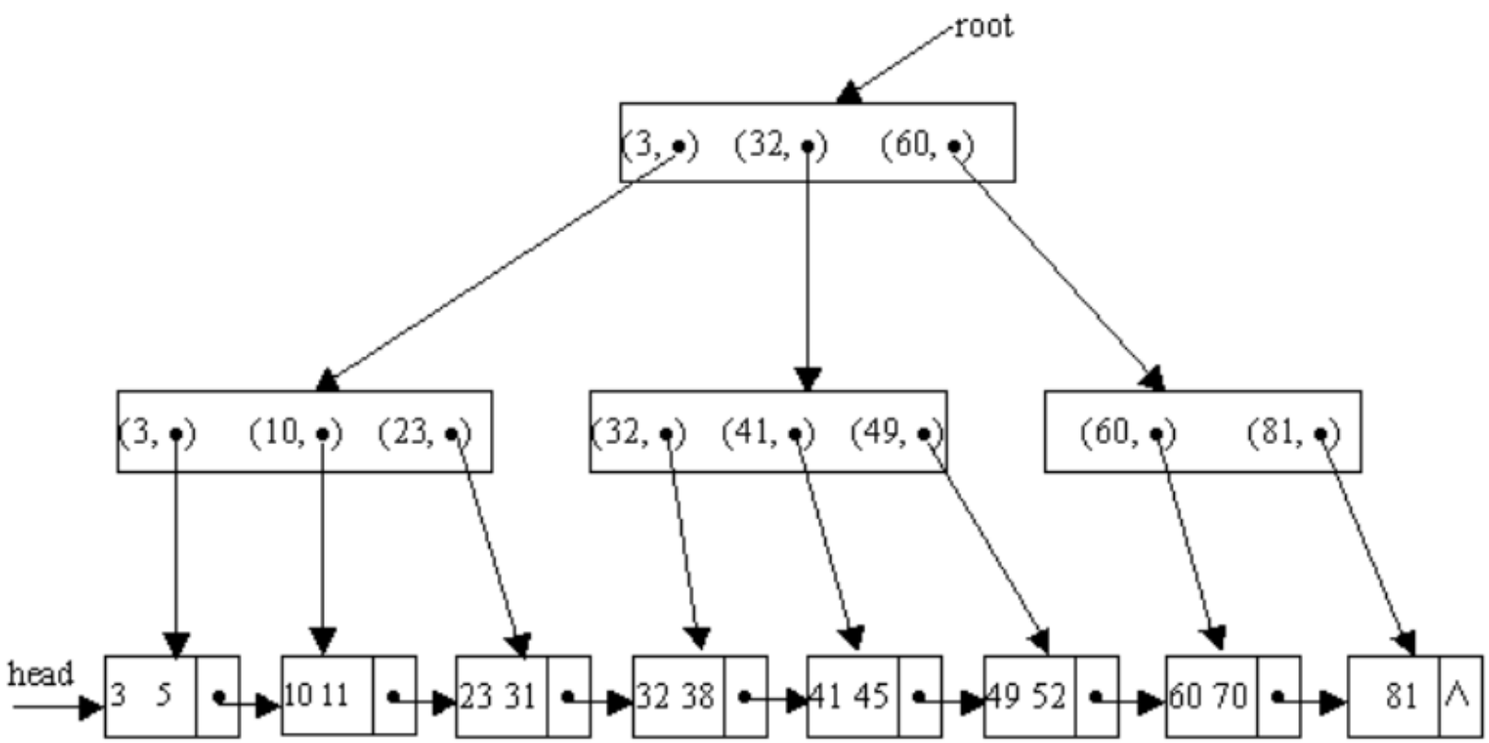
- 多分树的每个结点分配一个页块，结点上的信息是许多二元组 (k,p) 的序列，它们在结点内按关键码 k 的值排序。
- 第 i 个二元组中的 p 是本结点的第 i 个子结点（页块）的地址，k 是这个子结点上的最小（或者最大）关键码。
- 多分树的叶结点就是主文件的最底层索引。
- 主文件的每个页块可以看成是多分树的外部结点。

检索关键码为K的记录

- 先读入根结点页块，从中找到满足 $k \leq K$ 的最后索引项 (k,p) 后读入 p 所指页块(下一级索引)；
- 这样直到读入主文件页块，在其上查找关键码为 K 的记录。
- 查阅索引的访外次数等于多分树的高度加1。m分树的高度大约为 $log_m N$，其中 N 为最后一级索引所占的页块数。

插入

- 在这种文件上插入记录，为了使主文件的记录保持关键码递增的次序，需要把插入位置之后的每个记录向后移动，从而导致增加新的索引项和索引页块，需要对外存上的页块进行大量的调整。

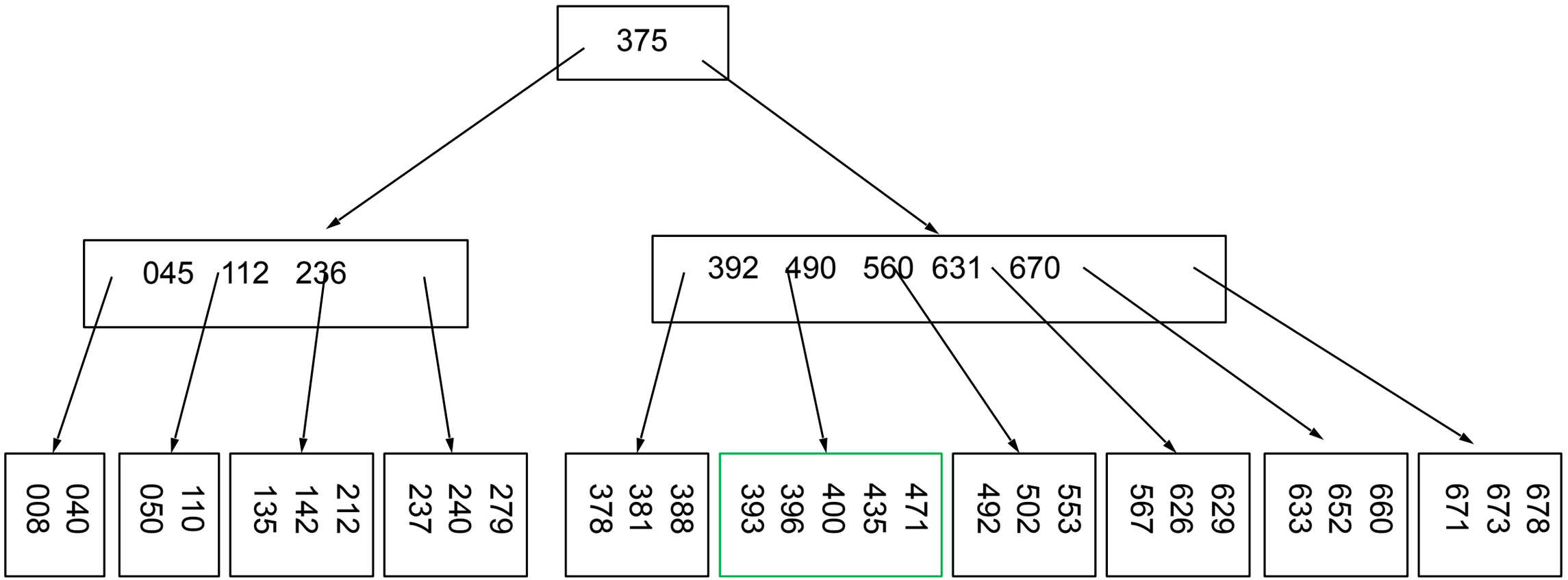


讨论

- 多分树主要用于静态的索引顺序文件。

- 对于经常需要插入和删除的动态索引顺序文件，需要采用动态索引结构，即下面要讨论的B树和B+树。

B树


375
045 112 236
392 490 560 631 670
040 008
110 050
212 142 135
279 240 237
388 381 378
471 435 400 396 393
553 502 492
629 626 567
660 652 633
678 673 671

一棵6阶的B树

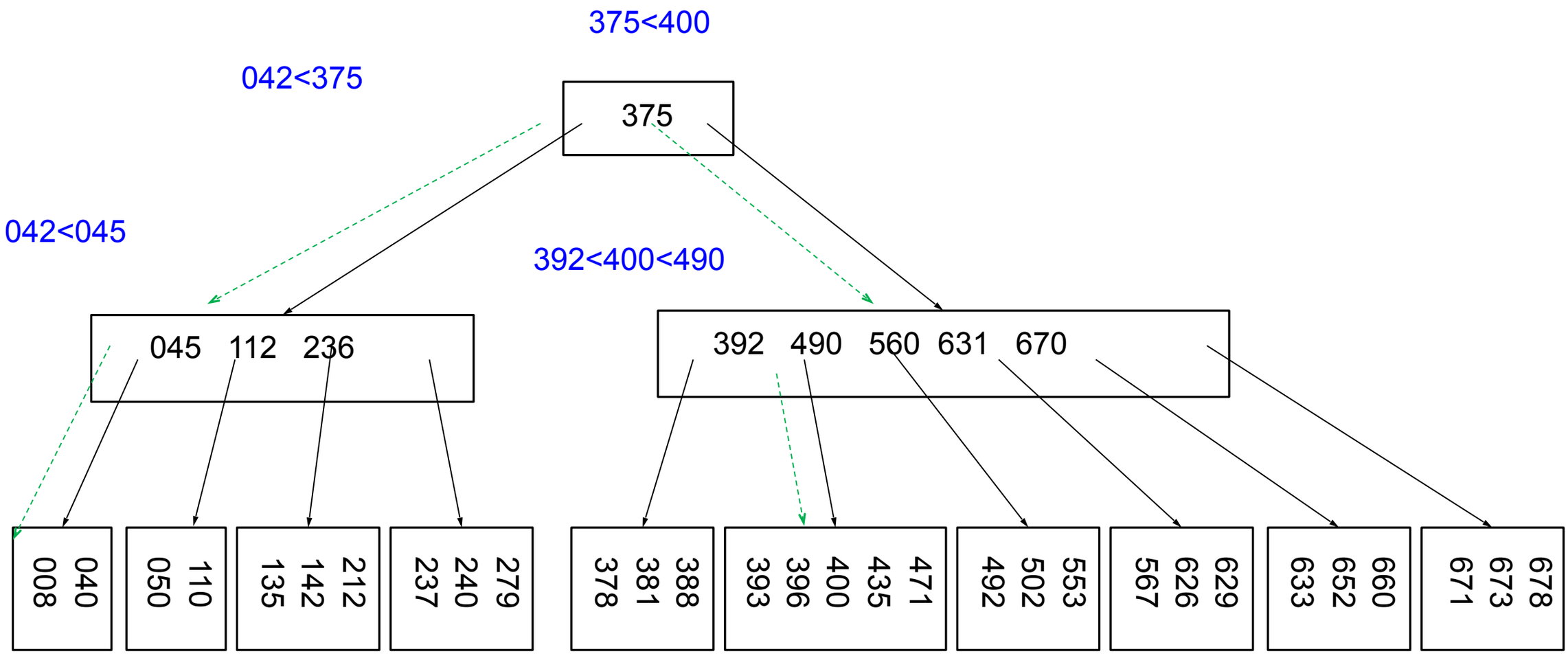
B树的定义

- 一棵m阶的B树满足下列条件：
  - 每个结点至多有m棵子树。
  - 除根结点外，其它每个分支结点至少有$\lceil m/2 \rceil$棵子树。
  - 根结点至少有两棵子树(除非B树只包含一个结点)。
  - 所有叶结点在同一层上。B树的叶结点可以看成一种外部结点，不包含任何信息。
  - 有j个孩子的非叶结点恰好有$j-1$个关键码，关键码按递增次序排列。

检索

- 在B树中检索关键码key的思想：
- 根据要查找的关键码key，在根结点的关键码集合中进行顺序或二分法检索，若$key = k_i$，则检索成功；
- 否则，key一定在某k_i和k_{i+1}之间，取与k_i所对应的p_i所链接的结点继续查找；
- 重复上述检索过程，直到检索成功，或p_i为空链接，检索失败。

在B树里检索关键码400，042



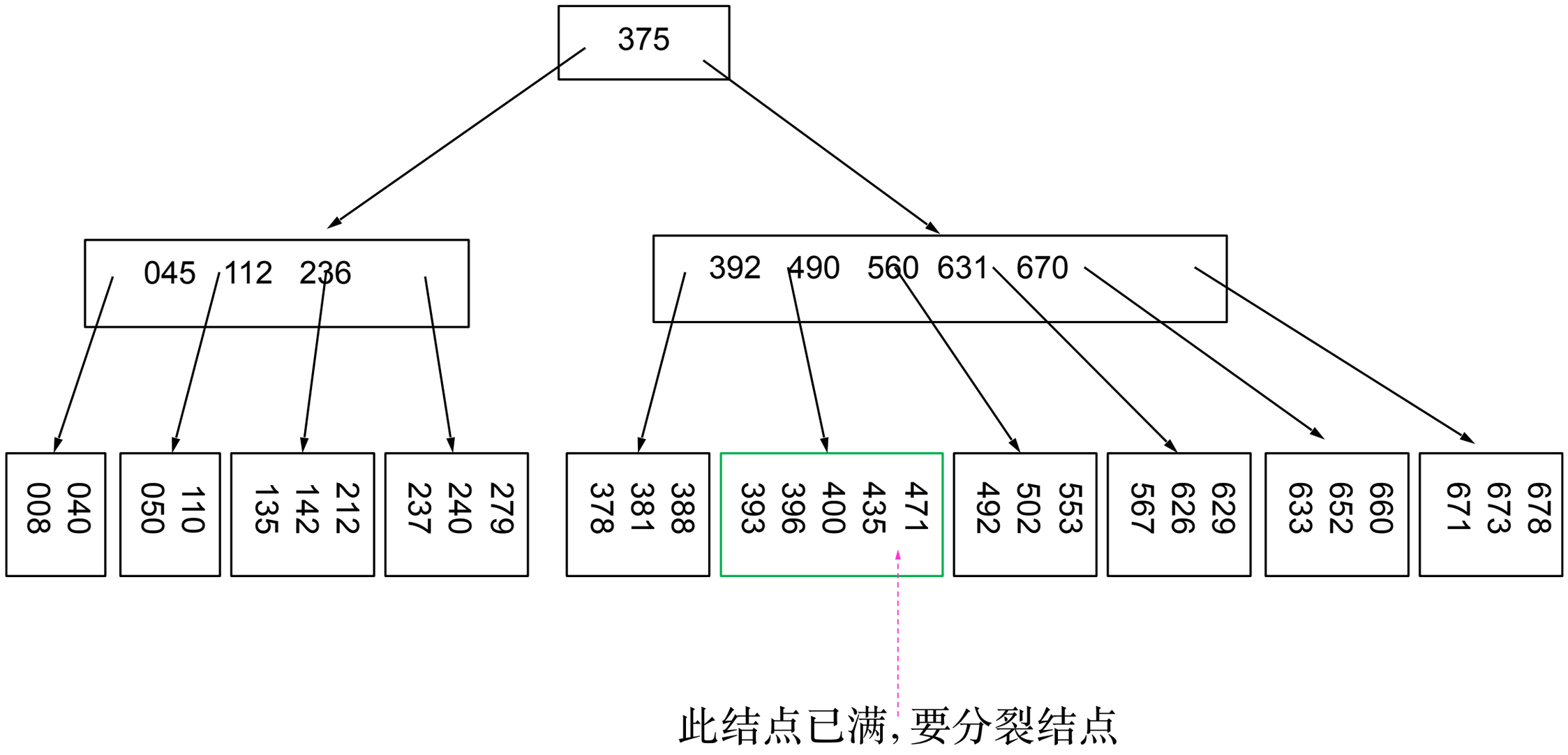
检索042
失败

检索400
成功

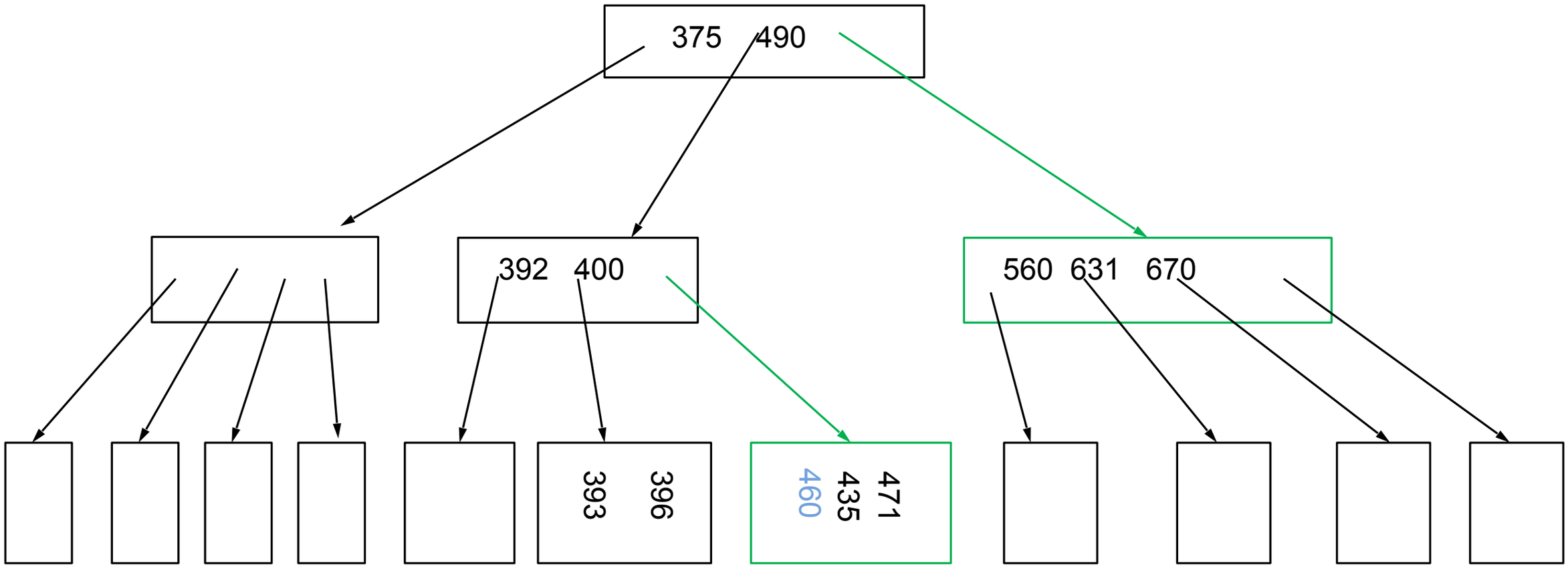
B树的插入运算

- 在B树中插入关键码*key*的思想：高度为*h*的*m*阶B树，新结点一般插入到第*h*层。通过检索确定关键码应插入的结点位置。分两种情况：
  - 若应该插入的结点中关键码个数小于*m*-1，直接插入；
  - 若被插入结点中关键码个数等于*m*-1，则引起结点分裂：
    - 将结点关键码从小到大分为3部分，中间1个关键码，左右两部分关键码个数相同(或相差1)；
    - 建一个新结点,把关键码大的那一部分移入新结点；
    - 把中间关键码和新结点链接一起插入原结点的父结点；
  - 插入父结点的情况与上面一样，如果父结点已满，它也会分裂并把一个关键码向上传；
  - 如果这种分裂传播到根，导致根结点分裂，B树增高一层。

在6阶B树中插入460



在6阶B树中插入460



插入460以后的B树

B树的删除运算

- 在高度为h的m阶B树中删除一个关键码key，首先检索到key所在的结点，然后根据不同情况进行删除：

(1) 如果该结点在第h层，根据情况分别处理。该结点：

①关键码数大于$\lceil m/2 \rceil-1$，则简单删去该关键码 k_i；

②关键码数等于$\lceil m/2 \rceil-1$，其左(或右)兄弟结点中的关键码数多于$\lceil m/2 \rceil-1$，删除关键码后把该结点的左(或右)兄弟结点和父结点的信息调整过来，维持树结构；

③兄弟结点中的关键码数均等于$\lceil m/2 \rceil-1$，需合并。设右兄弟结点由父结点中链接p_j所指。删去关键码后把该结点剩余关键码和链接连同父结点中的关键码k_j，一起合并到p_j所指结点中(没有右兄弟时合并到左兄弟结点)；

④如果因为合并引起其父结点的子树个数减少，则需要在父结点层考虑调整和合并。

B树的删除运算

(2) 若结点层数小于h，设删除的是结点中第i个关键码k_i，则用p_i所指的子树中最小关键码k与k_i互换，然后像前面一样处理h层的删除关键码k_i。

情况（1）—② 的调整方法：设父结点中的信息为：

$$(p_0, k_1, p_1, k_2, p_2, \ldots, k_x, p_x),$$

由p_i指向被删除关键码的结点，其信息为：

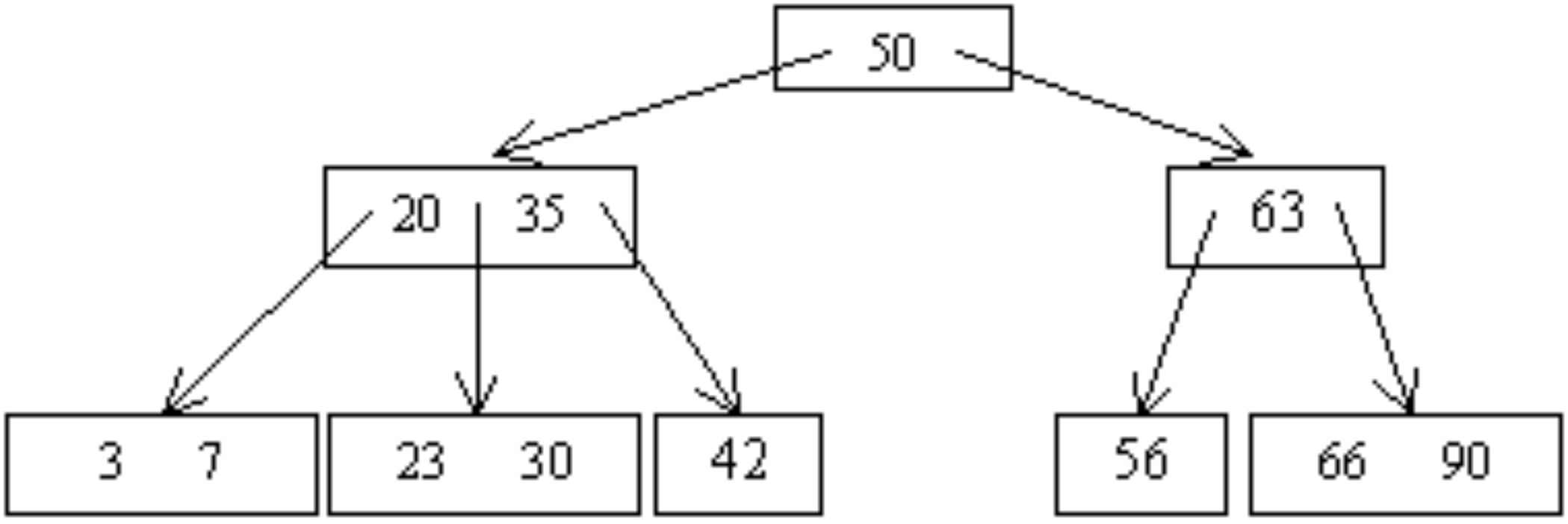
$$(p'_0, k'_1, p'_1, k'_2, p'_2, \ldots, k'_y, p'_y),$$

左兄弟结点中的信息为：

$$(p''_0, k''_1, p''_1, k''_2, p''_2, \ldots, k''_z, p''_z), \quad z > \lceil m/2 \rceil$$

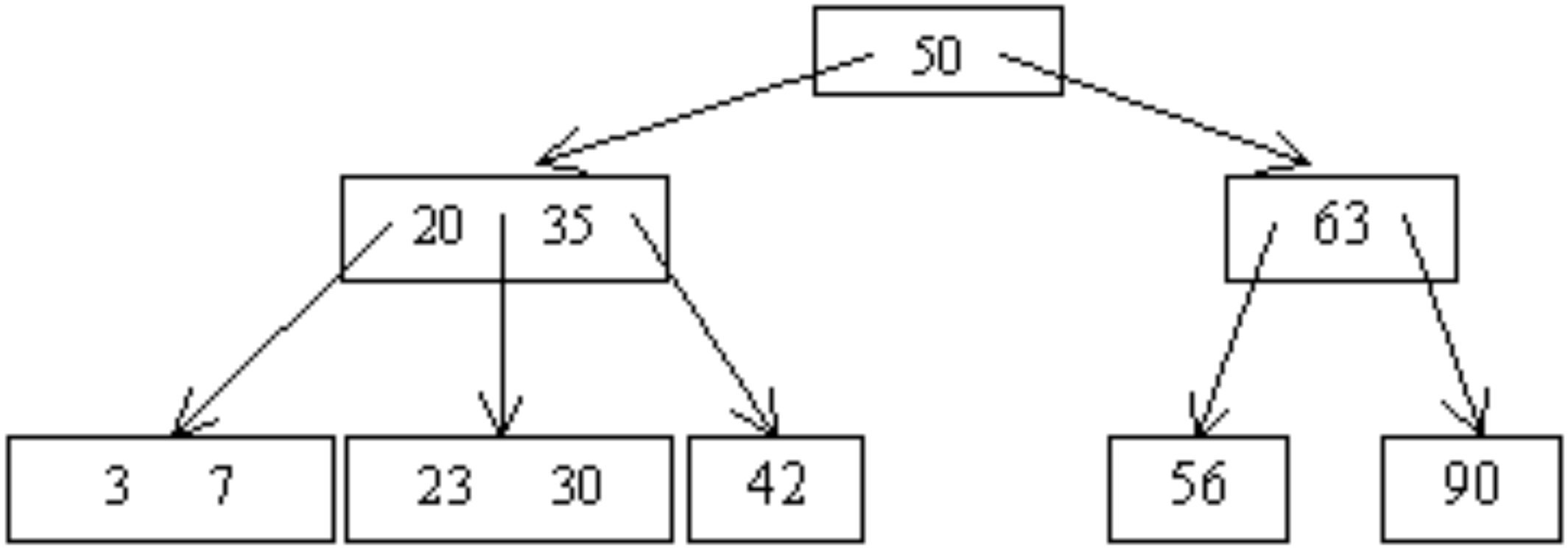
将$k''_{\lceil (z+y)/2+1 \rceil}$上移到父结点中，将左兄弟结点中大于$k''_{\lceil (z+y)/2+1 \rceil}$及父结点中的$k_i$移到被删除关键码的结点中。

删除前的3阶B树



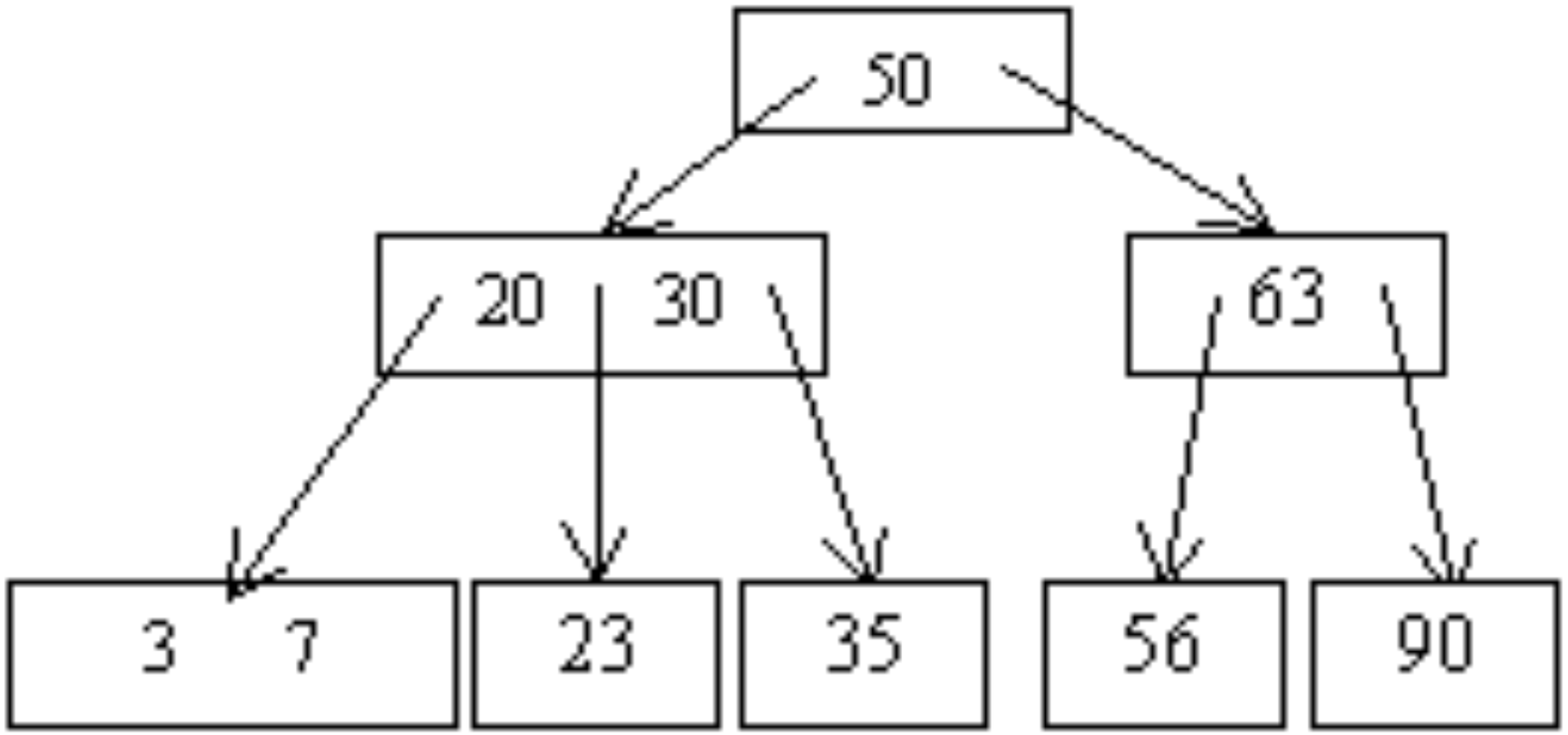
删除**66**

删除66后的B树



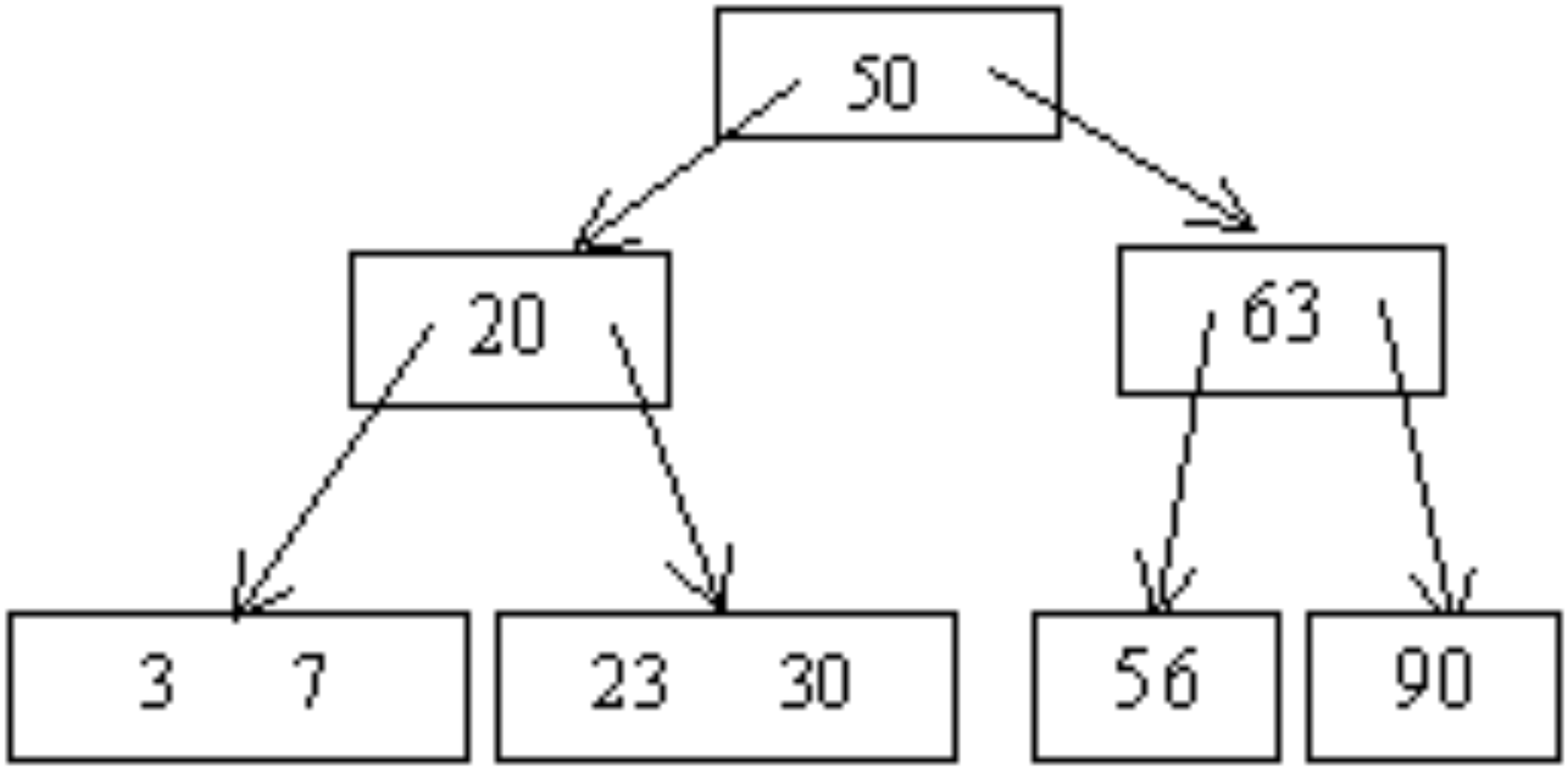
删除**42**

删除42后的B树



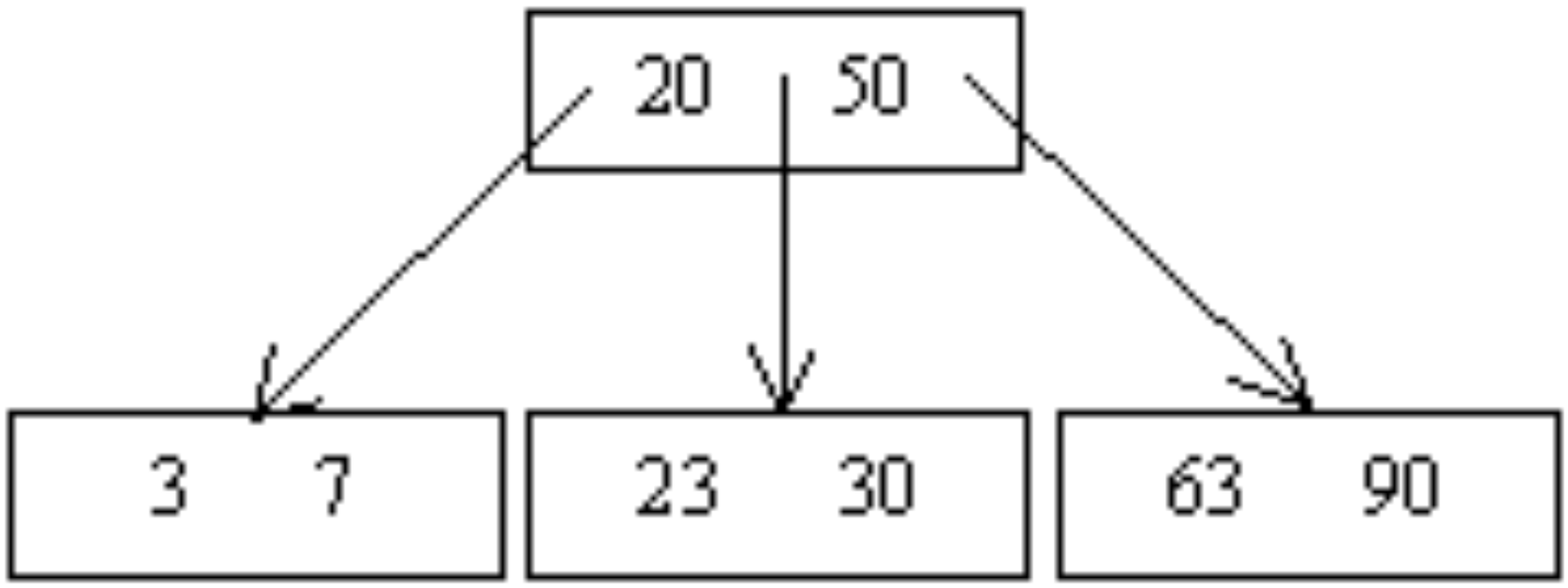
删除**35**

删除35后的B树



删除**56**

删除56后的B树



B+树的定义

- 一棵m阶的B+树满足下列条件：
  - 每个结点至多有m棵子树。
  - 除根结点外，其它每个分支结点至少有$\lceil m/2 \rceil$棵子树。
  - 非叶结点的根结点至少有两棵子树。
  - 所有叶结点在同一层上。
  - 有j个子树的结点有j个关键码，关键码按递增次序排列。

B+树和B树的差异

（1）B+树有n棵子树的结点中含有n个关键码；而B树有n棵子树的结点中含有n-1个关键码。

（2）B+树所有的叶子结点中包含了完整的索引的信息，而B树中非叶结点的关键码与叶结点的关键码均不重复，它们共同构成全部索引信息。

（3）B+树所有的非叶结点可以看成是高层索引，结点中仅含有其子树中最大（或最小）关键码.

说明

- 每个叶结点中至少包含$\lceil m/2 \rceil$个关键码。所有主文件记录的索引项都存放在B+树的叶结点中。
- 所有分支结点可看成是索引的索引。结点中仅包含它的各个子结点中最大(或最小)关键码的分界值及指向子结点的链接。

B+树的运算

- 在B＋树中检索关键码*key*的方法与B树的检索方式相似，但若在分支结点上找到检索的关键码时，检索并不结束，要继续找到B＋树的叶结点为止。
- 插入与B树的插入操作相似，总是插在叶结点上。当结点中原关键码个数等于m时，该结点分裂成两个结点，分别使关键码个数为$\lceil (m+1)/2 \rceil$和$\lceil (m+1)/2 \rceil$。
- 删除时仅在叶结点删除关键码。若因删除操作使结点中关键码数少于$\lceil m/2 \rceil$时，则从兄弟结点调整或者合并。合并的过程和B树相似，区别是父结点中作为分界的关键码，不放入合并后的结点中。

B+树的主要优点

- **B+树可以比B树更快地实现检索：同样大的字典，用B+树组织索引的层数比B树的索引层数少。**
- **由于B+树所有的叶子结点中包含了完整的索引的信息，可以比较方便的表示文件的稀疏索引。**
- **B+树的检索，插入和删除统一在叶结点上进行，比B树相对简单。**

本讲小结

- 多分树是一种树形、静态、多级索引结构。多分树的叶结点就是主文件的最底层索引。主文件的每个页块可以看成是多分树的外部结点。
- B树是一种树形、多级、动态索引结构，B+树可以看作是B树的改进。对于相同大小的结点空间，B+树可以设计的阶数比B树大。因此，B+树可以比B树更快地实现检索。