B-树

2019年8月5日　10:26

B-树中所有结点中孩子结点数的最大值成为B-树的阶，通常使用m表示，一般要求m>=3

(1）每个结点最多有m个分支（子树）；而最少分支数要看是否为根结点，如果是根结点且不是叶子结点，则至少要有两个分支，非根非叶结点至少有（m/2，向上取整)个分支。

(2)如果一个结点有n-1个关键词，那么该结点有n个分支。这个n-1个关键字按照递增序列排序。

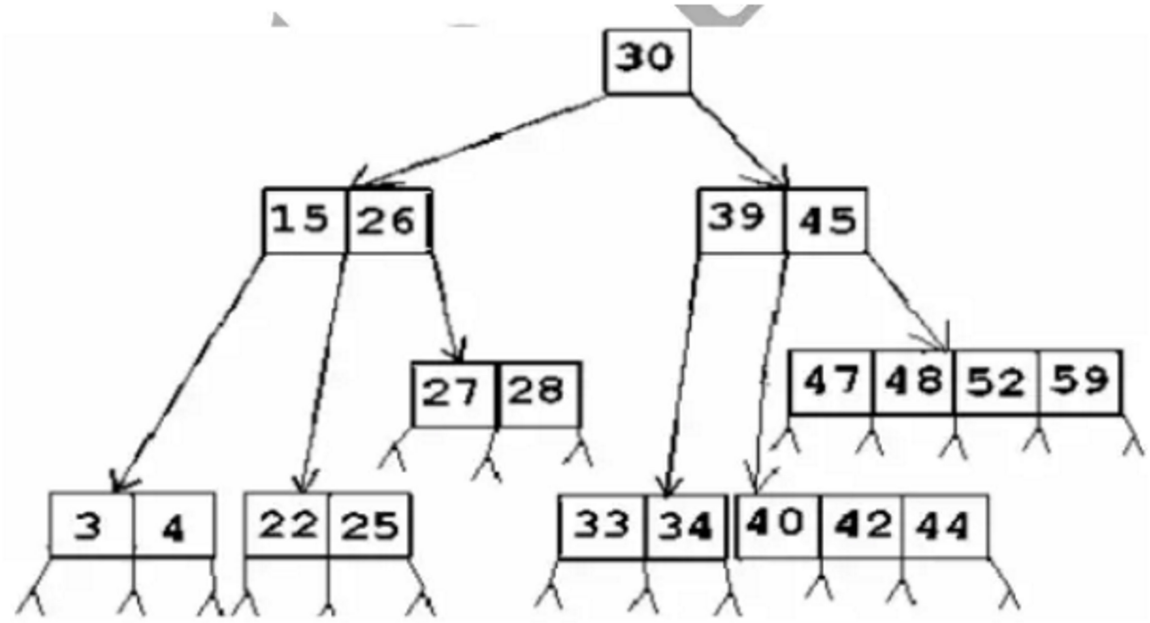
(3)每个结点的结构为：

n	k1	k2	...	kn
p0	p1	p2	...	pn

其中，n为该结点中关键字的个数，ki为该结点的关键字且满足ki<ki+1；pi为该结点的孩子结点指针且满足pi所指结点上的关键字大于ki且小于ki+1，p0所指结点上关键字小于k1，pn所指结点上关键字大于kn

(4)结点内各关键字互不相等且按从小到大排序。

(5)叶子结点处于同一层；可以用空指针表示，是查找失败到达的位置



上面图片显示了一颗B-树，最底层的叶子结点没有显示，

1）结点的分支数等于关键字数+1，最大的分支数就是B-树的阶数，因此m阶的B-树中结点最多有m个分支，所以可以看到，上面的一棵树是一个5-阶B-树。

2）因为上面是一棵5阶B-树，所以非根非叶结点至少要有ceil(5/2)=3个分支。根结点可以不满足这个条件，图中的根结点有两个分支。

3）如果根结点中没有关键字就没有分支，此时B-树是空树，如果根结点有关键字，则其分支数比大于或等于2，因为分支数等于关键字数+1.

4）上图中除根结点外，结点中的关键字个数至少为2，因为分支数至少为3，分支数比关键字数多1，还可以看出结点内关键字都是有序的，并且在同一层中，左边结点内所有关键字均小于右边结点内的关键字，例如，第二层上的两个结点，左边结点内的关键字为15，26，他们均小于右边结点内的关键字39和

45.

B-树一个很重要的特征是，下层结点内的关键字取值总是落在由上层结点关键字所划分的区间内，具体落在哪个区间内可以由指向它的指针看出。例如，第二层最左边的结点内的关键字划分了三个区间，小于15，15到26，大于26，可以看出其下层中最左边结点内的关键字都小于15，中间结点的关键字在15和26之间，右边结点的关键字大于26.

5）上图中叶子结点都在第四层上，代表查找不成功的位置

B-树的查找操作

B-树的查找很简单，是二叉排序树的扩展，二叉排序树是二路查找，B-树是多路查找，因为B-树结点内的关键字是有序的，在结点内进行查找时除了顺序查找外，还可以用折半查找提升效率。

1）先让key与根结点中的关键字比较，如果key等于k[i]（k[]为结点内的关键字数组），则查找成功

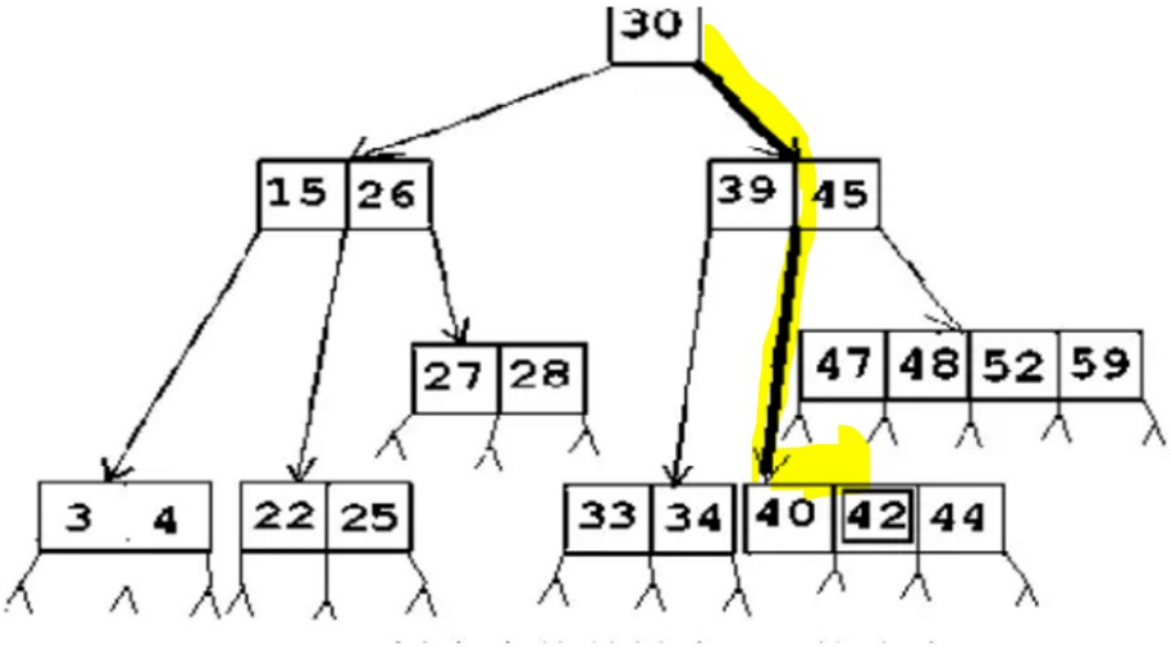
2）若key<k[1]，则到p[0]所指示的子树中进行继续查找（p[]为结点内的指针数组），这里要注意B-树中每个结点的内部结构。

3）若key>k[n]，则道p[n]所指示的子树中继续查找。

4）若k[i]<key<k[i+1]，则沿着指针p[I]所指示的子树继续查找。

5）如果最后遇到空指针，则证明查找不成功。

拿上面的二叉树进行举例，比如我们想要查找关键字42，下图加粗的部分显示了查找的路径：



3、B-树的插入

与二叉排序树一样，B-树的创建过程也是将关键字逐个插入到树中的过程

在进行插入之前，要确定一下每个结点中关键字的范围，如果B-树的阶数为m，则结点中关键字个数的范围为ceil(m/2)-1~m-1个

对于关键字的插入，需要找到插入位置。在B-树的查找过程中，当遇到空指针时，则证明查找不成功，同时也找到了插入位置，即根据空指针可以确定在最底层非叶结点中的插入位置，为了方便，我们称最底层的非叶结点为**终端结点**，由此可见，B-树结点的插入总是落在终端结点上。在插入过程中有可能破坏B-树的特征，如新关键字的插入使得结点中关键字的个数超过规定个数，这是要进行**结点的拆分**。

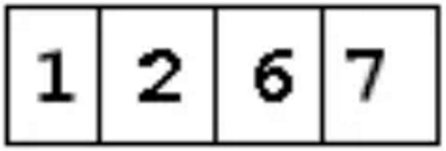
接下来，我们以关键字序列{1,2,6,7,11,4,8,13,10,5,17,9,16,20,3,12,14,18,19,15}创建一棵5阶B-树，我们

将详细体会B-树的插入过程。

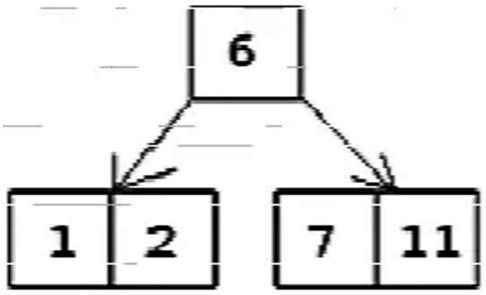
（1）确定结点中关键字个数范围

由于题目要求建立5阶B-树，因此关键字的个数范围为2～4

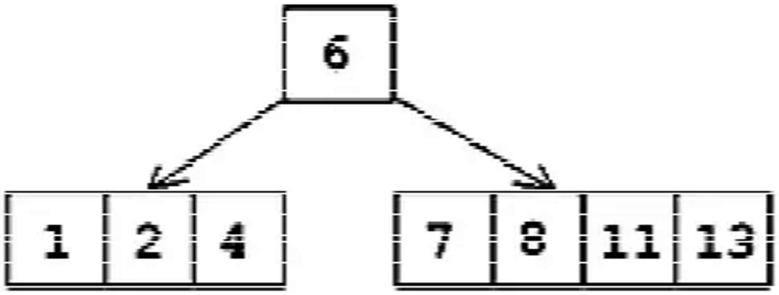
（2）根结点最多可以容纳4个关键字，依次插入关键字1、2、6、7后的B-树如下图所示：



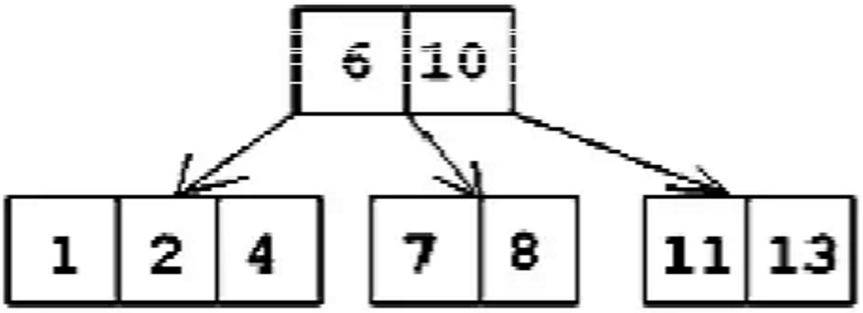
（3）当插入关键字11的时候，发现此时结点中关键字的个数变为5，超出范围，需要拆分，去关键字数组中的中间位置，也就是k[3]=6，作为一个独立的结点，即新的根结点，将关键字6左、右关键字分别做成两个结点，作为新根结点的两个分支，此时树如下图所示



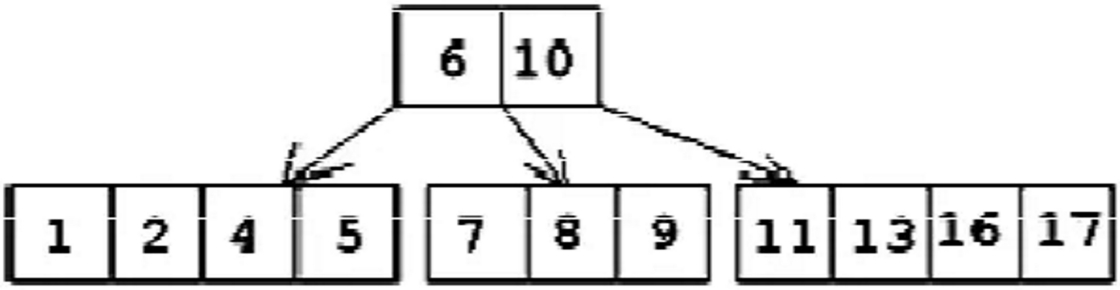
（4）新关键字总是插在叶子结点上，插入关键字4、8、13之后树为：



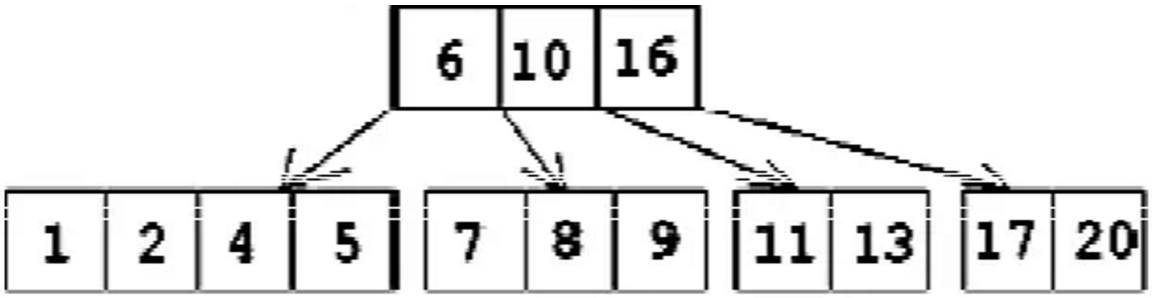
（5）关键字10需要插入在关键字8和11之间，此时又会出现关键字个数超出范围的情况，因此需要拆分。拆分时需要将关键字10纳入根结点中，并将10左右的关键字做成两个新的结点连在根结点上。插入关键字10并经过拆分操作后的B-树如下图：



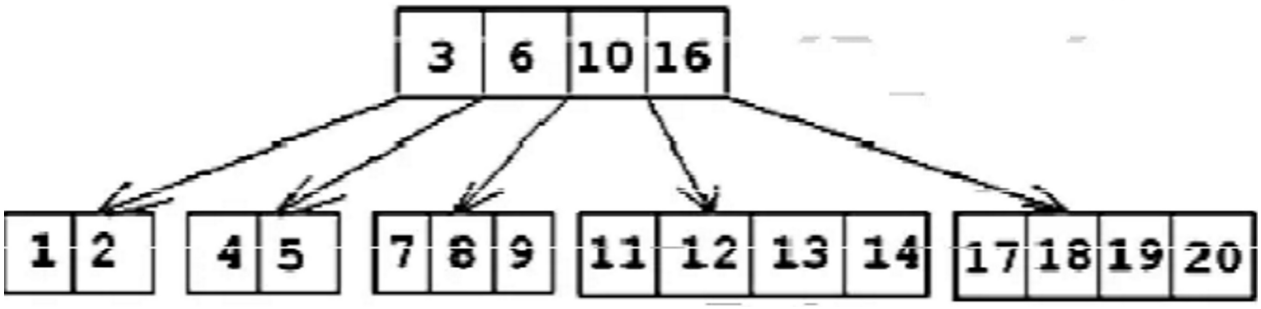
（6）插入关键字5、17、9、16之后的B-树如图所示：



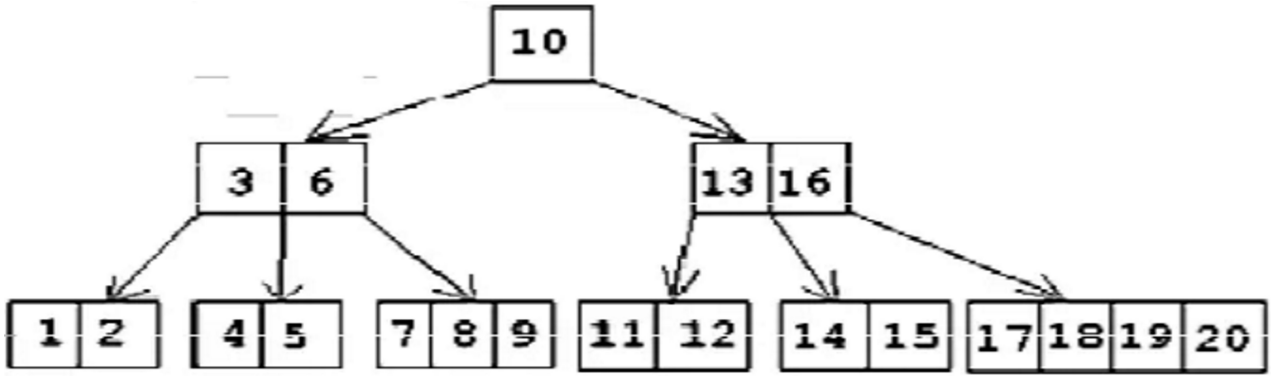
（7）关键字20插入在关键字17以后，此时会造成结点关键字个数超出范围，需要拆分，方法同上，树为



（8）按照上述步骤依次插入关键字3、12、14、18、19之后B-树如下图所示：



（9）插入最后一个关键字15，15应该插入在14之后，此时会出现关键字个数超出范围的情况，则需要进行拆分，将13并入根结点，13并入根结点之后，又使得根结点的关键字个数超出范围，需要再次进行拆分，将10作为新的根结点，并将10左、右关键字做成两个新结点连接到新根结点的指针上，这种插入一个关键字之后出现多次拆分的情况称为**连锁反应**，最终形成的B-树如下图所示：



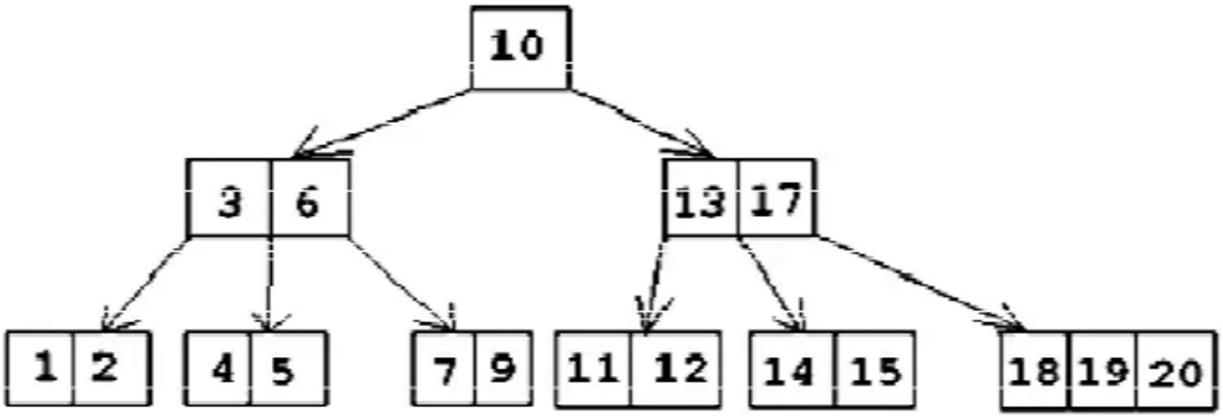
4、B-树的删除

对于B-树关键字的删除，需要找到待删除的关键字，在结点中删除关键字的过程也有可能破坏B-树的特性，如旧关键字的删除可能使得结点中关键字的个数少于规定个数，这是可能需要向其兄弟结点**借关键字**或者和其孩子结点进行**关键字的交换**，也可能需要进行**结点的合并**，其中，和当前结点的孩子进行关键字交换的操作可以保证删除操作总是发生在终端结点上。

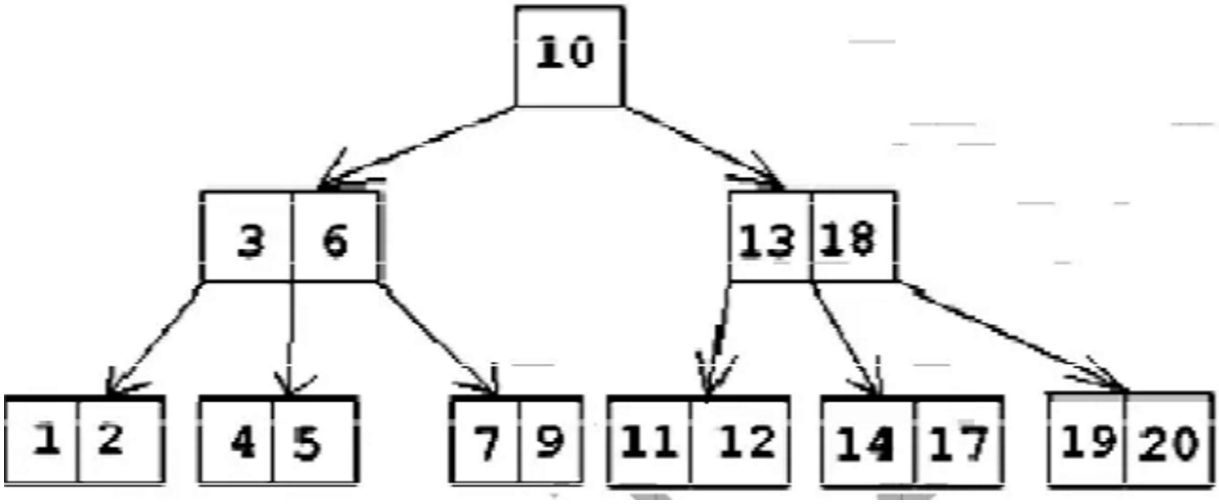
我们用刚刚生成的B-树作为例子，一次删除8、16、15、4这4个关键字。

（1）删除关键字8、16。关键字8在终端结点上，并且删除后其所在结点中关键字的个数不会少于2，因

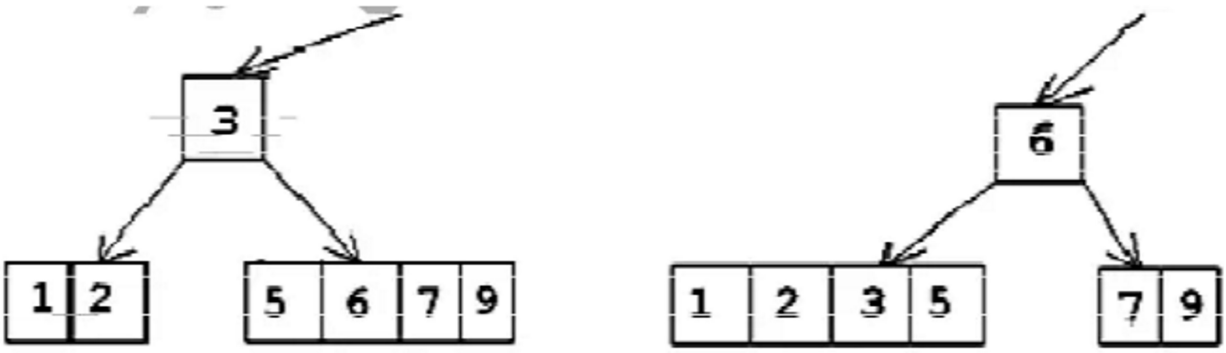
此可以直接删除。关键字16不在终端结点上，但是可以用17来覆盖16，然后将原来的17删除掉，这就是上面提到的和孩子结点进行关键字交换的操作。这里不能用15和16进行关键字交换，因为这样会导致15所在结点中关键字的个数小于2。因此，删除8和16之后B-树如下图所示：



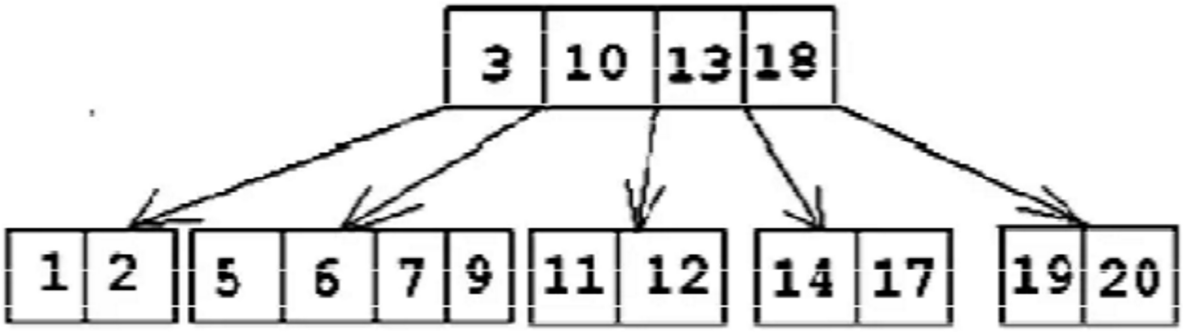
（2）删除关键字15，15虽然也在终端结点上，但是不能直接删除，因为删除后当前结点中关键字的个数小于2。这是需要向其兄弟结点借关键字，显然应该向其右兄弟来借关键字，因为左兄弟的关键字个数已经是下限2.借关键字不能直接将18移到15所在的结点上，因为这样会使得15所在的结点上出现比17大的关键字，所以正确的借法应该是先用17覆盖15，在用18覆盖原来的17，最后删除原来的18，删除关键字15后的B-树如下图所示：



（3）删除关键字4，4在终端结点上，但是此时4所在的结点的关键字个数已经到下限，需要借关键字，不过可以看到其左右兄弟结点已经没有多余的关键字可借。所以就需要进行关键字的合并。可以先将关键字4删除，然后将关键字5、6、7、9进行合并作为一个结点链接在关键字3右边的指针上，也可以将关键字1、2、3、5合并作为一个结点链接在关键字6左边的指针上，如下图所示：



显然上述两种情况下都不满足B-树的规定，即出现了非根的双分支结点，需要继续进行合并，合并后的B-树如下图所示：



有时候删除的结点不在终端结点上，我们首先需要将其转化到终端结点上，然后再按上面的各种情况进行删除。在讲述这种情况下的删除方法之前，要引入一个相邻关键字的概念，对于不在终端结点的关键字a，它的相邻关键字为其左子树中值最大的关键字或者其右子树中值最小的关键字。找a的相邻关键字的方法为：沿着a的左指针来到其子树根结点，然后沿着根结点中最右端的关键字的右指针往下走，用同样的方法一直走到叶结点上，叶结点上的最右端的关键字即为a的相邻关键字（这里找的是a左边的相邻关键字，我们可以用同样的思路找到a右边的相邻关键字）。可以看到下图中a的相邻关键字是d和e，要删除关键字a，可以用d来取代a，然后按照上面的情况删除叶子结点上的d即可。

6、B-树的应用

为了将大型数据库文件存储在硬盘上，以减少访问硬盘次数为目的，在此提出了一种平衡多路查找树——B-树结构。由其性能分析可知它的检索效率是相当高的 为了提高 B-树性能’ 还有很多种B-树的变型，力图对B-树进行改进，比如B+树。