9.2.2 B-树和B+树

1．B-树的定义

- B-树中所有结点的最大度称为B-树的阶，通常用*m*表示，从查找效率考虑，要求*m*≥3。
- 一棵*m*阶B-树或者是一棵空树，或者是满足下列要求的*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。

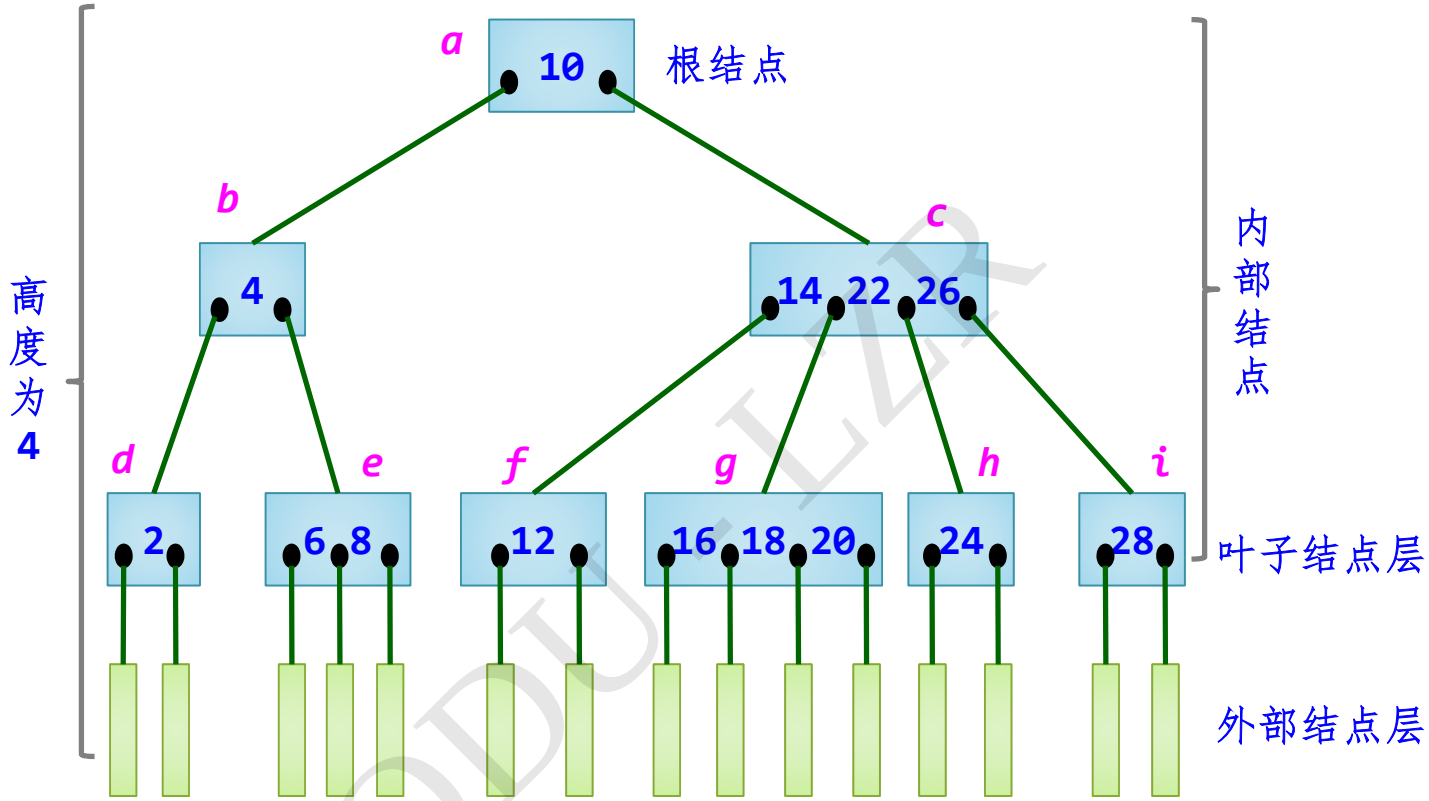
③ 除根结点外，所有内部结点至少有$\lceil m/2 \rceil$棵子树。

④ 所有的内部结点中包含下列信息数据：

n	P_0	K_1	P_1	K_2	P_2	…	K_n	P_n

- K_i（i=1，…，n）为关键字，且K_i<K_{i+1}（i=1，…，n-1）；
- P_i（i=0，…，n）为指向子树根结点的指针，且指针P_{i-1}所指子树中所有结点的关键字均小于K_i（i=1，…，n），P_n所指子树中所有结点的关键字均大于K_n。
- n为该结点中关键字个数，并且满足$\lceil m/2 \rceil$-1≤n≤m-1（设min=$\lceil m/2 \rceil$-1，max=m-1）。

⑤ 树的所有外部结点都出现在同一层次上，并且不带信息（实际上这些结点不存在，指向这些结点的指针为空）。



一棵4阶B-树

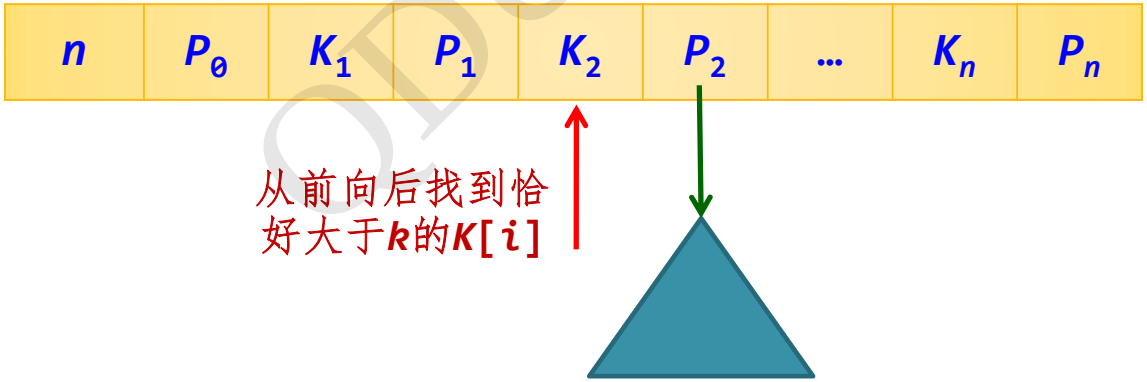
- $m=4$
- 内部结点关键字最多个数$max=m-1=3$
- 内部结点关键字最少个数$min=\lceil m/2 \rceil-1=1$

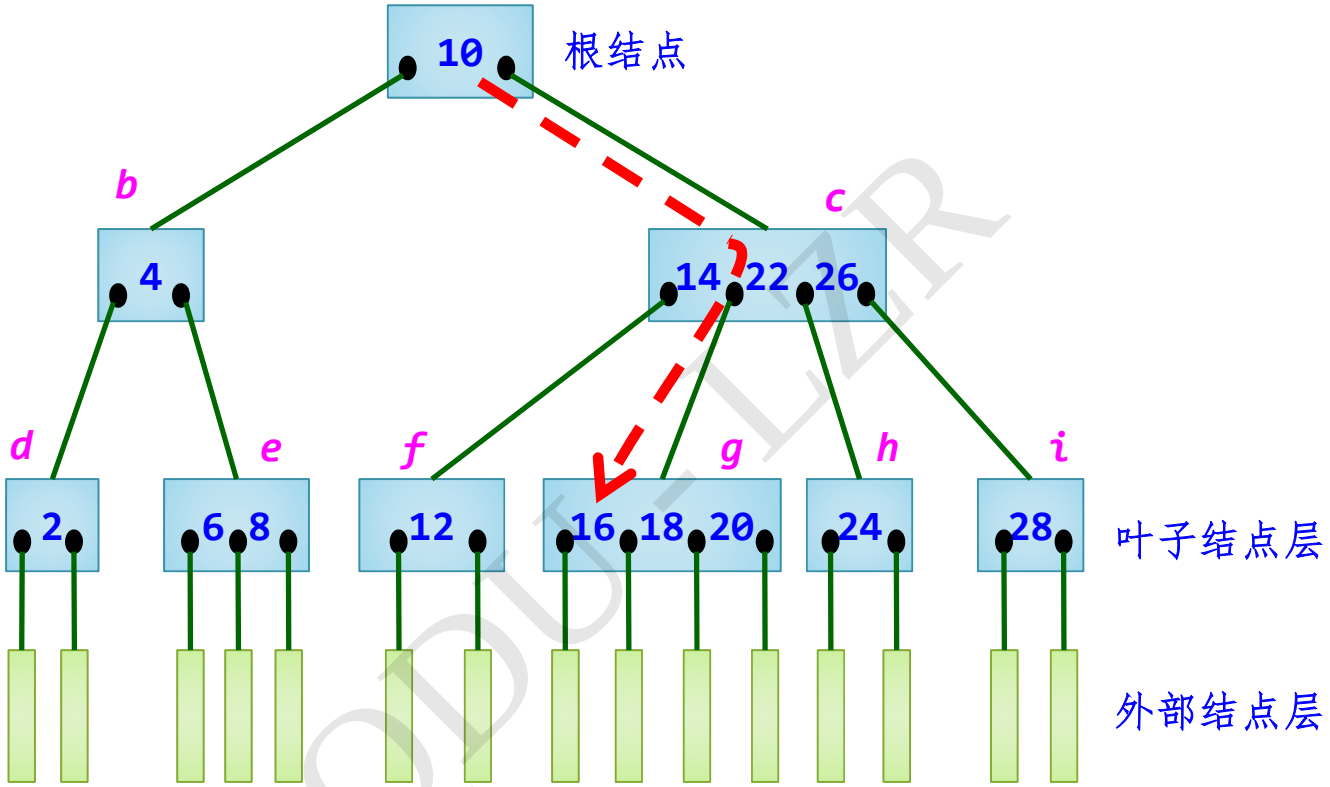
2．B-树的查找

- 在B-树中查找给定关键字的方法类似于二叉排序树上的查找，不同的是对每个结点确定向下查找的路径不一定是二路（即二叉）的，而是*n*+1路的。
- 因为结点内的关键字key[1..*n*]序列是有序的，故既可以用顺序查找关键字为*k*的方法查找，也可以用折半查找。

将k与根结点比较开始：

- 若$k=K[i]$，则查找成功；
- 若$k<K[1]$，则沿着指针$P[0]$所指的子树继续查找；
- 若$K[i]<k<K[i+1]$，则沿着指针$P[i]$所指的子树继续查找；
- 若$k>K[n]$，则沿着指针$P[n]$所指的子树继续查找。





查找关键字16

3．B-树的插入

向*m*阶B-树中插入一个关键字*k*的步骤如下：

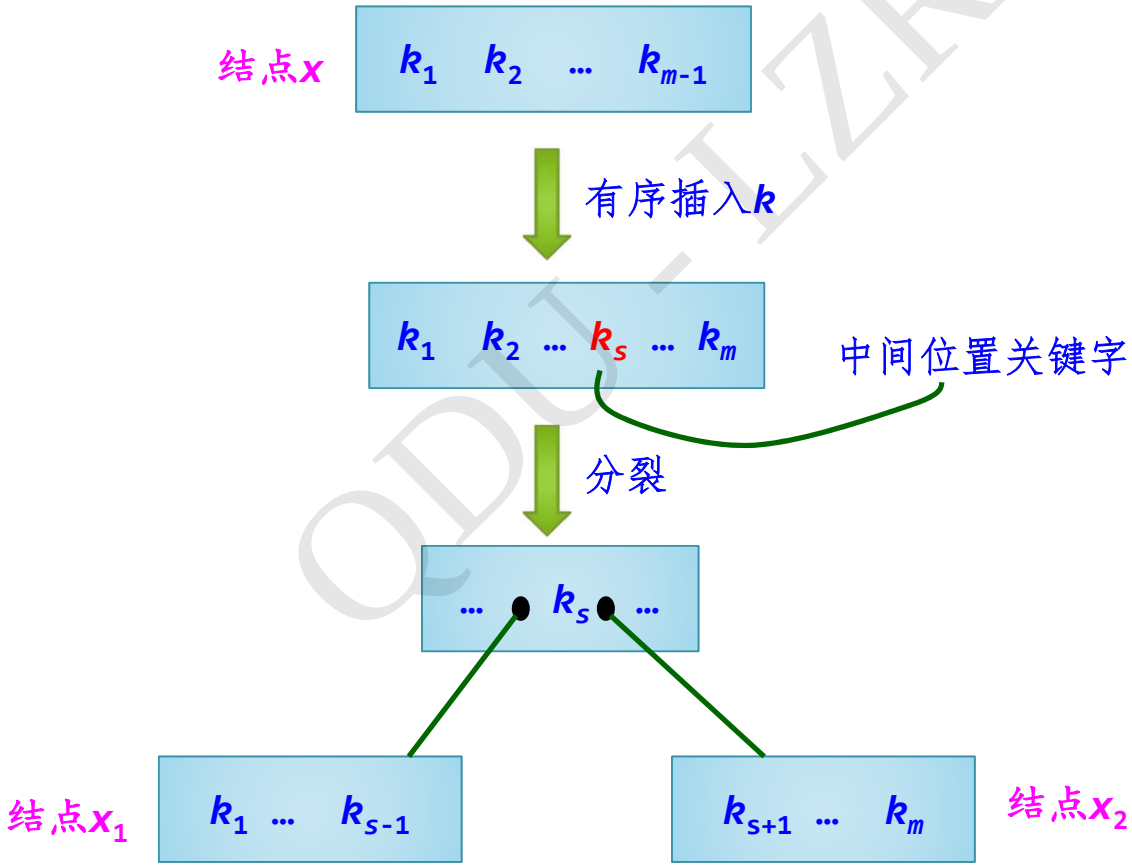
（1）查找：从根结点开始比较，类似于查找过程，找到一个合适的叶子结点来插入关键字*k*。也就是说，关键字*k*一定是插入到某个叶子结点中。

（2）插入：在叶子结点*x*中插入关键字*k*。

在叶子结点x中插入关键字k。分为两种情况：

① 若x结点的关键字个数小于max（m-1）⇨ 直接有序插入k。

② 若x结点的关键字个数恰好为max，⇨ **分裂结点：**

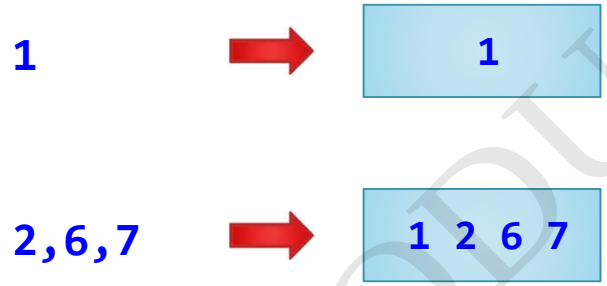


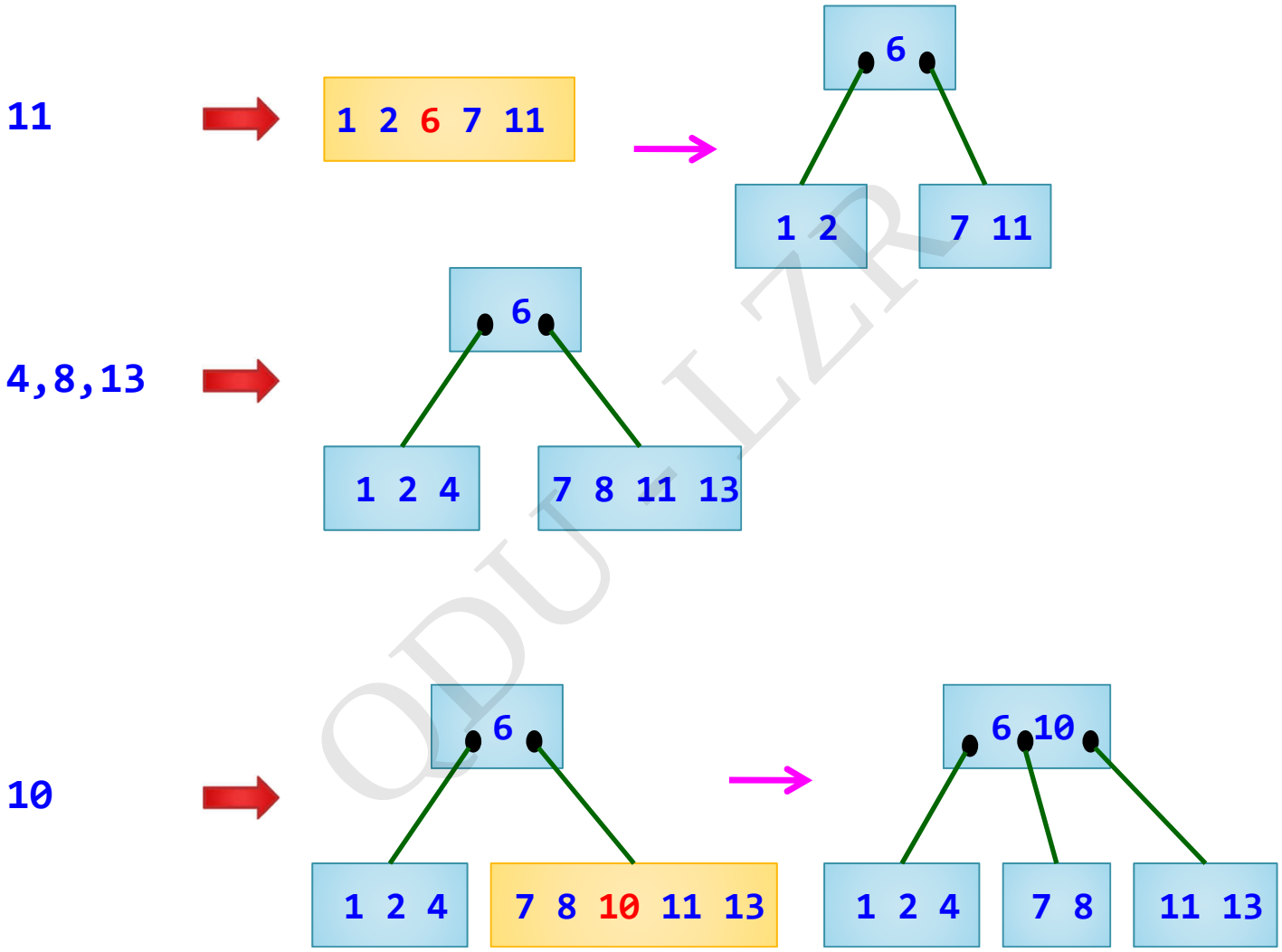
4．B-树的创建

- 给定一个关键字序列创建一棵*m*阶B-树的过程是，从一棵空树开始，扫描所有的关键字*k*，采用前面介绍的B-树插入方式将其插入到B-树中。
- 显然，*m*值的不同，构造的B-树是不同的。

【示例-1】给定的关键字序列为（1，2，6，7，11，4，8，13，10，5，17，9，16，20，3，12，14，18，19，15），创建一棵5阶B-树。

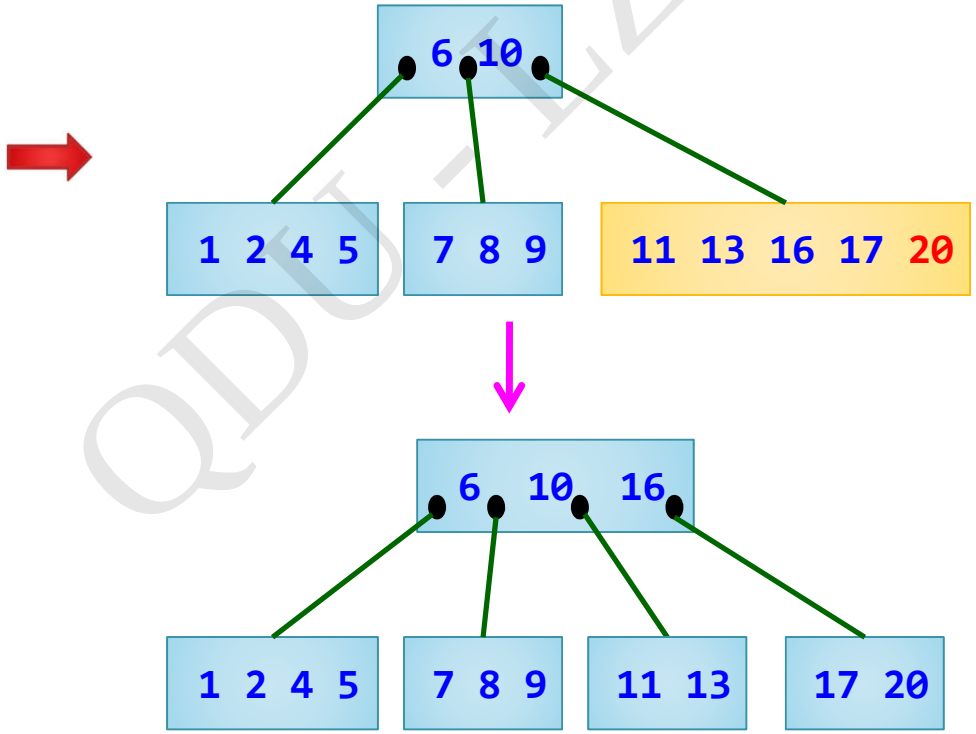
解：这里m=5，结点中最大关键字个数max=m-1=4。



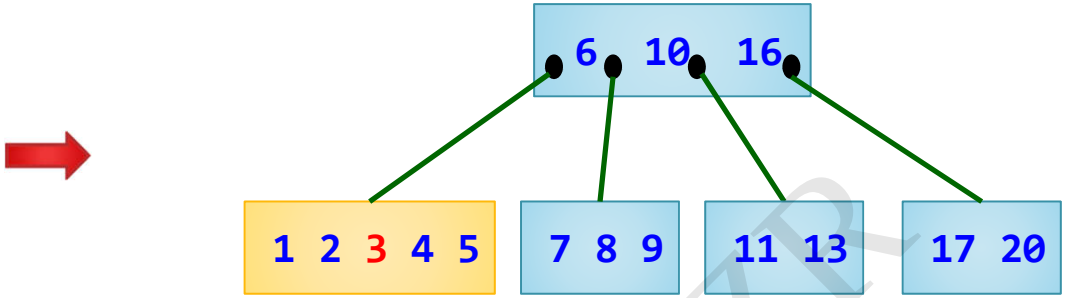

11
1 2 6 7 11
6
1 2
7 11
4,8,13
6
1 2 4
7 8 11 13
10
6
1 2 4
7 8 10 11 13
6 10
1 2 4
7 8
11 13

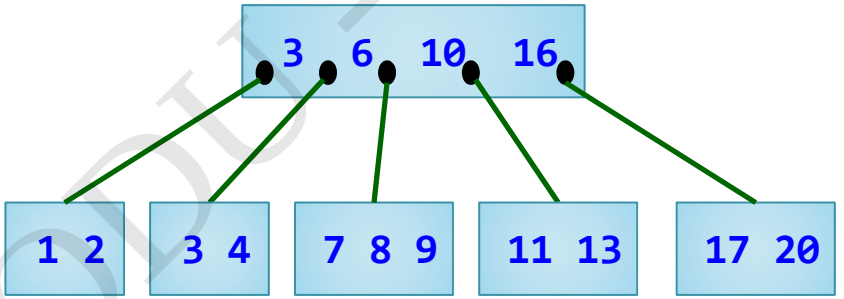
5,17,9,16

20

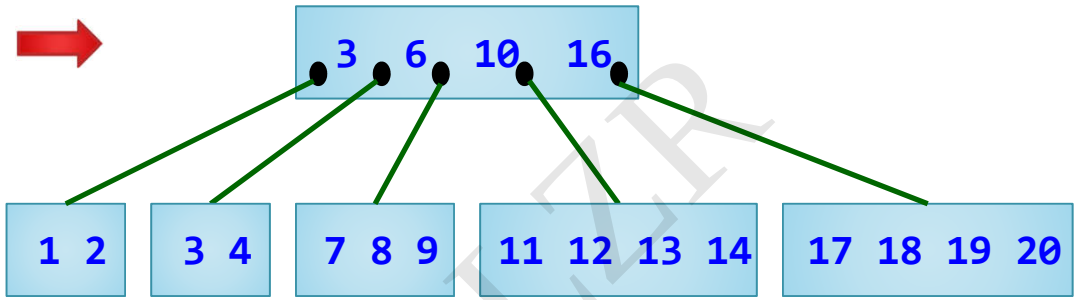


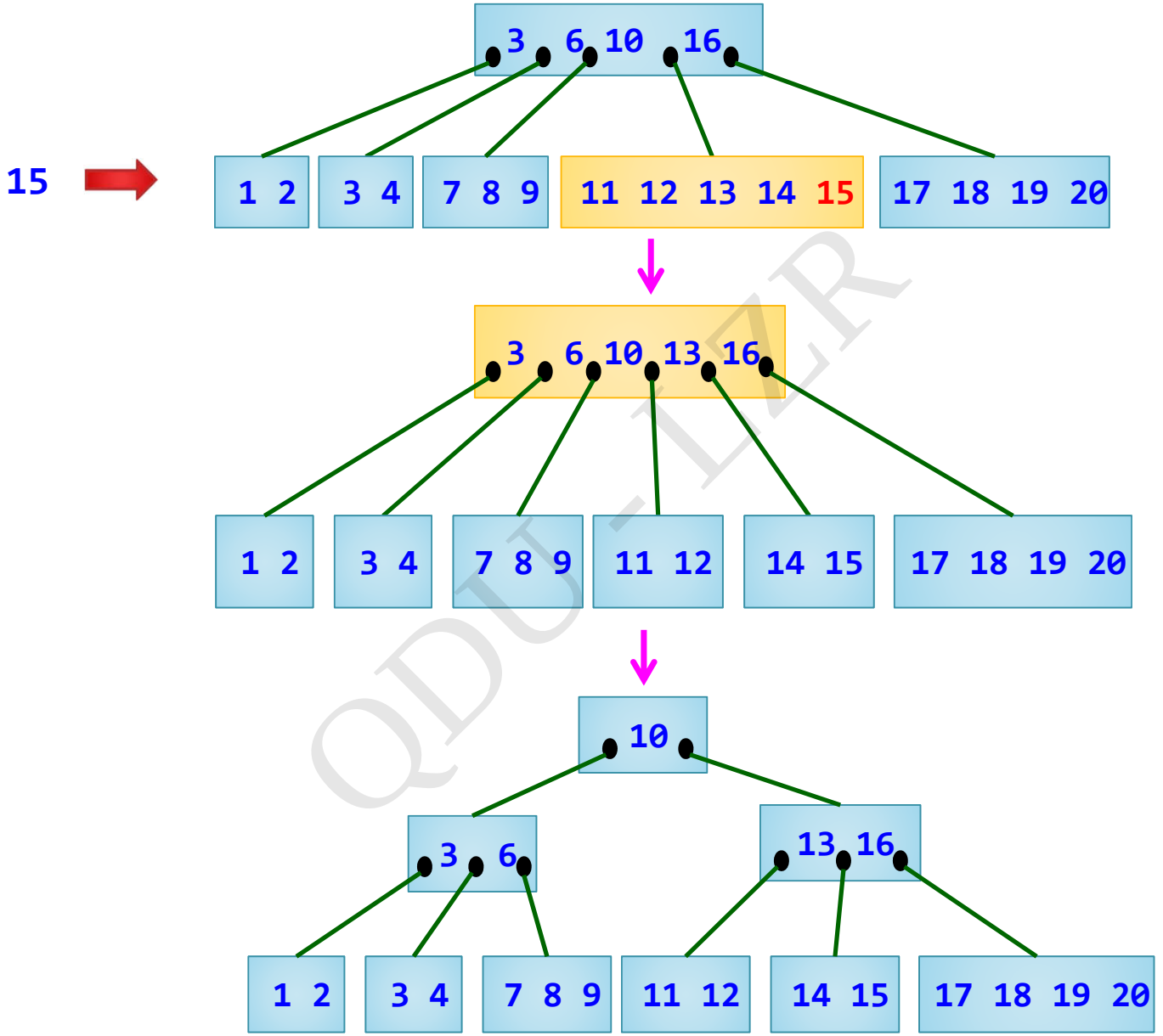
3





12,14,18,19




15
3 6 10 16
1 2
3 4
7 8 9
11 12 13 14 15
17 18 19 20
3 6 10 13 16
1 2
3 4
7 8 9
11 12
14 15
17 18 19 20
10
3 6
13 16
1 2
3 4
7 8 9
11 12
14 15
17 18 19 20

5．B-树的删除

在*m*阶B-树中删除一个关键字*k*的步骤如下：

（1）查找：从根结点开始比较，类似于查找过程，找到包含关键字*k*的结点。

（2）删除：在该结点中删除关键字*k*分两种情况：一种是在叶子结点上删除关键字*k*；另一种是在非叶子结点上删除关键字*k*。

- 在非叶子结点上删除关键字k可以转化为在叶子结点中删除：

  - 如果这个非叶子结点中某个关键字$k_i=k$，则以指针P_i所指子树中的最小关键字$mink$替代k_i。mink所在的结点一定是叶子结点。
  - 然后在相应的叶子结点中删除mink；

对称地：也可以用指针P_{i-1}所指子树中的最大关键字maxk替代k_i，然后在相应的叶子结点中删除maxk。

- 在B-树的叶子结点y中删除关键字mink共有以下三种情况：

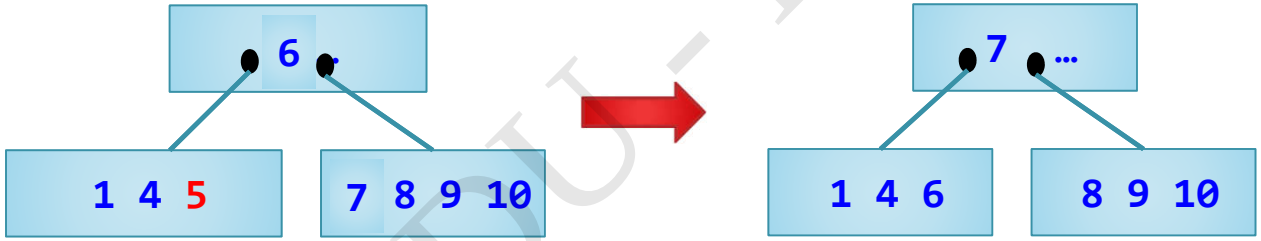
（1）从y结点中删除关键字mink后，其中关键字个数仍大于等于min（min=⌈m/2⌉-1），直接删除关键字mink，删除过程结束。

例如：

- min=3
- mink=1

（2）从y结点中删除关键字mink后，其中关键字个数少于min，而与y结点相邻的左兄弟（或右兄弟）结点中的关键字多于min个 ⇨ 则向兄弟借一个关键字。

例如：

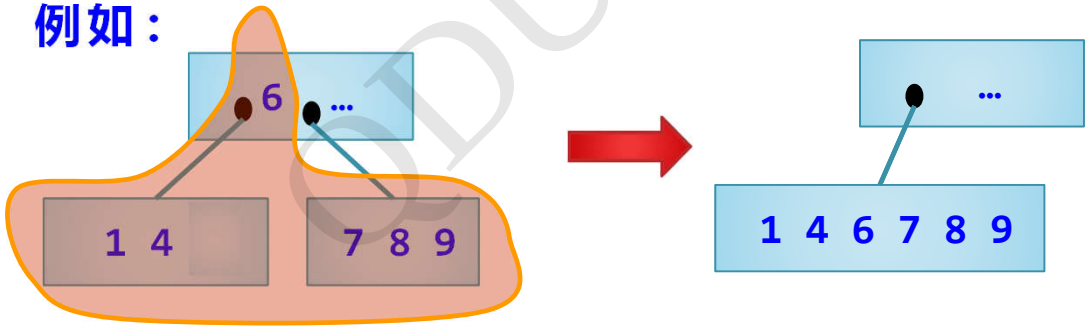


- min=3
- mink=5

（3）从y结点中删除关键字mink后，其中关键字个数少于min，而且y结点左右相邻的兄弟结点中的关键字都是min个 ⇨ 则可将y结点和它的双亲结点中的一个关键字合并到它的兄弟结点中。

如果这样合并后使双亲结点中的关键字少于min个，则需要继续进行合并，直到根结点。

例如：



- min=3
- mink=5

8.3.4 B+树

在索引文件组织中，经常使用B-树的一些变形，其中B+树是一种应用广泛的B-树的变形。

一棵*m*阶B+树满足下列条件：

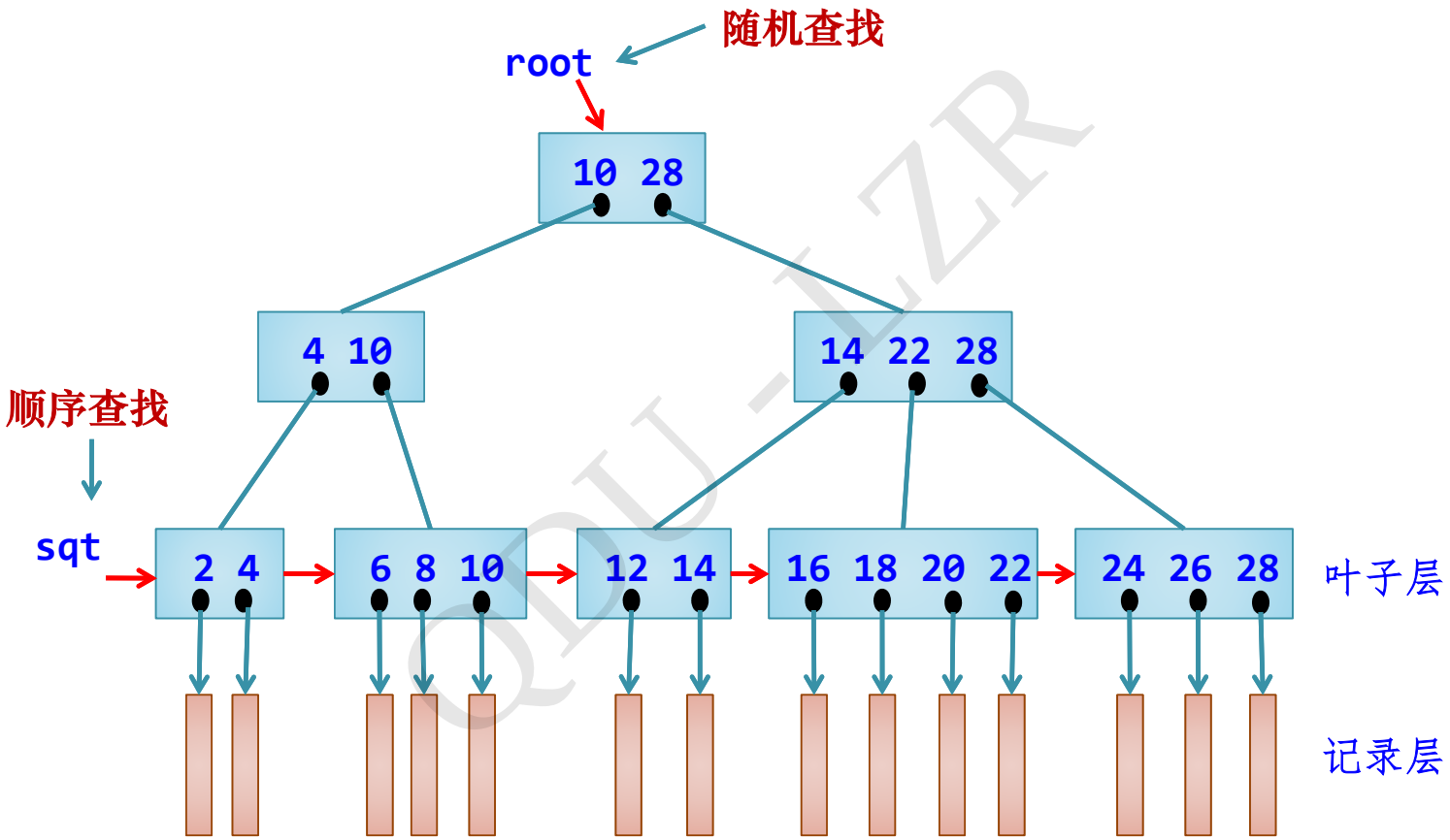
① 每个分支结点至多有*m*棵子树。

② 根结点或者没有子树，或者至少有两棵子树。

③ 除根结点外，其他每个分支结点至少有$\lceil m/2 \rceil$棵子树。

④ 有*n*棵子树的结点有*n*个关键字。

一棵4阶的B+树示例：



⑤ 所有叶子结点包含全部关键字及指向相应记录的指针，而且叶子结点按关键字大小顺序链接（可以把每个叶子结点看成是一个基本索引块，它的指针不再指向另一级索引块，而是直接指向数据文件中的记录）。

⑥ 所有分支结点（可看成是索引的索引）中仅包含它的各个子结点（即下级索引的索引块）中最大关键字及指向子结点的指针。

m阶的B+树和m阶的B-树的差异是：

- B+树中具有n个关键字的结点含有n棵子树，即每个关键字对应一棵子树。而在B-树中，具有n个关键字的结点含有(n+1)棵子树。
- B+树中所有非叶子结点仅起到索引的作用，而所有叶子结点包含了全部关键字。而在B-树中，叶子结点包含的关键字与其他结点包含的关键字是不重复的。
- B+树支持随机查找和顺序查找。而B-树仅仅支持随机查找。

END