数据结构

深圳技术大学
大数据与互联网学院

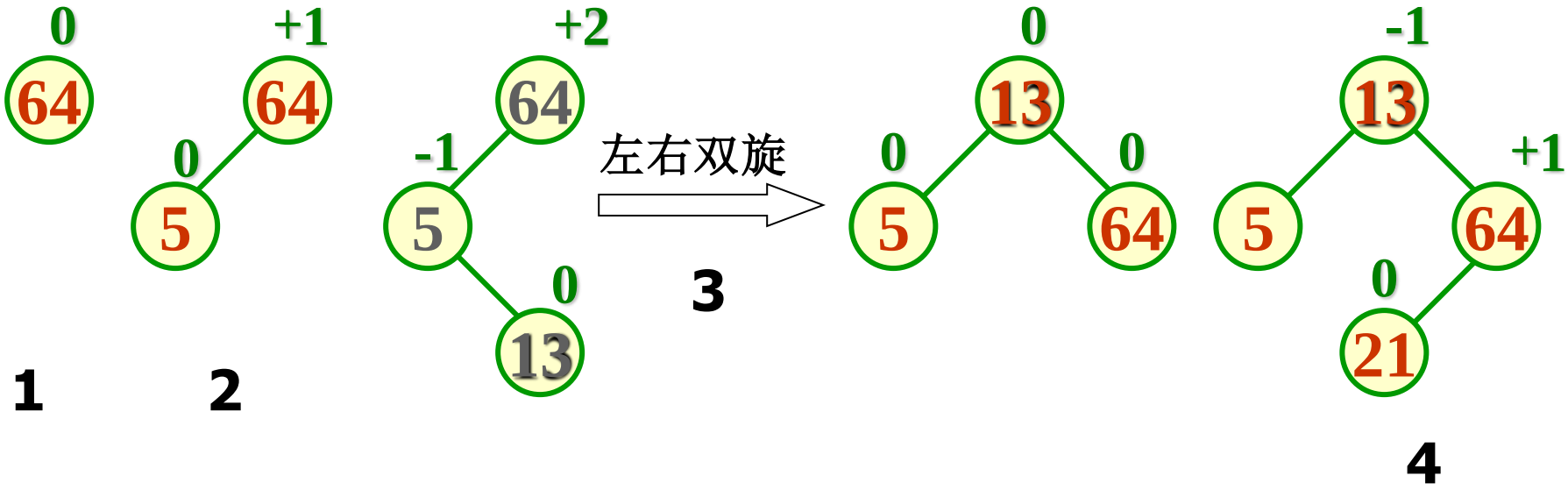
第九章 查找

9.1 静态查找表
9.2 动态查找表
9.3 哈希表

9.2 动态查找表

二. 平衡二叉树

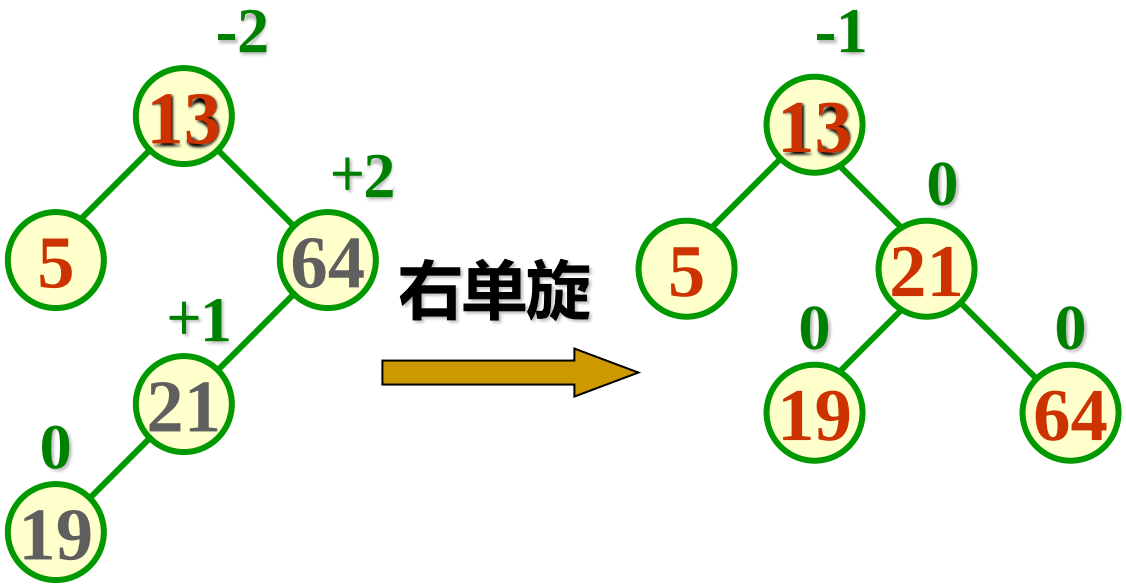
- 综合举例
  - 画出在初始为空的AVL树中依次插入 64, 5, 13, 21, 19, 80, 75, 37, 56的生长过程，并在有旋转时说出旋转的类型。



9.2 动态查找表

二. 平衡二叉树

- 举例
  - 在完成*64, 5, 13, 21*后，继续插入19, 80, 75, 37, 56时该树的生长过程，

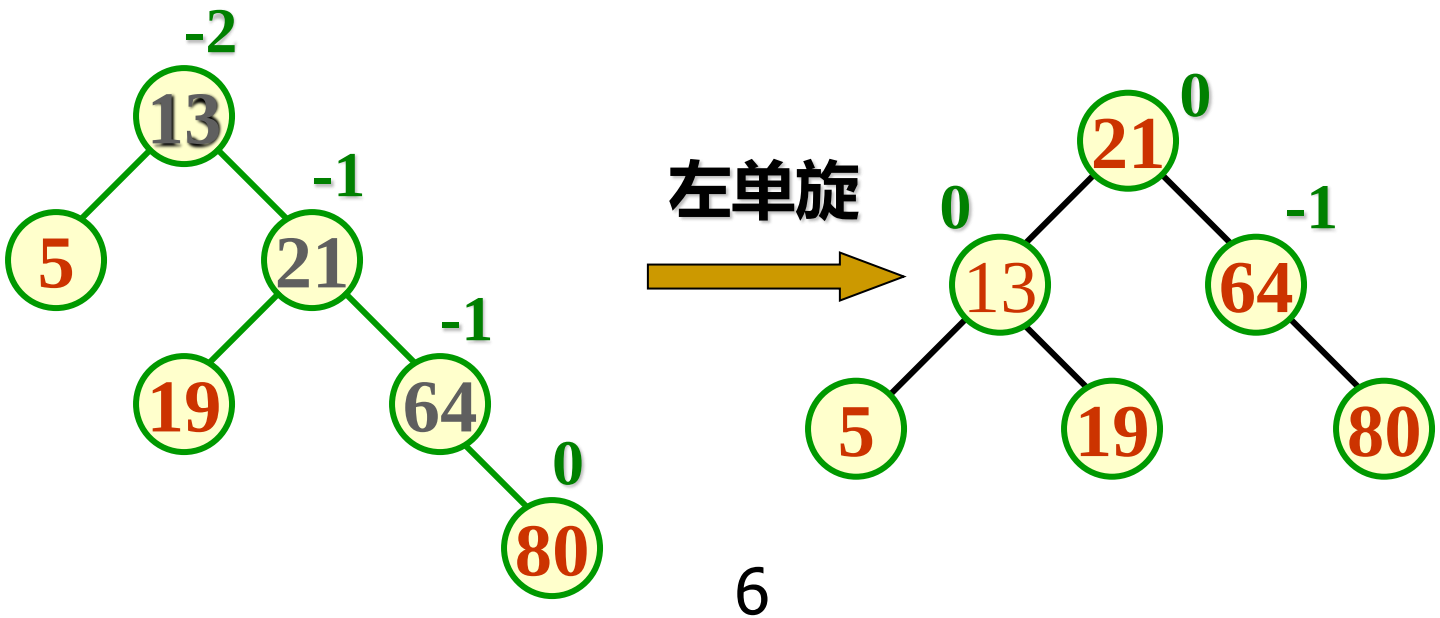


5

9.2 动态查找表

二. 平衡二叉树

- 举例
  - 在完成*64, 5, 13, 21, 19*后，继续插入80, 75, 37, 56时该树的生长过程，

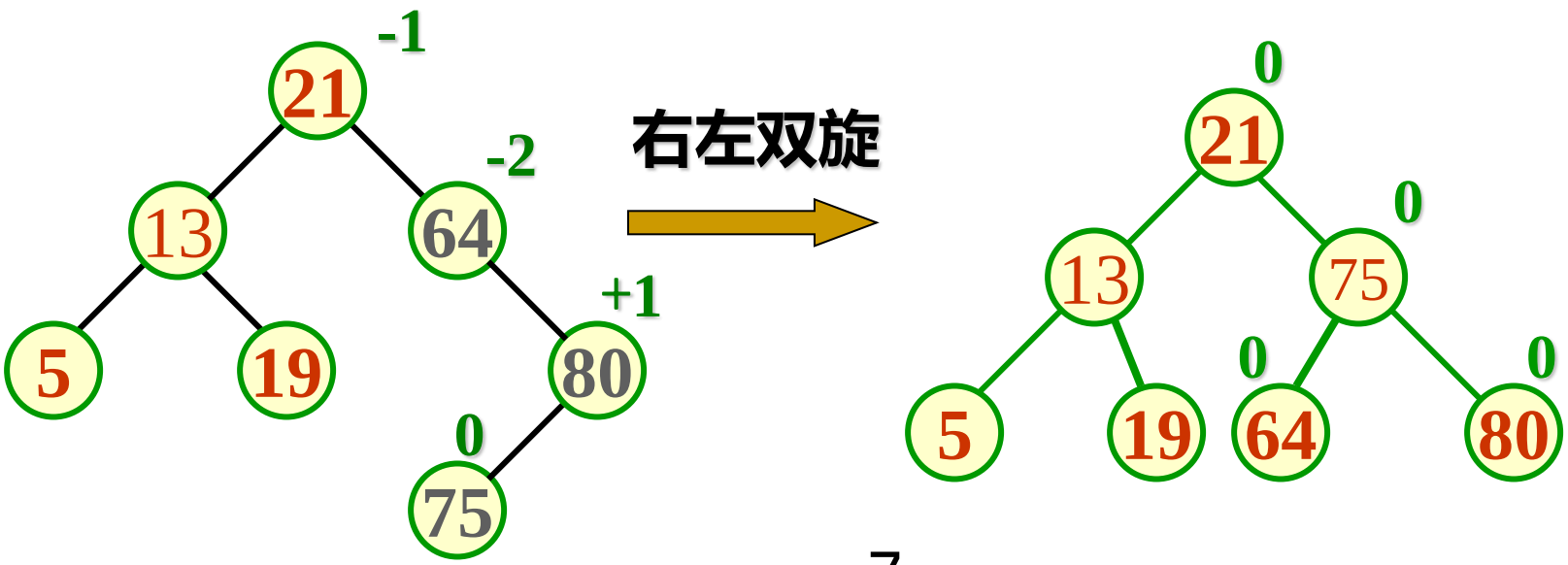


6

9.2 动态查找表

二. 平衡二叉树

- 举例
  - 在完成*64, 5, 13, 21, 19, 80*后，继续插入75, 37, 56时该树的生长过程，

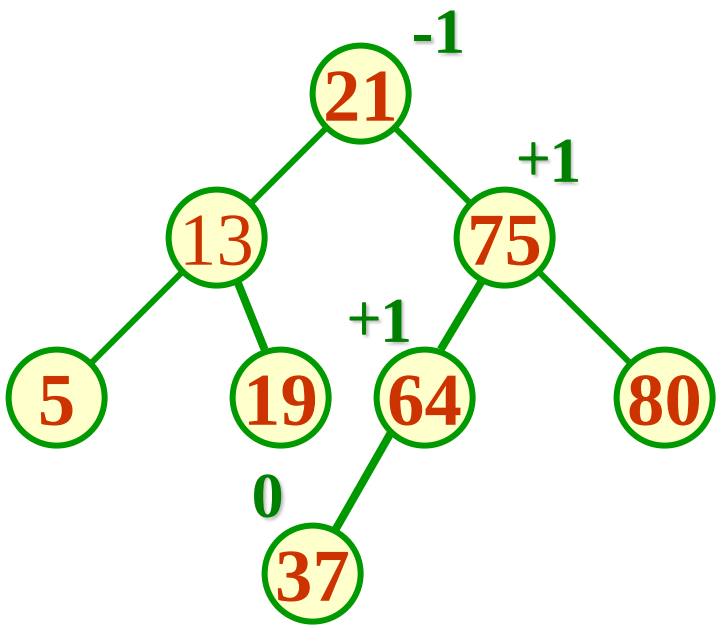


7

9.2 动态查找表

二. 平衡二叉树

- 举例
  - 在完成*64, 5, 13, 21, 19, 80, 70*后，继续插入37, 56时该树的生长过程，

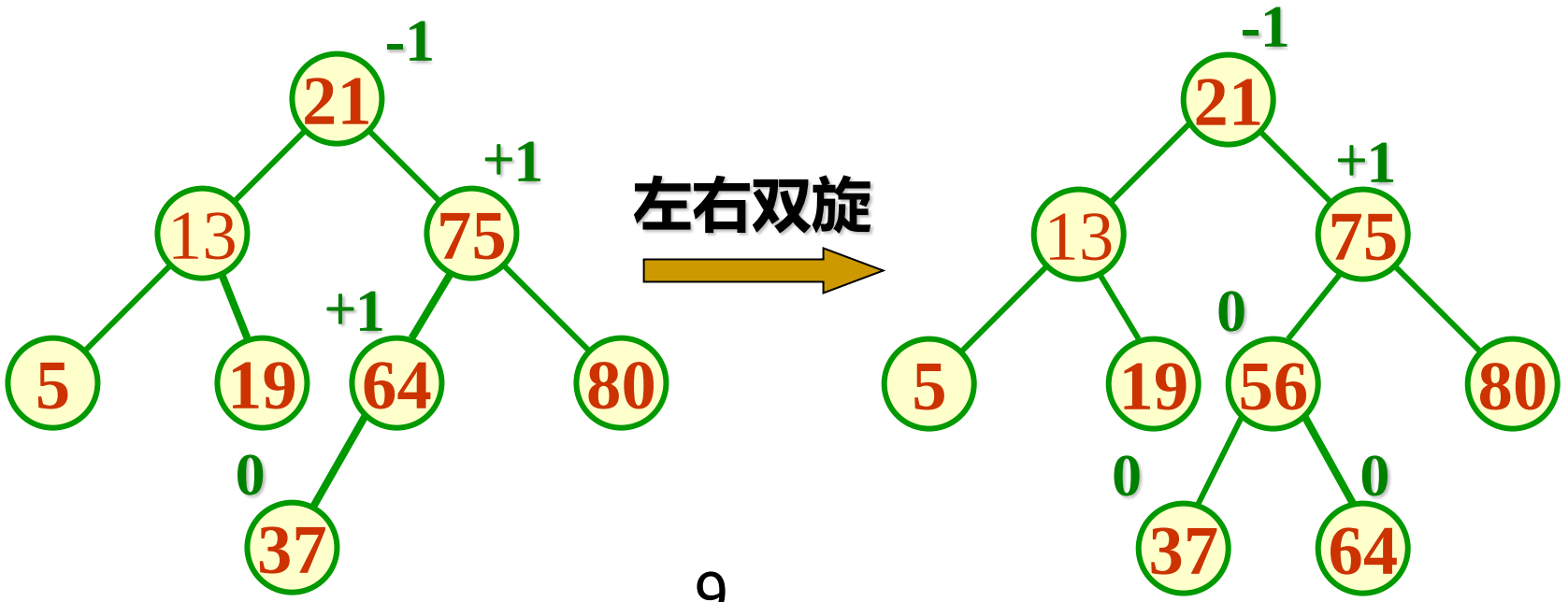


8

9.2 动态查找表

二. 平衡二叉树

- 举例
  - 在完成*64, 5, 13, 21, 19, 80, 70, 37*后，继续插入56时该树的生长过程，

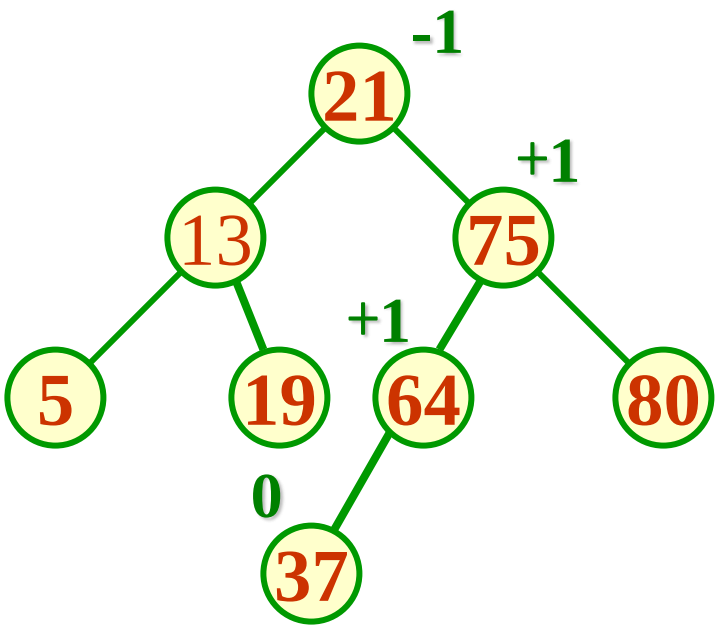


8 9

9.2 动态查找表

二. 平衡二叉树

- 平衡二叉树删除举例
  - 独立删除5、64、37，无需平衡化处理
  - 独立删除80，单向右旋处理
  - 连续删除5和13，以21-75-64做右左双旋



9.2 动态查找表

三. B-树和B+树

- 平衡二叉排序树便于动态查找，因此用平衡二叉排序树来组织索引表是一种可行的选择。
- 当用于大型数据库时，所有数据及索引都存储在外存，因此，涉及到内、外存之间频繁的数据交换，这种交换速度的快慢成为制约动态查找的瓶颈。
- 若以二叉树的结点作为内、外存之间数据交换单位，则查找给定关键字时对磁盘平均进行$\log_2 n$次访问是不能容忍的，因此，必须选择一种能尽可能降低磁盘I/O次数的索引组织方式，树结点的大小尽可能地接近页的大小。
- **R.Bayer和E.Mc Creight在1972年提出了一种多路平衡查找树，称为B-树(其变型体是B+树)**

9.2 动态查找表

三. B-树和B+树

- **B-树是一种多路平衡查找树，一棵度为m的B-树称为m阶B-树，其定义是：**
  - 一棵m阶B-树，或者是空树，或者是满足以下性质的m叉树：

(1) 根结点或者是叶子，或者至少有两棵子树，至多有m棵子树；

(2) 除根结点外，所有非终端结点至少有$\lceil m/2 \rceil$棵子树，至多有m棵子树

(3) 所有叶子结点都在树的同一层上；

(4) 每个结点应包含如下信息：

$$(n, A_0, K_1, A_1, K_2, A_2, \cdots, K_n, A_n)$$

(5) 所有叶子结点都出现在同一层次，且不带任何信息，可以把叶子看作外部结点或查找失败结点。实际上叶子不存在，指向叶子的指针也为空

9.2 动态查找表

三. B-树和B+树

- **B-树的每个结点应包含如下信息：**

  $(n, A_0, K_1, A_1, K_2, A_2, \ldots, K_n, A_n)$

  - **$K_i(1 \le i \le n)$是关键字，且$K_i < K_{i+1}$ $(1 \le i \le n-1)$；**
  - **$A_i(i=0, 1, \ldots, n)$为指向孩子结点的指针，**
  - **A_{i-1}所指向的子树中所有结点的关键字都小于K_i，A_i所指向的子树中所有结点的关键字都大于K_i；**
  - **n是结点中关键字的个数，且$\lfloor m/2 \rfloor - 1 \le n \le m-1$**

9.2 动态查找表

三. B-树和B+树

- **B-树举例**

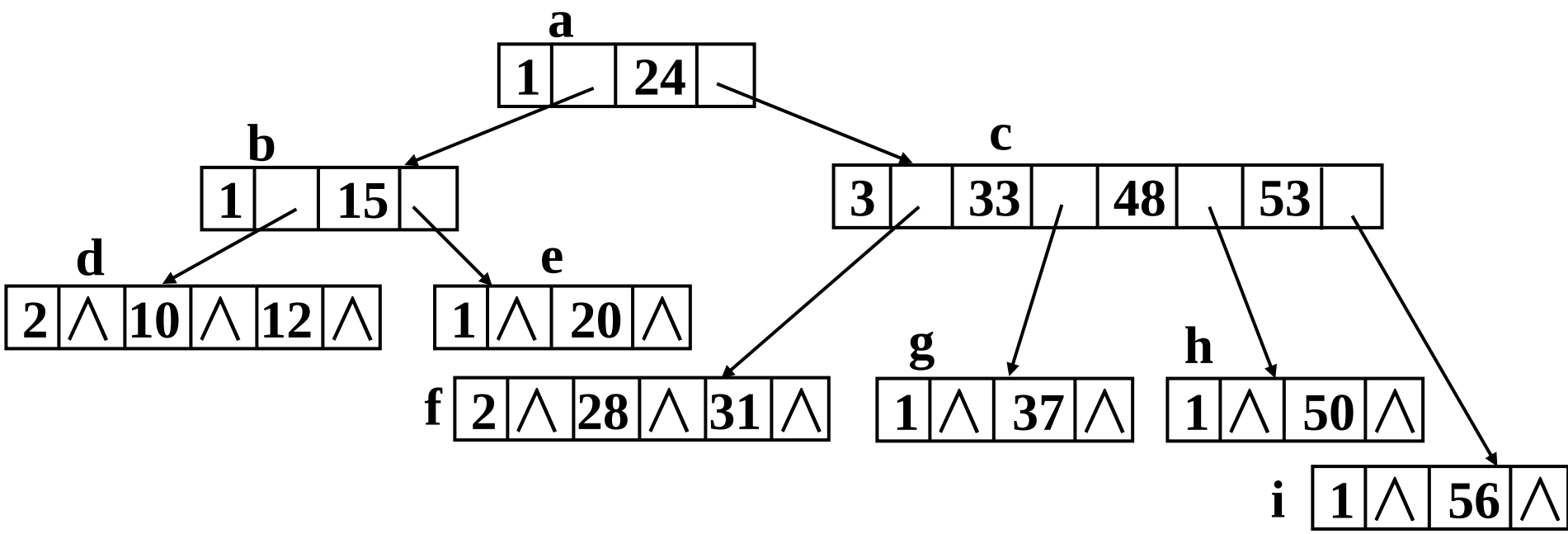


图　一棵包含13个关键字的4阶B-树

9.2 动态查找表

三. B-树和B+树

- **B-树的查找，查找过程和二叉排序树的查找相似**

① 从树的根结点T开始，在T所指向的结点的关键字向量 **key[1…keynum]**中查找给定值**K**(用折半查找)：

若**key[i]=K(1≤i≤keynum)**，则查找成功，返回结点及关键字位置；否则，转 ② ；

② 将**K**与向量**key[1…keynum]**中的各个分量的值进行比较，以选定查找的子树：

- 若**K<key[1]**：**T=T->ptr[0]**；
- 若**key[i]<K<key[i+1](i=1, 2, …keynum-1)**：

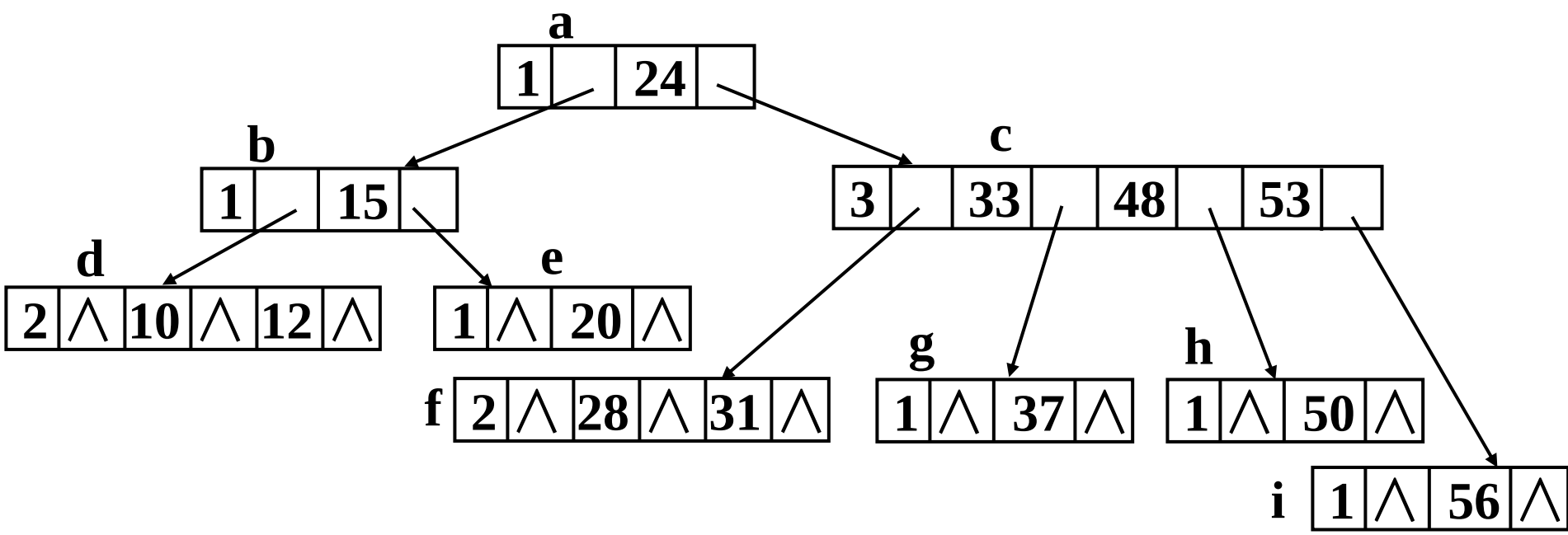
  T=T->ptr[i]；
- 若**K>key[keynum]**：**T=T->ptr[keynum]**；

转①，直到T是叶子结点且未找到相等的关键字，则查找失败。

9.2 动态查找表

三. B-树和B+树

- **B-树的查找举例**
  - 查找37：a-c-g
  - 查找22：a-b-e



9.2 动态查找表

三. B-树和B+树

- **B-树查找的代码**

```
int Search(BTree p, KeyType K) {
   for(int i=0;  i < p->keynum && p->key[i+1] <= K;  i++);
   return i;
}

Result SearchBTree(BTree T, KeyType K, int f) {  // 算法9.13
   // 在m阶B树T上查找关键字K，返回结果(pt,i,tag)
   BTree p, q;
   int found, i, j=0;
   Result R;
   p = T;  q = NULL;  found = FALSE;  i = 0;
                  // 初始化，p指向待查结点，q指向p的双亲
   while (p && !found) {
      i = Search(p, K);  // 在p->key[1..keynum]中查找i,
                         // 使得：p->key[i]<=K<p->key[i+1]
      if (i>0 && p->key[i]==K) found = TRUE;    // 找到待查关键字
      else { q = p;  p = p->ptr[i]; }
      j++;
   }
   if (found) {  // 查找成功
     R.pt = p;  R.i = i;  R.tag = 1;
   } else {      // 查找不成功
     R.pt = q;  R.i = i;  R.tag = 0;
   }
   return R;                  // 返回结果信息：K的位置(或插入位置)
} // SearchBTree
```

9.2 动态查找表

三. B-树和B+树

- **B-树在磁盘中的应用，文件29查找举例**

(1) 根据根结点指针找到文件目录的根磁盘块1，将其中的信息导入内存（磁盘操作1次）

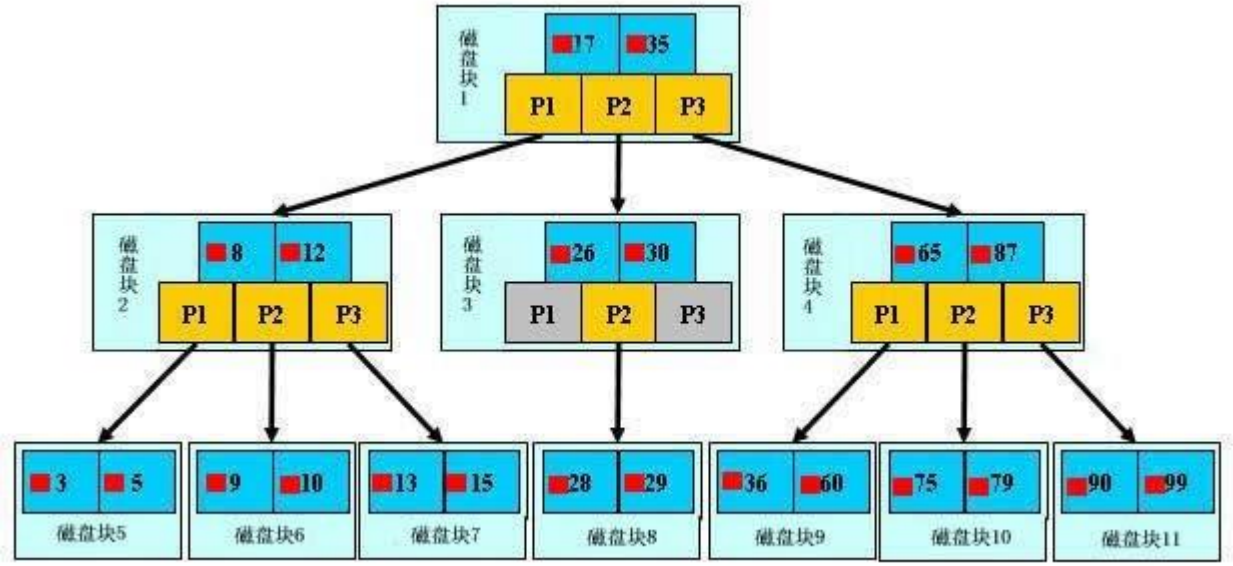
(2) 此时内存中有两个文件名17，35和三个存储其他磁盘页面地址的数据。根据算法我们发现17<29<35，因此我们找到指针p2。

(3) 根据p2指针，我们定位到磁盘块3，并将其中的信息导入内存（磁盘IO操作2次）

(4) 此时内存中有两个文件名26，30和三个存储其他磁盘页面地址的数据。根据算法我们发现26<29<30，因此我们找到指针p2。

(5) 根据p2指针，我们定位到磁盘块8，并将其中的信息导入内存（磁盘IO操作3次）

(6) 此时内存中有两个文件名28，29。根据算法我们查找到文件29，并定位了该文件内存的磁盘地址。



9.2 动态查找表

三. B-树和B+树

- **B-树的查找性能举例**
  - $h \leqq 1+\log_s((n+1)/2)=1+\log_{\lceil m/2 \rceil}((n+1)/2)((n+1)/2)$
  - 即在含有n个关键字的B-树上进行查找时，从根结点到待查找记录关键字的结点的路径上所涉及的结点数不超过

    $$1+\log_{\lceil m/2 \rceil}((n+1)/2)$$

  - 即查找性能取决于树的度m和树的结点数n

9.2 动态查找表

三. B-树和B+树

- **B-树的插入**
  - B-树的生成也是从空树起，逐个插入关键字。
  - 插入时不是每插入一个关键字就添加一个叶子结点，而是首先在最低层某个叶子结点添加一个关键字，然后可能“分裂”
- **B-树的插入算法**

① 在B-树的中查找关键字K，若找到，表明关键字已存在，返回；否则，K的查找操作失败于某个叶子结点，转 ②；

② 将K插入到该叶子结点中，插入时，若：

- 叶子结点的关键字数<m-1：直接插入；
- 叶子结点的关键字数=m-1：将结点“分裂”

9.2 动态查找表

三. B−树和B+树

- **B-树的分裂方法**
  - 设待分裂结点包含信息为：

$(m, A_0, K_1, A_1, K_2, A_2, \ldots, K_m, A_m)$

  - 从其中间位置分为两个结点和中间关键字：

$(\lceil m/2 \rceil - 1, A_0, K_1, A_1, \ldots, K_{\lceil m/2 \rceil - 1}, A_{\lceil m/2 \rceil - 1})$

$(m - \lceil m/2 \rceil, A_{\lceil m/2 \rceil}, K_{\lceil m/2 \rceil + 1}, A_{\lceil m/2 \rceil + 1}, \ldots, K_m, A_m)$

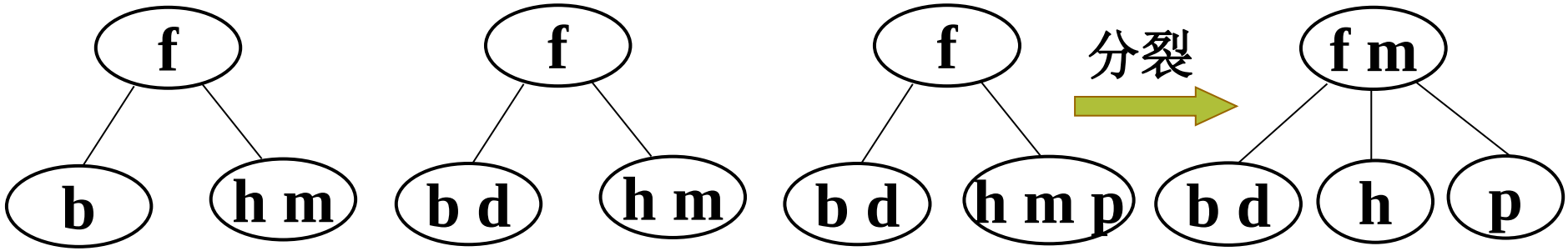
$K_{\lceil m/2 \rceil - 1}$

  - 并将中间关键字$K_{\lceil m/2 \rceil}$插入到p的父结点中，以分裂后的两个结点作为中间关键字$K_{\lceil m/2 \rceil}$的两个子结点
  - 当将中间关键字$K_{\lceil m/2 \rceil}$插入到p的父结点后，父结点也可能不满足m阶B-树的要求(分枝数大于m)，则必须对父结点进行“分裂”，一直进行下去，直到没有父结点或分裂后的父结点满足m阶要求
  - 当根结点分裂时因没有父结点，则建立一个新的根，B-树增高一层

9.2 动态查找表

三. B−树和B+树

- B-树的分裂举例

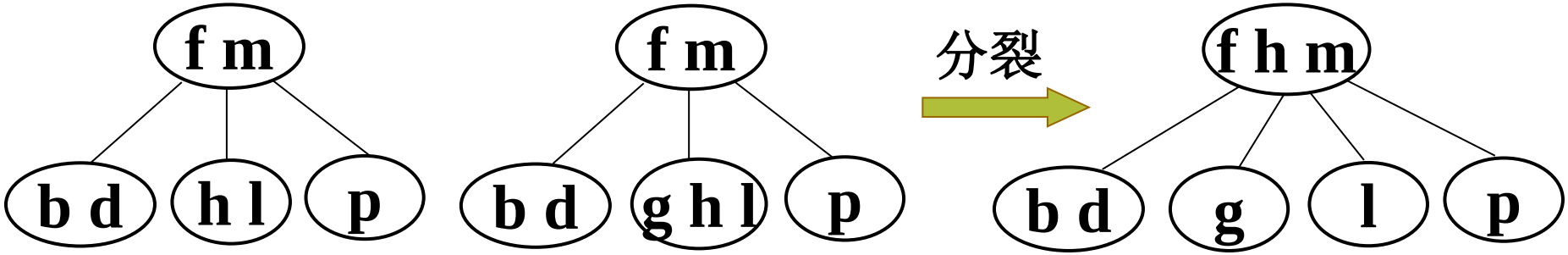


(a) 一棵2-3树　(b) 插入d后　(c) 插入p后并进行分裂

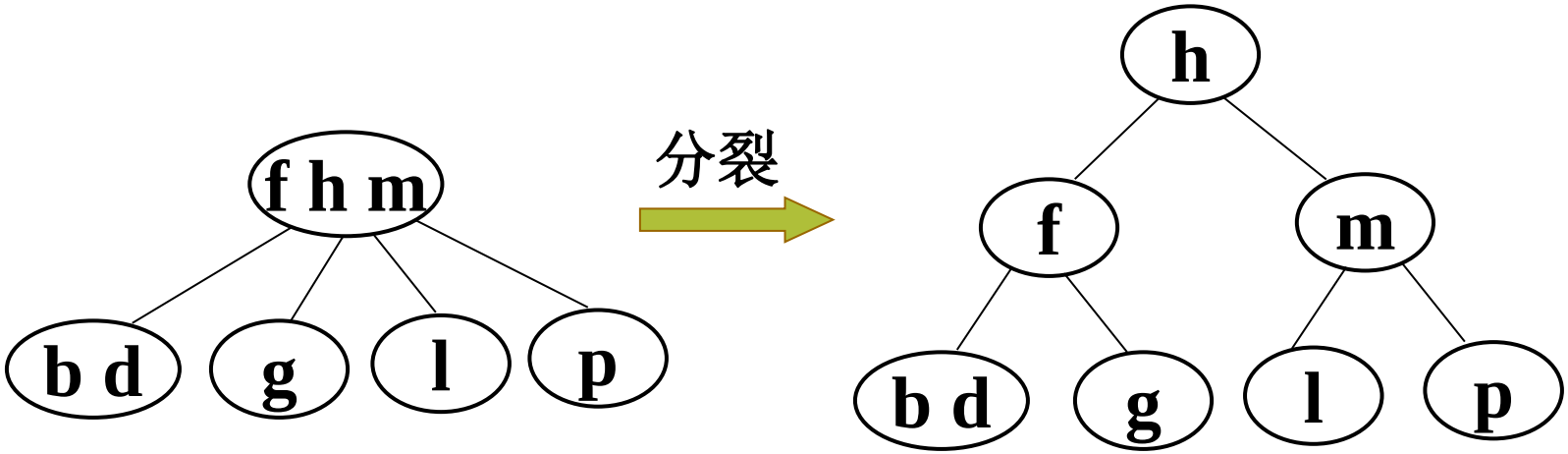
9.2 动态查找表

三. B-树和B+树

- B-树的分裂举例



(d) 插入l后　　(e) 插入g后并进行分裂



(f) 继续进行分裂

9.2 动态查找表

三. B-树和B+树

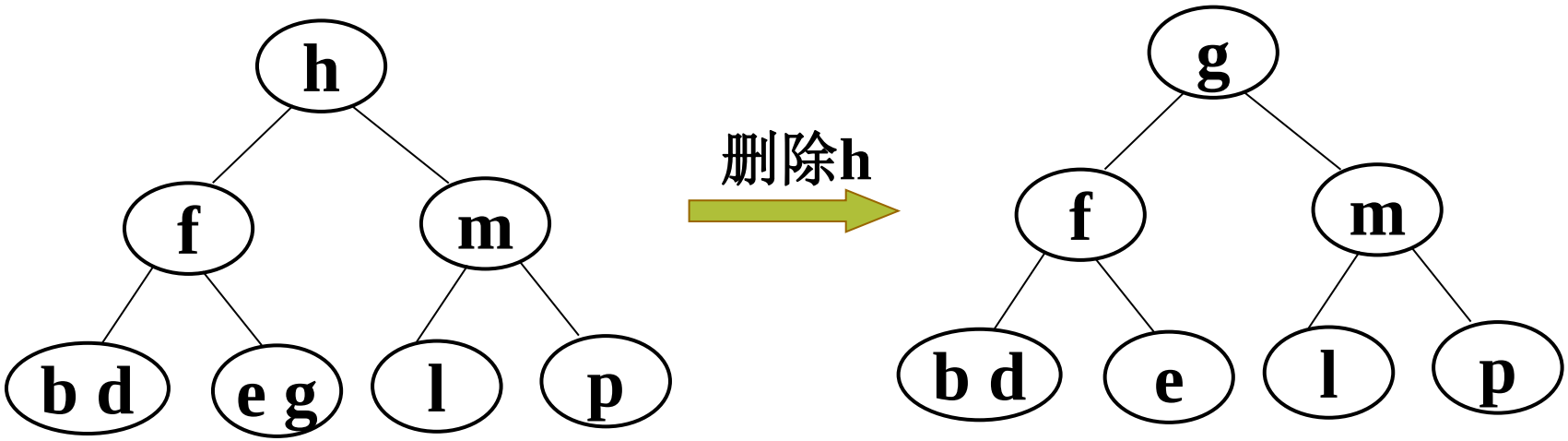
B-树的删除

- 首先找到关键字所在的结点N，然后在N中进行关键字的删除操作
- 若N不是叶子结点，设K是N中的第i个关键字，则将指针A_{i-1}或A_{i-1}所指子树中的最大关键字(或最小关键字)K'放在(K)的位置，然后删除K'，而K'一定在叶子结点上。
- 接着处理在叶子结点上删除关键字的情况

9.2 动态查找表

三. B-树和B+树

- **B-树的删除——结点不是叶子的情况**
  - 在子树中找出最接近关键字进行替换

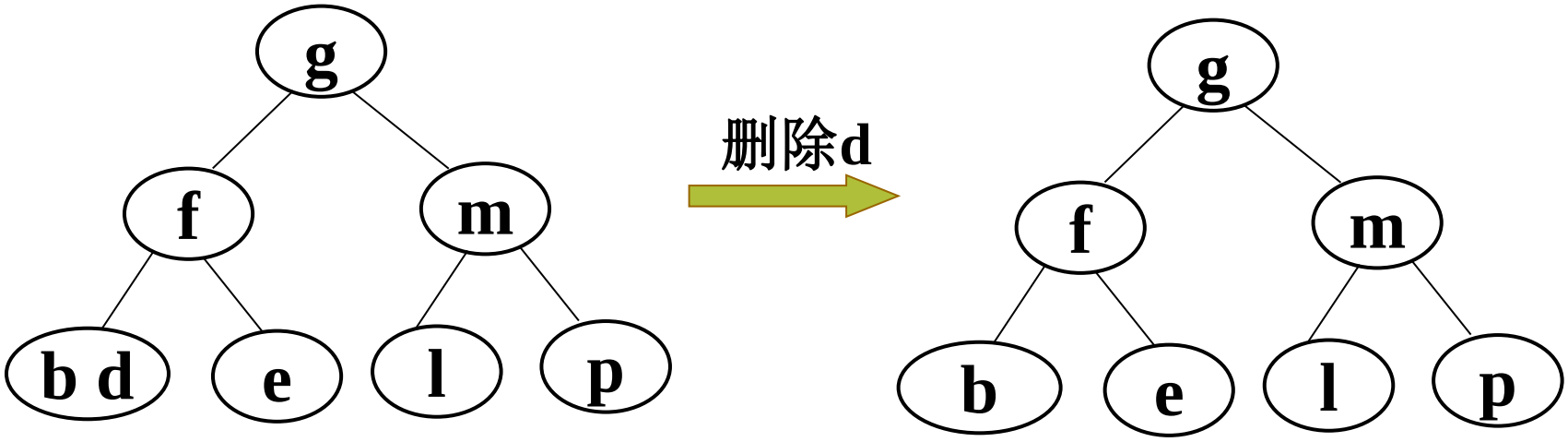


9.2 动态查找表

三. B-树和B+树

- **B-树的删除**
  - 在叶子结点上删除关键字的三种情况

(1) 若结点N中的关键字个数>⌈m/2⌉-1：在结点中直接删除关键字K



9.2 动态查找表

三. B-树和B+树

B-树的删除

- 在叶子结点上删除关键字的情况

⑵ 若结点N中的关键字个数=⌈m/2⌉-1，且若结点N的左(右)兄弟结点中的关键字个数>⌈m/2⌉-1

则将结点N的左(或右)兄弟结点中的最大(或最小)关键字上移到其父结点中，而父结点中大于(或小于)且紧靠上移关键字的关键字下移到结点N

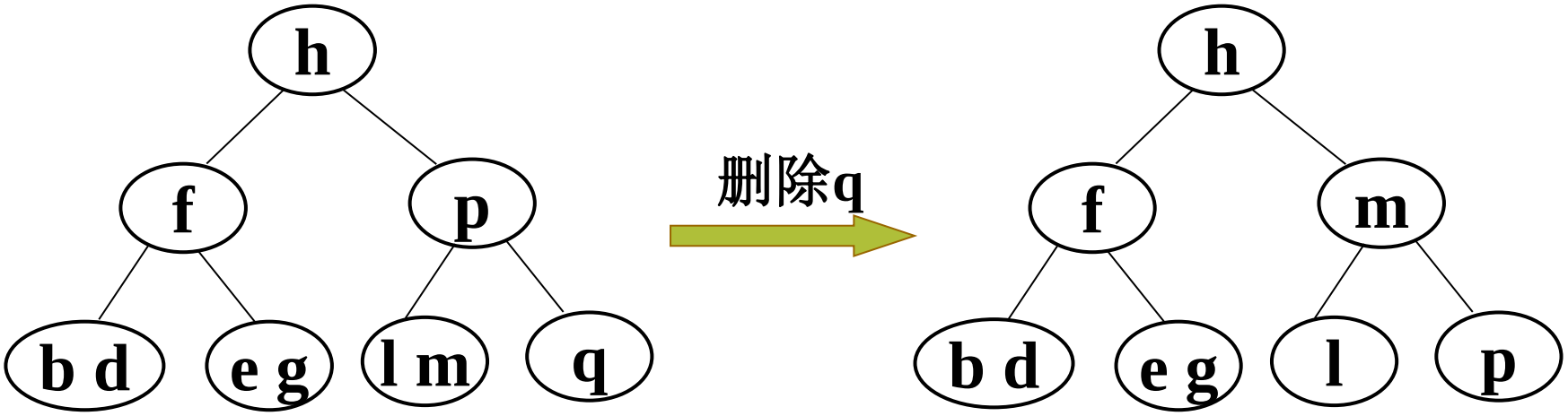
9.2 动态查找表

三. B-树和B+树

- **B-树的删除**
  - 在叶子结点上删除关键字的情况

⑵ 若结点N中的关键字个数=⌈m/2⌉-1，且若结点N的左(右)兄弟结点中的关键字个数>⌈m/2⌉-1

将兄弟中最接近删除关键字的数值放入父亲，将父亲最接近删除关键字的数值放入删除结点中



9.2 动态查找表

三. B−树和B+树

B-树的删除

- 在叶子结点上删除关键字的情况

(3) 若结点N和其兄弟结点中的关键字数=$\lceil m/2 \rceil$-1

删除结点N中的关键字，再将结点N中的关键字、指针与其兄弟结点以及分割二者的父结点中的某个关键字K_i，合并为一个结点，若因此使父结点中的关键字个数<$\lceil m/2 \rceil$-1，则依此类推

9.2 动态查找表

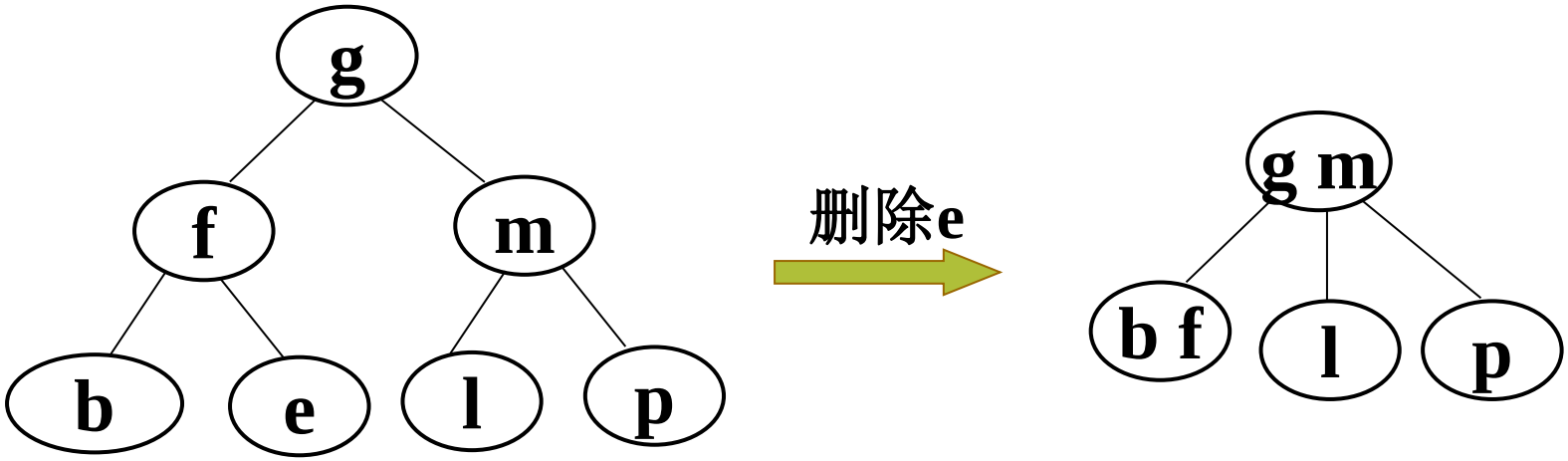
三. B-树和B+树

- **B-树的删除**
  - 在叶子结点上删除关键字的情况

(3) 若结点N和其兄弟结点中的关键字数=⌈m/2⌉-1。例如，删除e

先把f合并到b中，导致原有包含f的结点变空

然后把变空的结点和父亲g和兄弟m合并成新结点



9.2 动态查找表

三. B-树和B+树

- **B+树的概念**
  - 在实际的文件系统中，基本上不使用B-树，而是使用B-树的一种变体，称为m阶B^+树。
  - B^+树 与B-树的主要不同是叶子结点中存储记录。
  - 在B^+树中，所有的非叶子结点可以看成是索引，而其中的关键字是作为“分界关键字”，用来界定某一关键字的记录所在的子树。

9.2 动态查找表

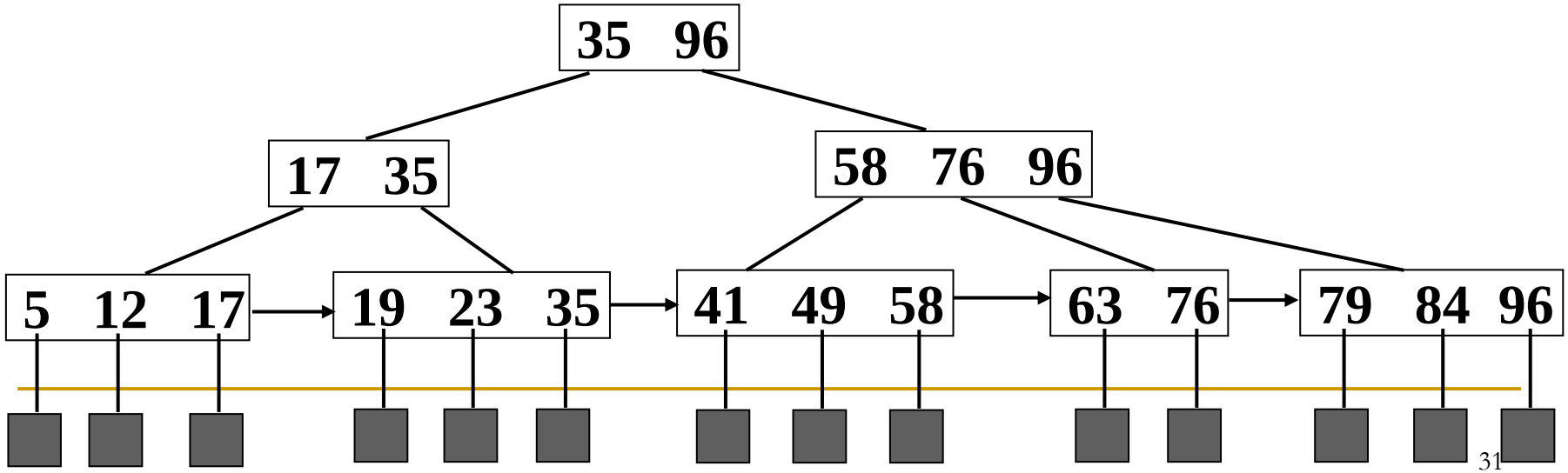
三. B-树和B+树

- **B+树与B-树的主要差异是：**

(1) 若一个结点有n棵子树，则必含有n个关键字；

(2) 所有叶子结点中包含了全部记录的关键字信息以及这些关键字记录的指针，而且叶子结点按关键字的大小从小到大顺序链接；在叶子结点上删除关键字的情况

(3) 所有的非叶子结点可以看成是索引的部分，结点中只含有其子树的根结点中的最大(或最小)关键字。



9.2 动态查找表

三. B-树和B+树

B+树的查找

- 与B-树相比，对B+树不仅可以从根结点开始按关键字随机查找，而且可以从最小关键字起，按叶子结点的链接顺序进行顺序查找。在B+树上进行随机查找、插入、删除的过程基本上和B-树类似。
- 在B+树上进行随机查找时，若非叶子结点的关键字等于给定的K值，并不终止，而是继续向下直到叶子结点(只有叶子结点才存储记录)， 即无论查找成功与否，都走了一条从根结点到叶子结点的路径。

9.2 动态查找表

三. B-树和B+树

B+树的插入

- B+树的插入仅仅在叶子结点上进行。当叶子结点中的关键字个数大于m时，“分裂”为两个结点，两个结点中所含有的关键字个数分别是$\lfloor (m+1)/2 \rfloor$和$\lceil (m+1)/2 \rceil$，且将这两个结点中的最大关键字提升到父结点中，用来替代原结点在父结点中所对应的关键字。提升后父结点又可能会分裂，依次类推。