9.2.2 B-树

1. B-树的概念

B-树是一种平衡的多路查找树。 一棵m阶的B-树，或为空树，或为满足下列特性的m叉树：

1）树中每个结点至多有m棵子树；

2）若根结点不是叶子结点，则至少有两棵子树；

3）除根之外的所有非终端结点至少有$\lceil m/2 \rceil$棵子树；

4）所有的非终端结点中包含下列信息数据

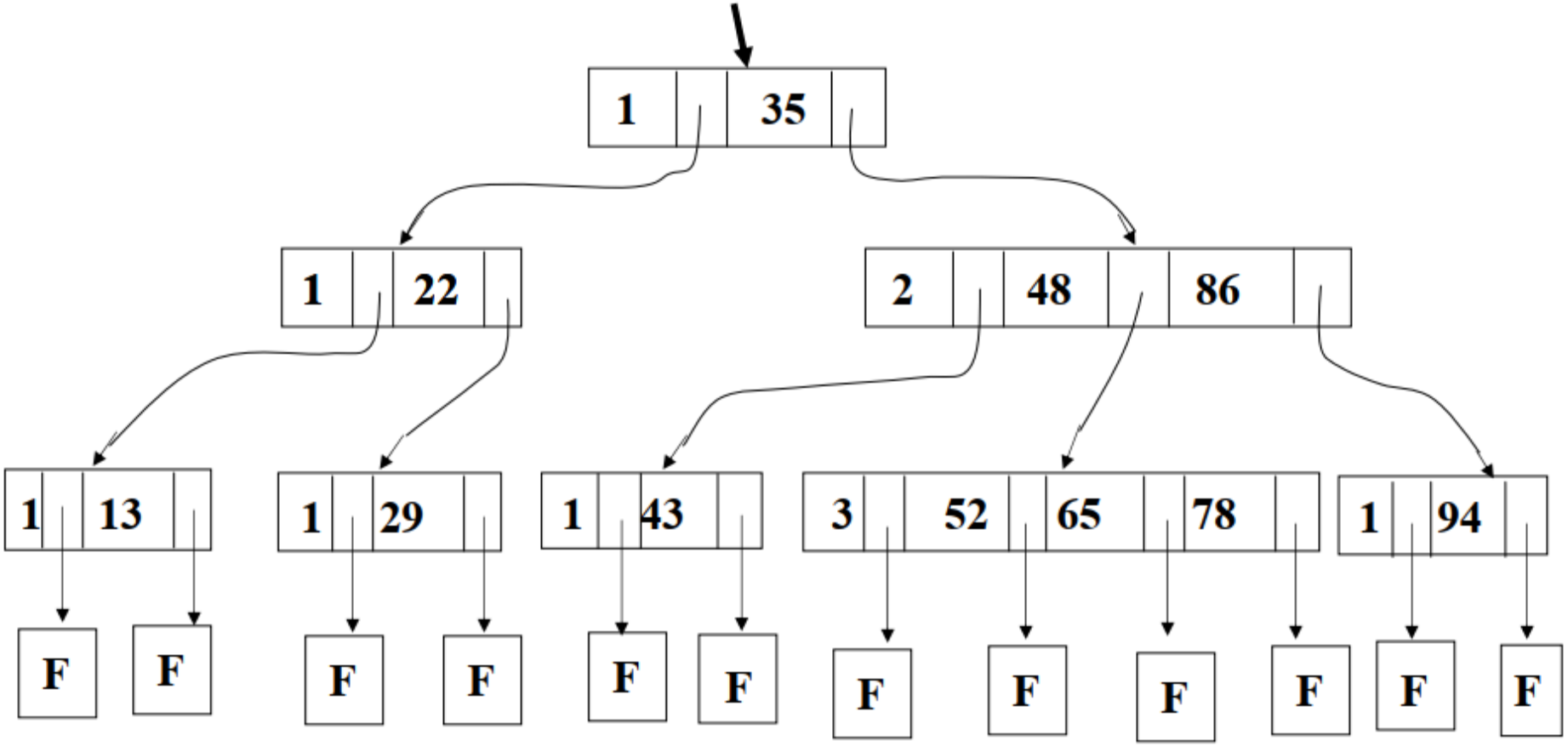
（n，A_0，K_1，A_1，K_2，A_2，…，K_n，A_n）

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

其中： $K_i(i=1, \ldots, n)$为关键字, 且$K_i < K_{i+1}$ ($i=1, \ldots, n-1$)；$A_i(i=0, \ldots, n)$为指向子树根结点的指针，且指针A_{i-1}所指子树中所有结点的关键字均小于K_i（$i=1, \ldots, n$），A_n所指子树中所有结点的关键字均大于 K_n, n（$\lceil m/2 \rceil - 1 \le n \le m-1$）为关键字的个数（或n+1为子树个数）。

5）所有的叶子结点都出现在同一层次上，并且不带信息，仅表示查找失败。

B-树的例子



一棵4阶的B-树

根据m阶B_树的定义，结点的类型定义如下：

```
#define M    5        /*  根据实际需要定义B_树的阶数   */
typedef  struct BTNode
  { int   keynum ;    /*    结点中关键字的个数    */
    struct BTNode  *parent ; /*    指向父结点的指针    */
    KeyType  key[M+1] ;    /* 关键字向量,key[0]未用    */
    struct BTNode  *ptr[M+1] ; /*    子树指针向量    */
    RecType   *recptr[M+1] ;
      /*    记录指针向量,recptr[0]未用    */
  }BTNode ;
```

2　B_树的查找

由B_树的定义可知，在其上的查找过程和二叉排序树的查找相似。

算法思想：

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：

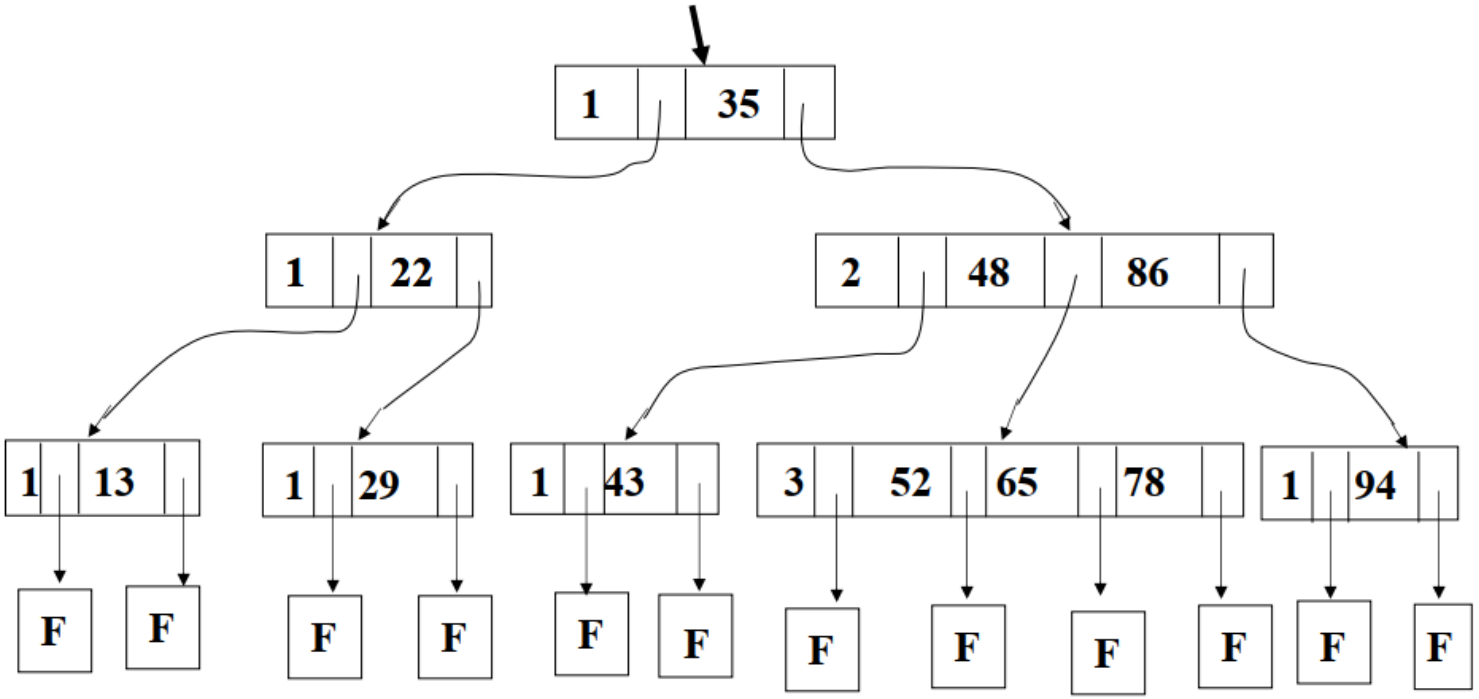
- 若K<key[1]：T=T->ptr[0]；
- 若key[i]<K<key[i+1](i=1, 2, …keynum-1)：
  T=T->ptr[i]；
- 若K>key[keynum]：T=T->ptr[keynum]；

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。

B - 树

从根结点出发，沿指针搜索结点和在结点内进行顺序（或折半）查找两个过程交叉进行。

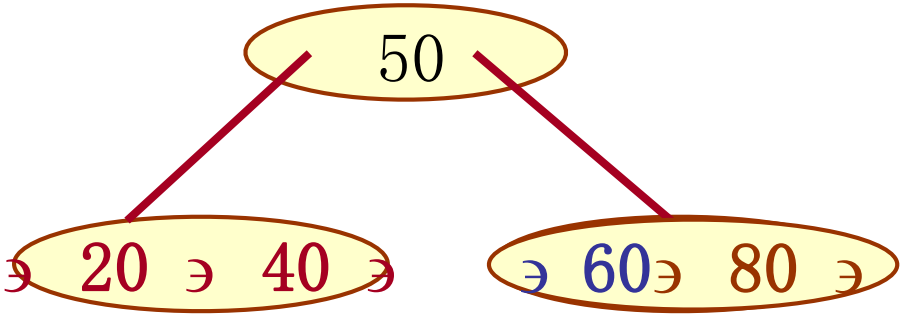
若查找成功，则返回指向被查关键字所在结点的指针和关键字在结点中的位置；



B - 树 插入结点

在查找不成功之后，需进行插入。显然，关键字插入的位置必定在最下层的叶子结点，有下列几种情况（以 3-阶为例）：

1）插入后，该结点的子树个数n<m，不需要修改指针；如插入关键字60



结点的关键字个数不超过m-1

B - 树 插入结点

2）插入后，该结点的子树个数 **n=m**，则需进行“结点分裂”:

令 $s = \lceil m/2 \rceil$

a.在原结点中保留

（A_0，K_1，。。。，K_{s-1}，A_{s-1}）；

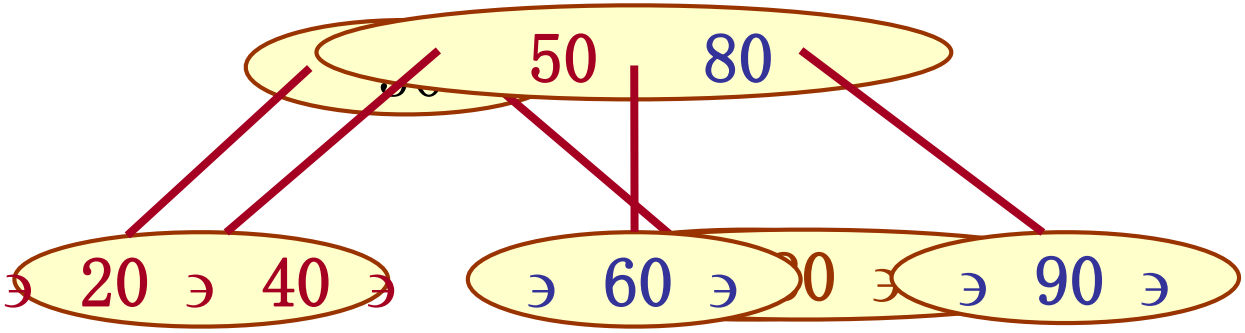
b.建新结点

（A_s，K_{s+1}，。。。，K_n，A_n）；

c.将（K_s，**p**）插入双亲结点

B - 树 插入结点

再插入关键字90



结点分裂”：

令 s = ⌈m/2⌉

a. 在原结点中保留

（A_0，K_1，。。。，K_{s-1}，A_{s-1}）；

b. 建新结点

（A_s，K_{s+1}，。。。，K_n，A_n）；

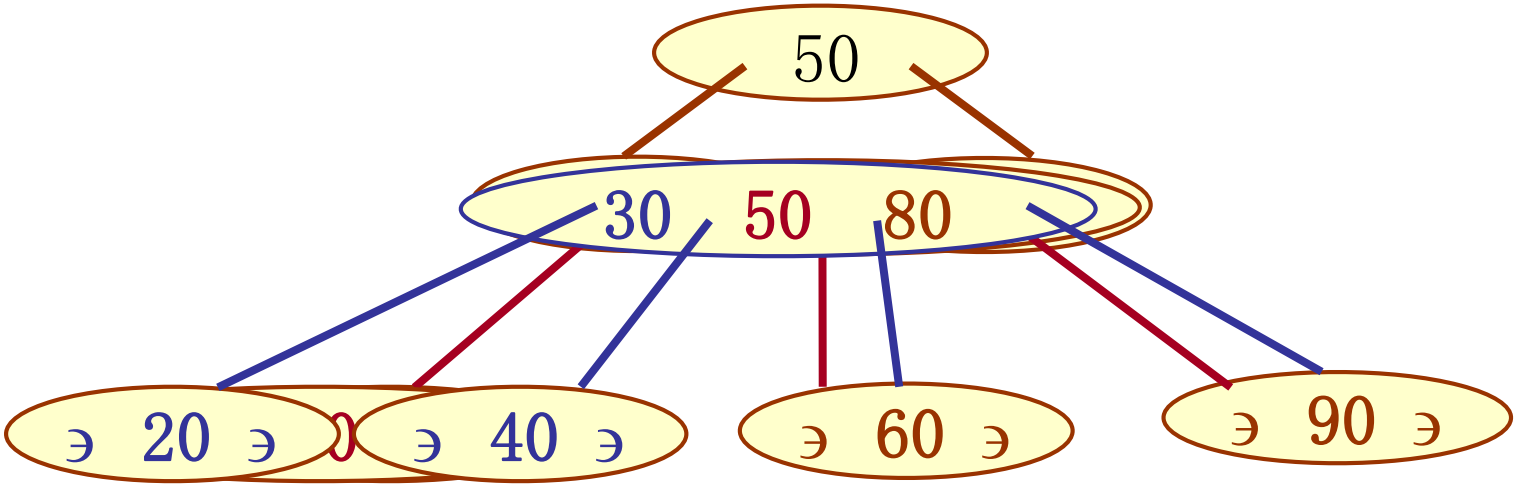
c. 将（K_s，p）插入双亲结点

结点的关键字个数不超过m-1

B - 树 插入结点

3）若双亲为空，则建新的根结点。

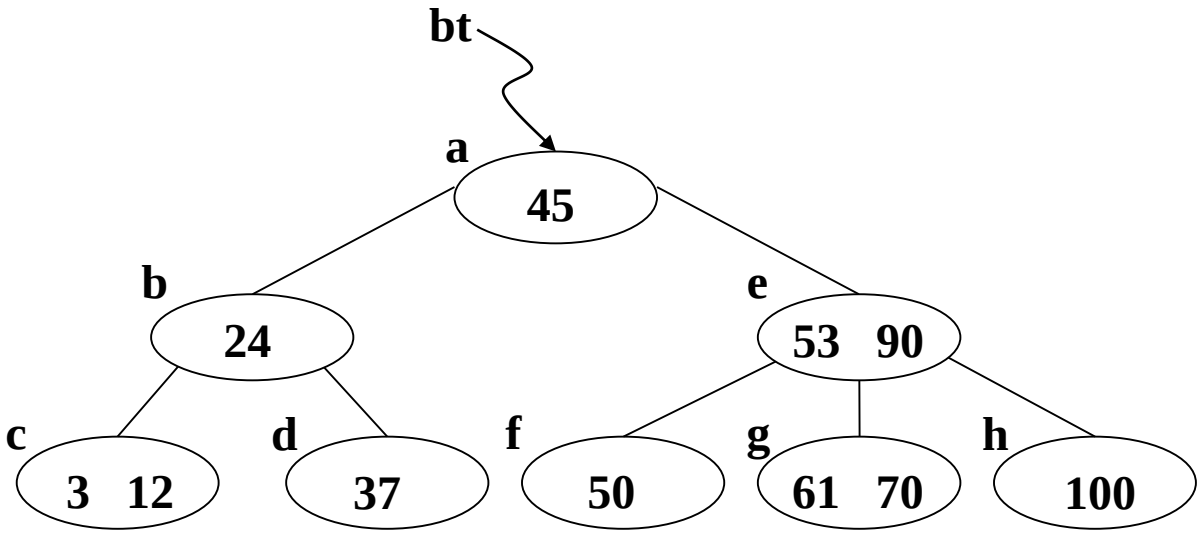
再插入关键字30



结点的关键字个数不超过m-1

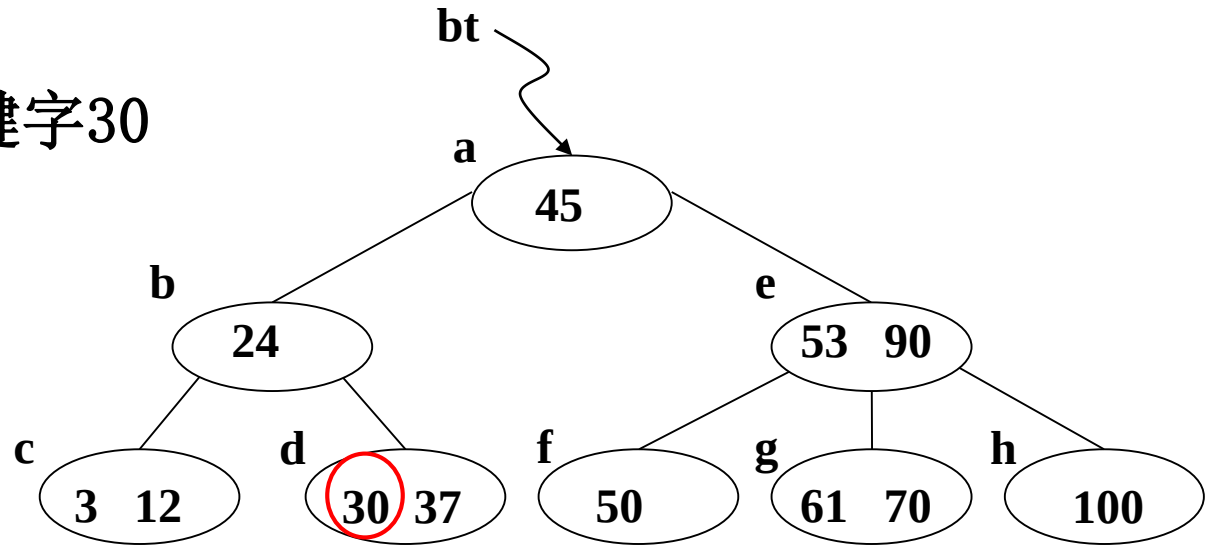
例子

如图所示为3阶B-树（图中略去F结点，即叶子结点），假设需依次插入关键字30，26，85和7。



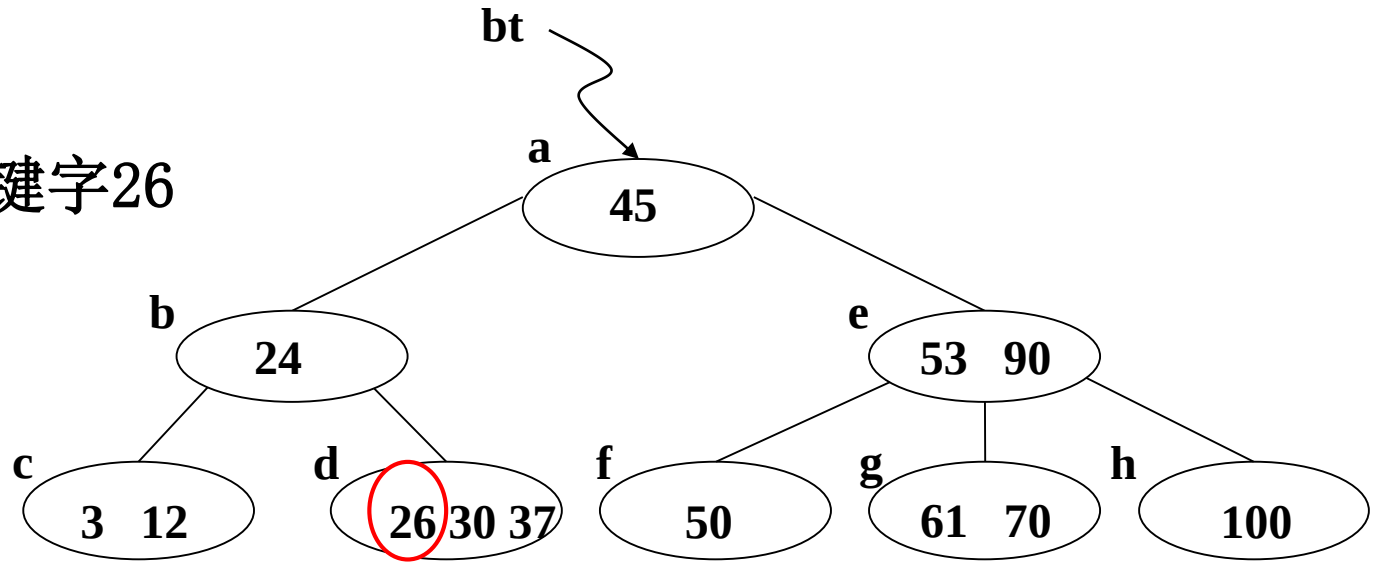
(a) 一棵2-3树

插入关键字30

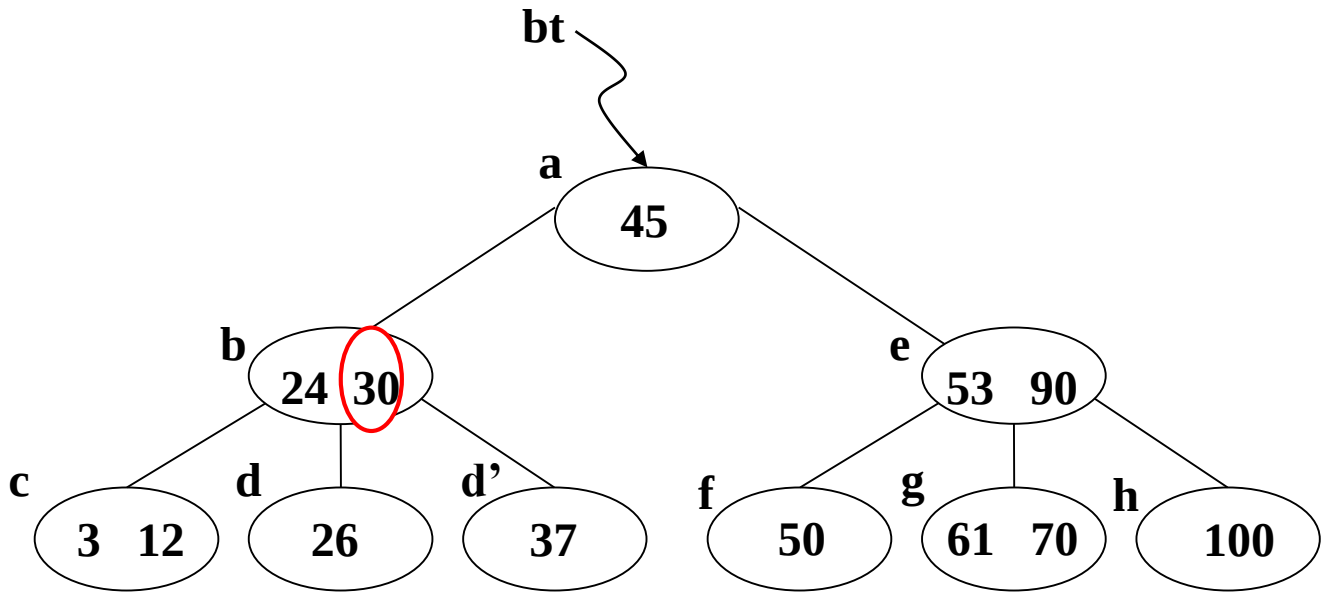


(b)

插入关键字26

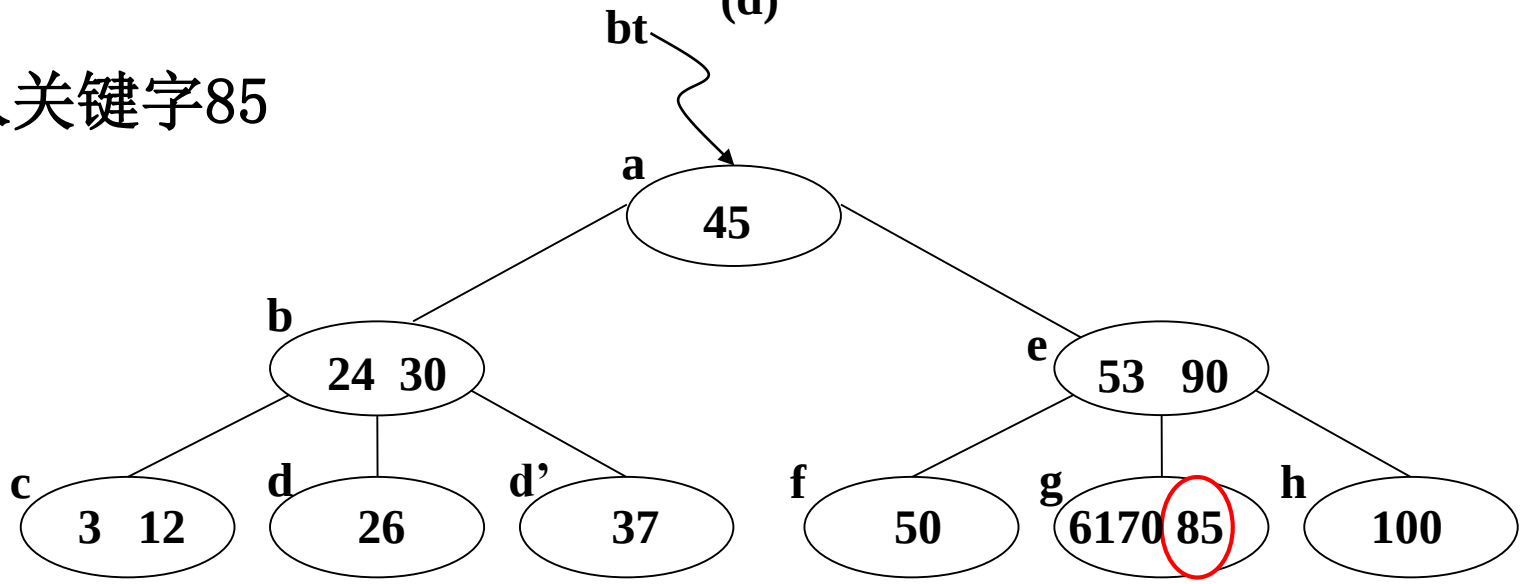


(c)

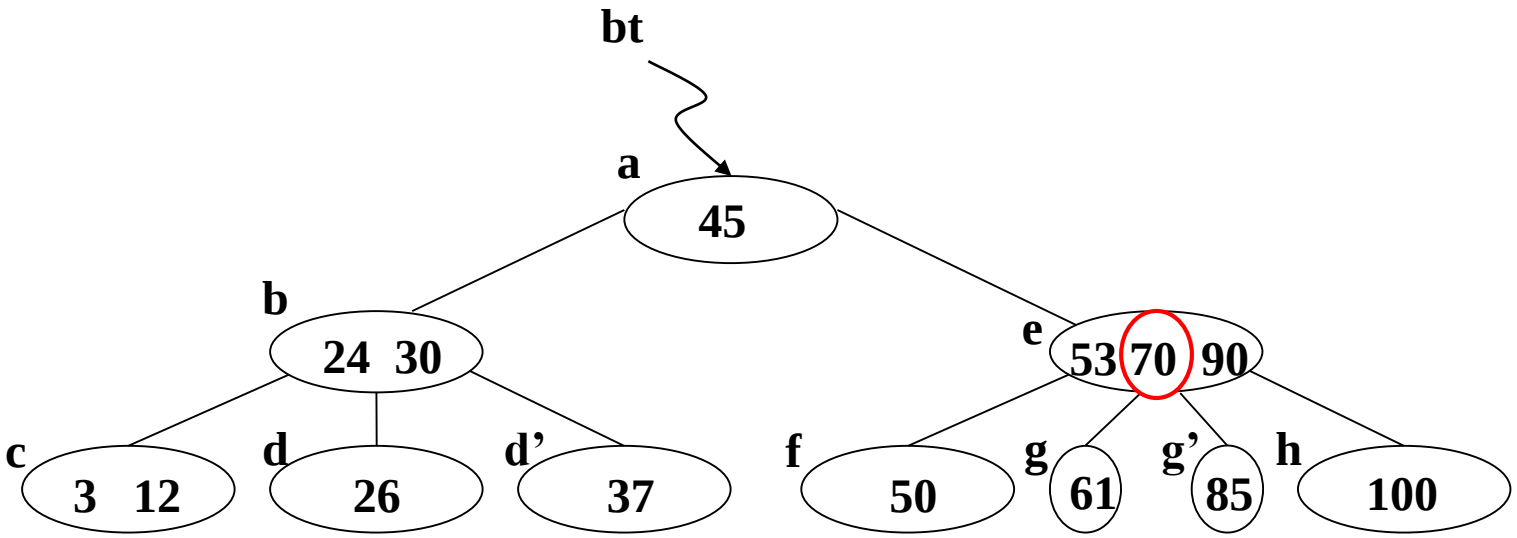

bt
a
45
b
24 30
e
53 90
c
3 12
d
26
d'
37
f
50
g
61 70
h
100

(d)

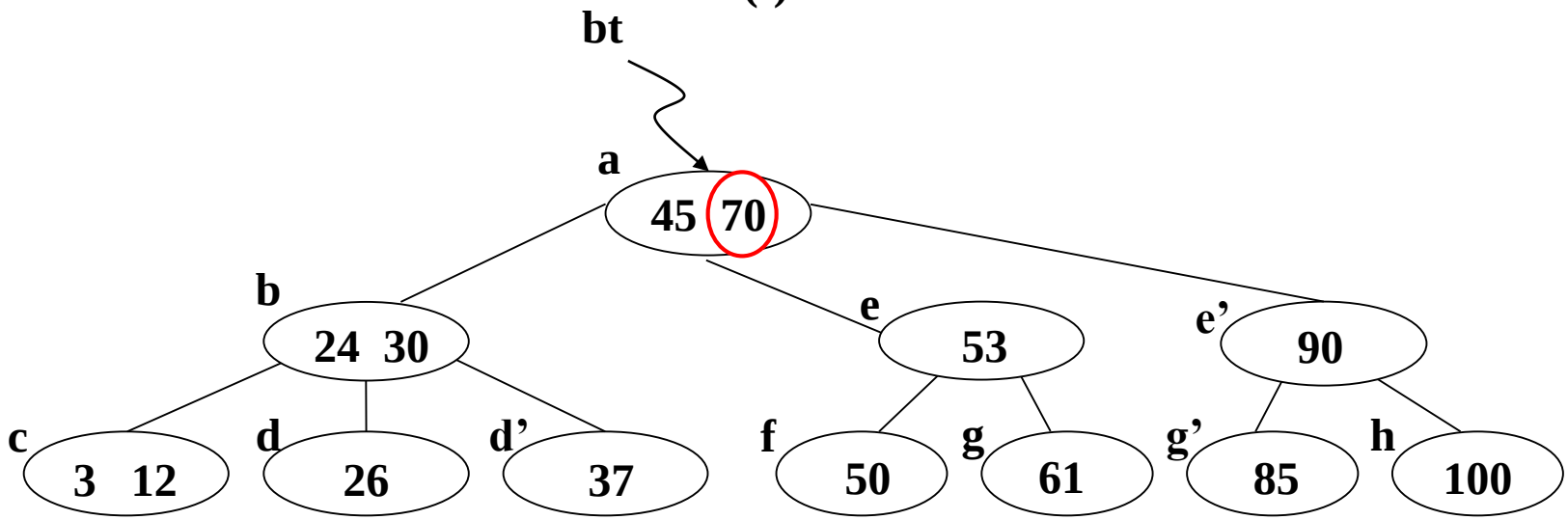
插入关键字85


bt
a
45
b
24 30
e
53 90
c
3 12
d
26
d'
37
f
50
g
6170 85
h
100

(e)


bt
a
45
b
24 30
e
53 70 90
c
3 12
d
26
d'
37
f
50
g
61
g'
85
h
100

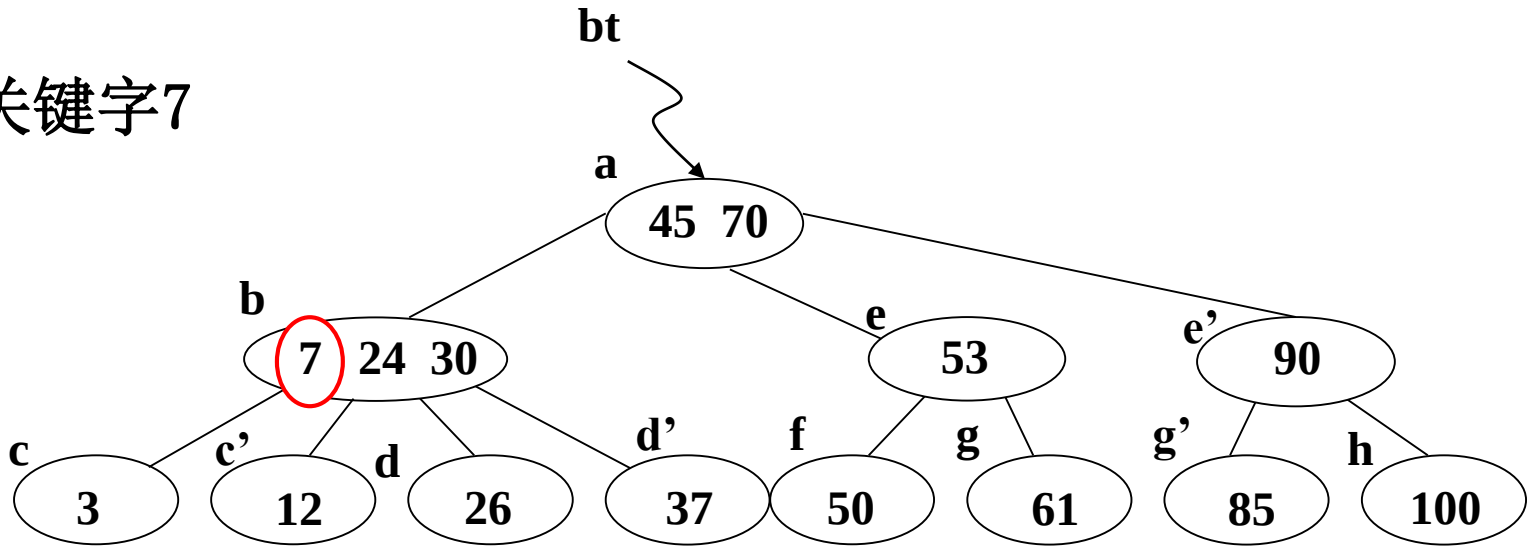
(f)


bt
a
45 70
b
24 30
e
53
e'
90
c
3 12
d
26
d'
37
f
50
g
61
g'
85
h
100

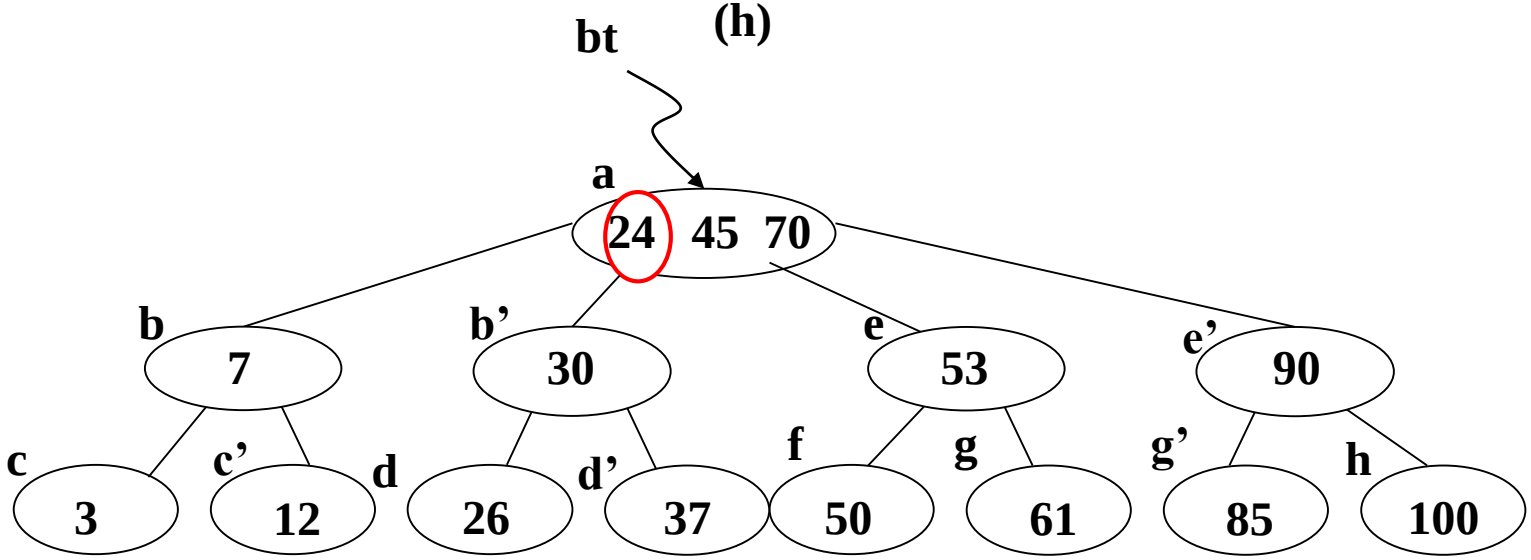
(g)

插入关键字7

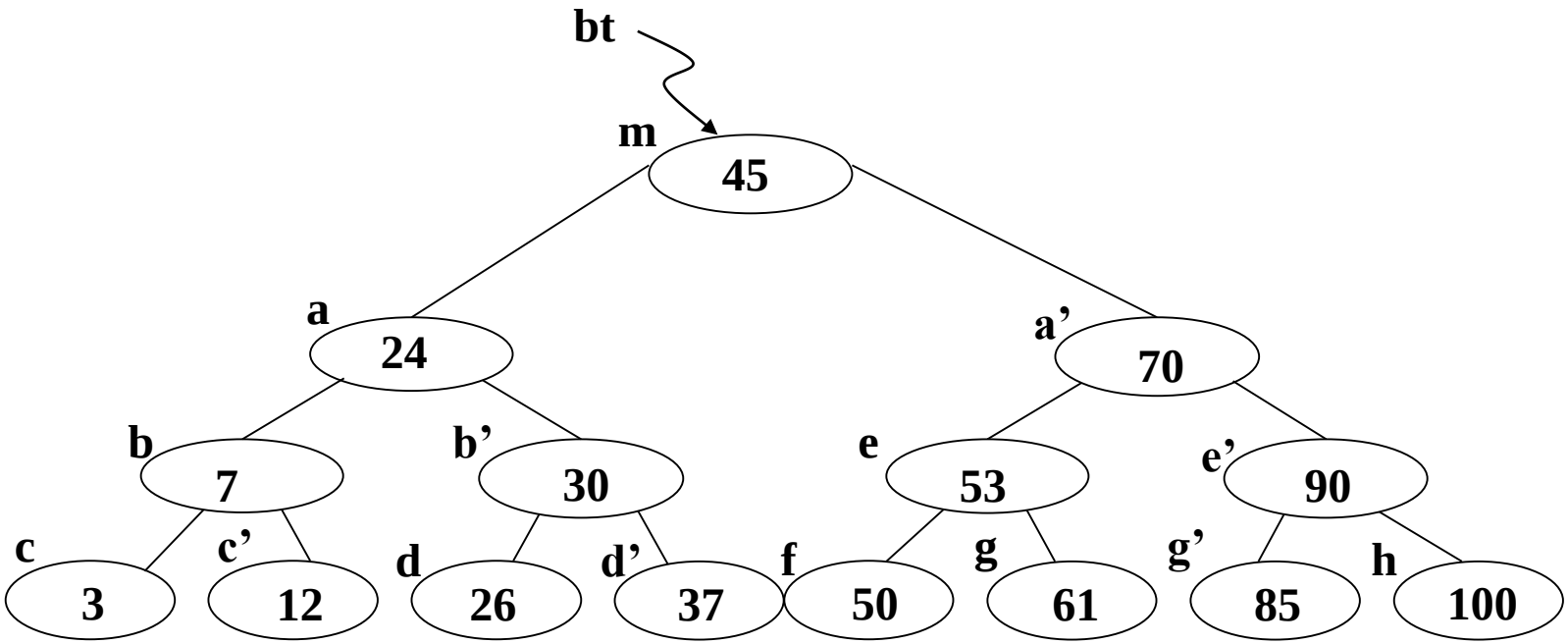
插入关键字7


bt
a
45 70
b
7 24 30
e
53
e'
90
c
3
c'
12
d
26
d'
37
f
50
g
61
g'
85
h
100

(h)


bt
a
24 45 70
b
7
b'
30
e
53
e'
90
c
3
c'
12
d
26
d'
37
f
50
g
61
g'
85
h
100

(i)



(j)

图9.8　　在B-树中进行插入（省略叶子结点）

(a)　一棵2-3树；　(b)　插入30之后；　(c)、(d)　插入26之后；
(e)~(g)　插入85之后；　(h)~(j)　插入7之后；

B - 树 删除结点

删除操作和插入结点的考虑相反

1）首先必须找到待删关键字所在结点，并且要求删除之后，结点中关键字的个数不能小于$\lceil m/2 \rceil - 1$

2）否则，要从其左(或右)兄弟结点“借调”关键字

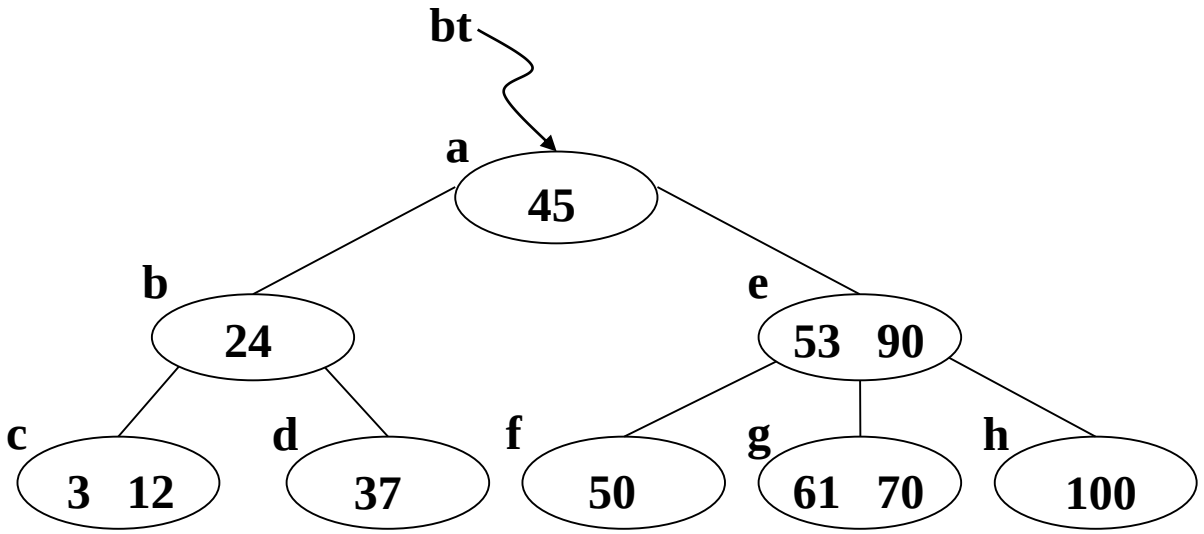
3）若其左和右兄弟结点均无关键字可借(结点中只有最少量的关键字),则必须进行结点的“合并”。

B-树的删除

1．算法思想：

在**B-**树中删除一个关键字，则首先应找到该关键字所在结点，并从中删除之。

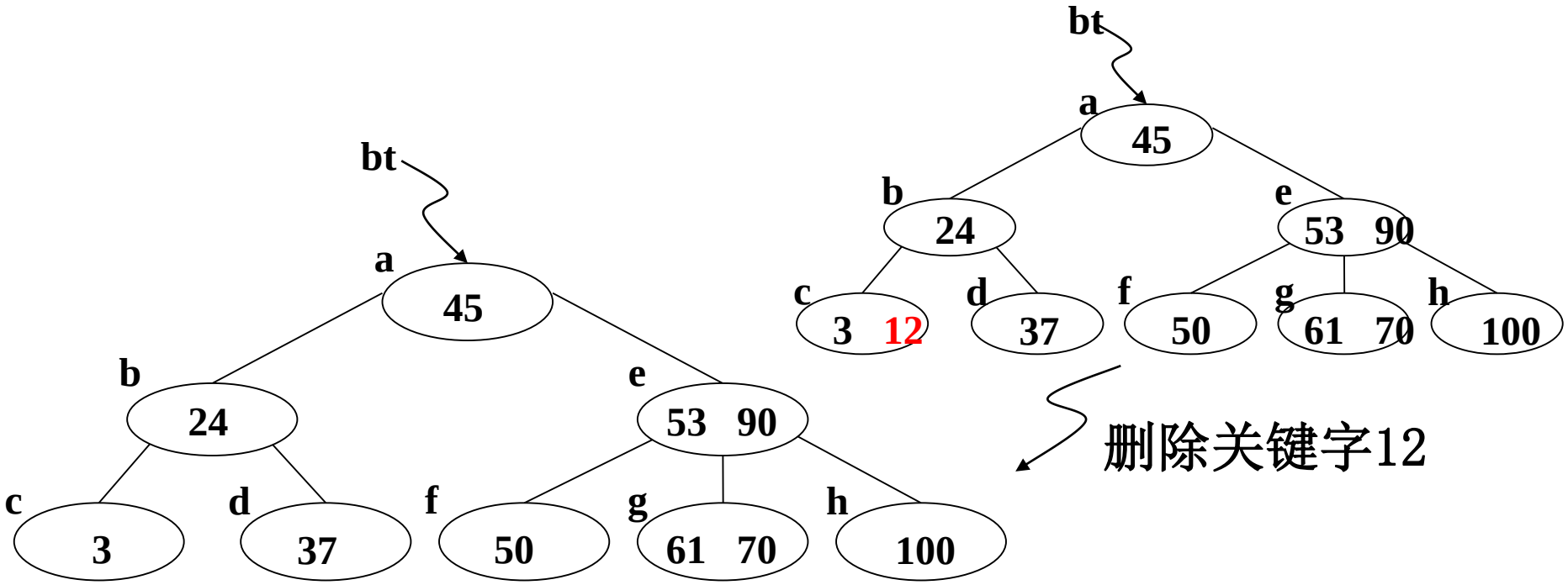
i．若所删关键字为非终端结点中的K_i，则可以指针A_i所指子树中的最小关键字**Y**替代K_i，然后在相应的结点中删去**Y**。



ii. 若所删关键字在最下层非终端结点中。有下列3种情况：

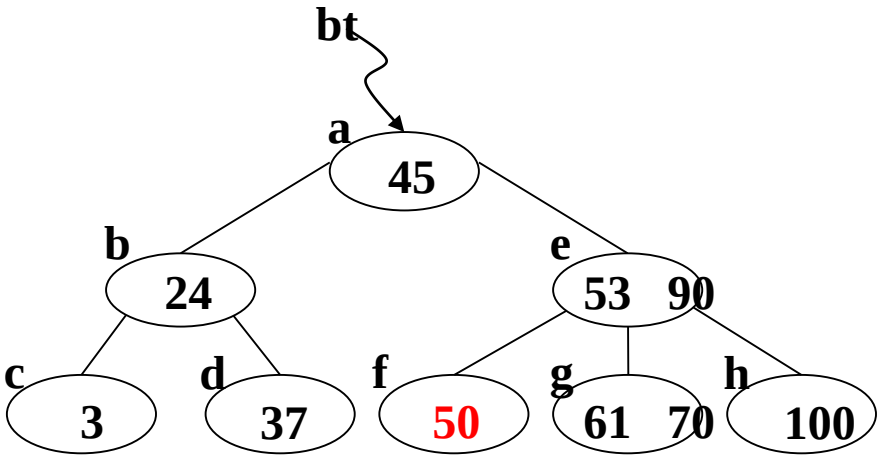
a. 被删关键字所在结点中的关键字数目不小于 $\lceil m/2 \rceil$，则只需从该结点中删去该关键字K_i和相应指针A_i，树的其他部分不变。

例如下图**3-**阶**B**树中删除关键字**12**时，直接将**12** 删除即可。

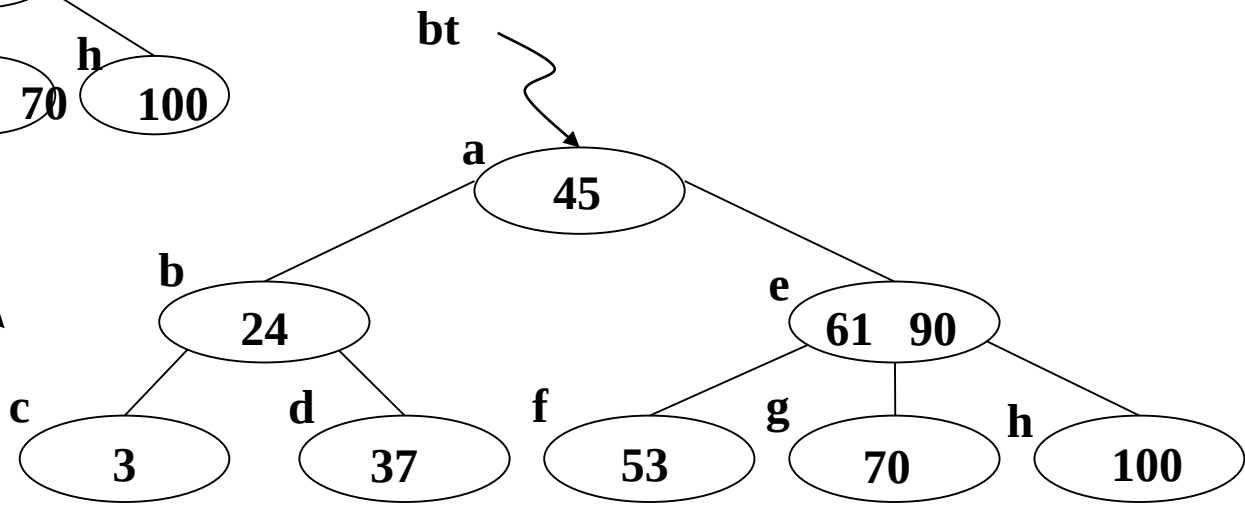


B-树结点中的关键字个数必须 $\geq \lceil m/2 \rceil - 1$

b．被删关键字所在结点中的关键字数目等于$\lceil m/2 \rceil - 1$，而与该结点相邻的右兄弟（或左兄弟）结点中的关键字数目大于$\lceil m/2 \rceil - 1$，则需将其兄弟结点中的最小（或最大）的关键字上移至双亲结点中，而将双亲结点中小于（或大于）且紧靠该上移关键字的关键字下移至被删关键字所在结点中。



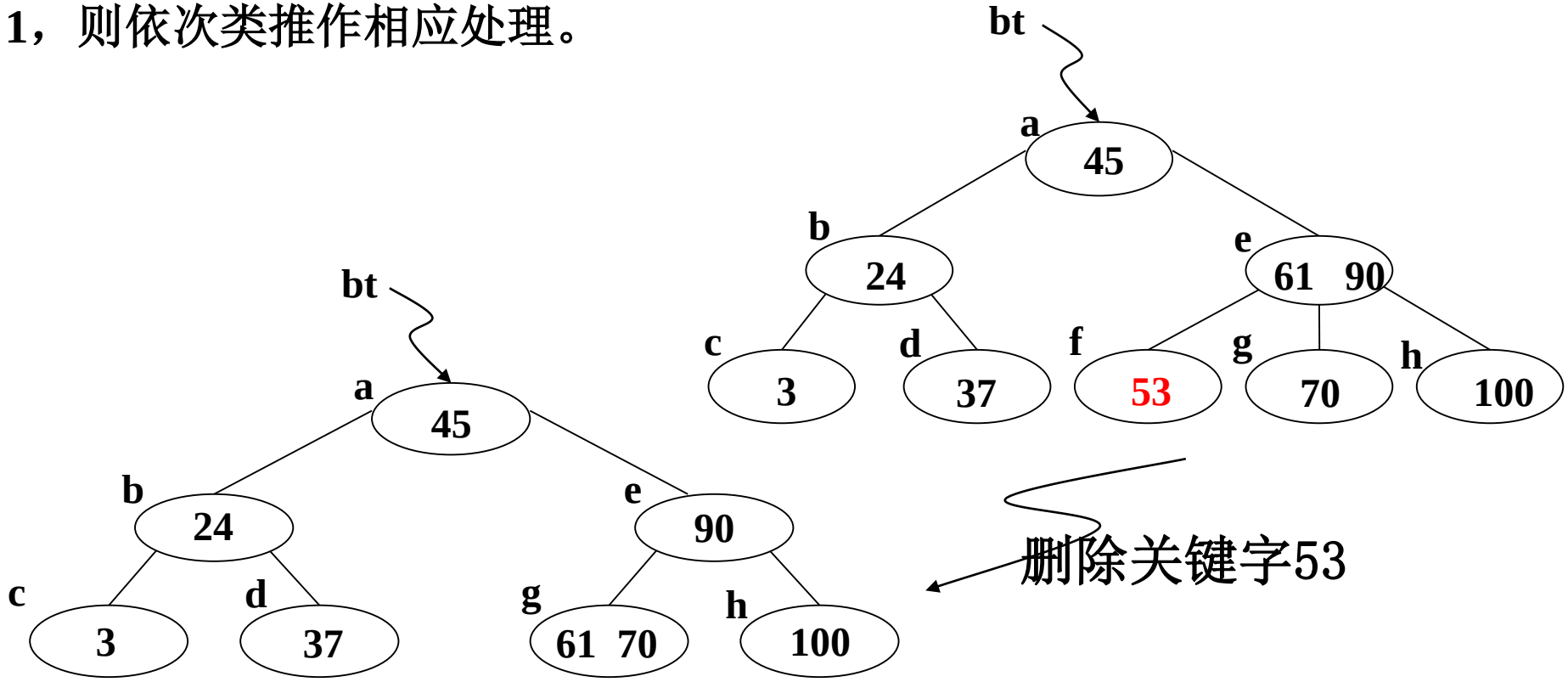
删除关键字50



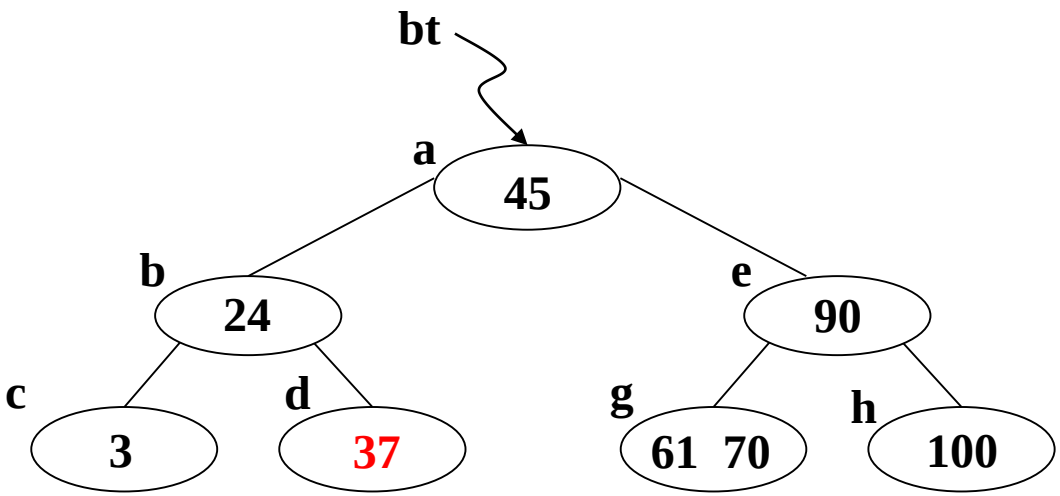
B-树结点中的关键字个数必须$\geq \lceil m/2 \rceil - 1$

c．被删关键字所在结点和其相邻的兄弟结点中的关键字数目均等于 $\lceil m/2 \rceil - 1$。

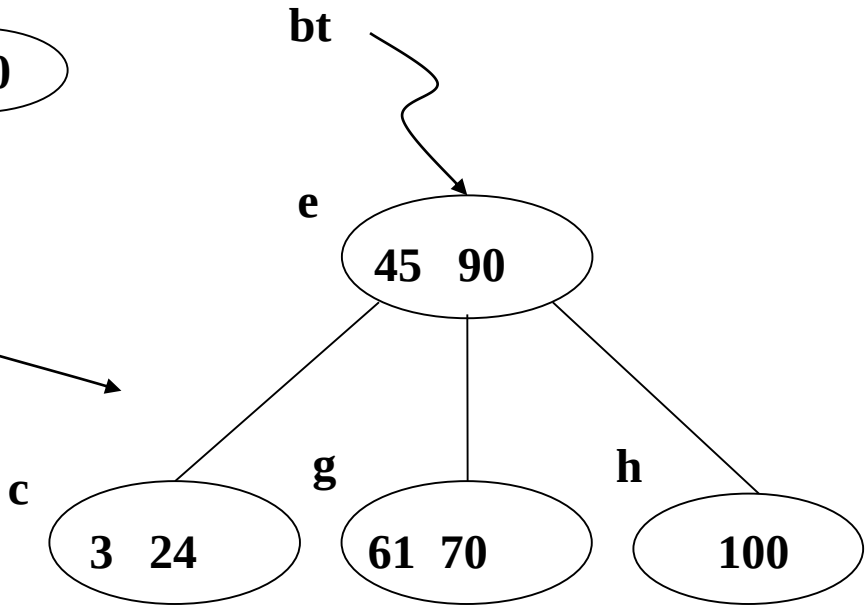
假设该结点有右兄弟，且其右兄弟结点地址由双亲结点中的指针A_i所指，则在删去关键字之后，它所在结点中剩余的关键字和指针，加上双亲结点中的关键字K_i一起，合并到A_i所指兄弟结点中（若没有右兄弟，则合并至左兄弟结点中）。如果因此使双亲结点中的关键字数目小于$\lceil m/2 \rceil - 1$，则依次类推作相应处理。



B-树结点中的关键字个数必须 $\geq \lceil m/2 \rceil - 1$



删除关键字37



双亲b结点中剩余信息(“指针c”）应和其双亲*a结点中关键字45一起合并至右兄弟结点*e中。

性能分析

在B-树中进行查找时，其查找时间主要花费在搜索结点（访问外存）上，即主要取决于B-树的深度。

问：1）含 N 个关键字的 m 阶 B-树可能达到的最大深度 H 为多少？

性能分析

2）深度为H的B-树中，至少含有多少个结点？

先推导每一层所含最少结点数：

第 1 层　　1 个

第 2 层　　2 个

第 3 层　　$2\times\lceil m/2\rceil$ 个

第 4 层　　$2\times(\lceil m/2\rceil)^2$ 个

…　　…

第 H+1 层　　$2\times(\lceil m/2\rceil)^{H-1}$ 个

性能分析

假设 m 阶 B-树的深度为 H+1，由于第 H+1 层为叶子结点，而当前树中含有 N 个关键字，则叶子结点必为 N+1 个

由此可推得下列结果：

$$N+1 \geq 2(\lceil m/2 \rceil)^{H-1}$$

$$H-1 \leq \log_{\lceil m/2 \rceil}((N+1)/2)$$

$$H \leq \log_{\lceil m/2 \rceil}((N+1)/2)+1$$

性能分析

结论：

在含 n 个关键字的 B-树上进行一次查找，需访问的结点个数不超过 $\log_{\lceil m/2 \rceil}((n+1)/2)+1$