**1.B树**

平衡二叉树的查找效率为O（log2N）与树的深度相关，通过降低树的深度，可以提高查找效率，但是还有一个瓶颈就是，每次查找一次就只能得到一个节点元素，如果查找一次能得到多个节点元素，那么在同样的高度就能够查找更多的元素，从而提高查找效率，因此提出多路查找树。

多路查找树（muitl-way search tree），其每一个结点的孩子数可以多于两个，且每个结点出可以存储多个元素。由于它是查找树，所有元素之间存在某种特定的排序关系。

B树就是一种平衡的多路查找树。

**基本概念**

B-树中所有结点中孩子结点个数的**最大值**成为**B-树的阶**，通常用m表示，从查找效率考虑，一般要求**m>=3**。一棵m阶B-树或者是**一棵空树**，或者是满足以下条件的**m叉树**。

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**ceil(m/2)个分支**，这里ceil代表向上取整。

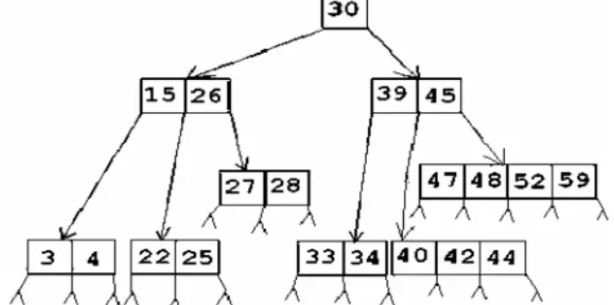
2）如果一个结点有n-1个关键字，那么该结点有n个分支。这n-1个关键字按照递增顺序排列。

3）每个节点的结构为

节点个数：n；

关键字数组： k[n]，这n个关键字按照递增顺序排列

孩子指针数组：p[n + 1], p0<=k1, 之后所有 ki < pi <= ki+1;



**查找操作**

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。

如果查找22，过程为22小于30，往p0指针走，到达15,26这个内节点，然后对比发现属于15到26区间，往p[1]走，找到22；

**插入操作**

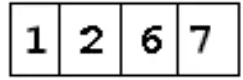
要确定一下每个结点中关键字个数的范围，如果B-树的阶数为m，则结点中关键字个数的范围为**ceil(m/2)-1 ~ m-1**个。

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**，中间结点为m/2向上取整的位置，这是要进行按**中间结点的拆分**，将中间结点放到父节点，如果父节点也不满足要求，就一直到网上分裂

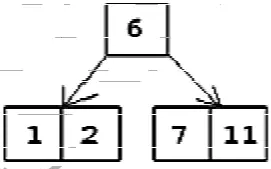
按照关键字序列来，{1,2,6,7,11,4,8,13,10,5,17,9,16,20,3,12,14,18,19,15}

创建5阶b数，每个节点的关键字范围2-4

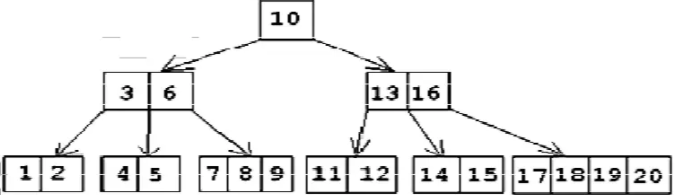
①根节点最多容纳4个关键字，插入1,2,6,7，的b树如图所示



②插入11时，超过4，此时需要分裂5/2向上取整，中间结点上移



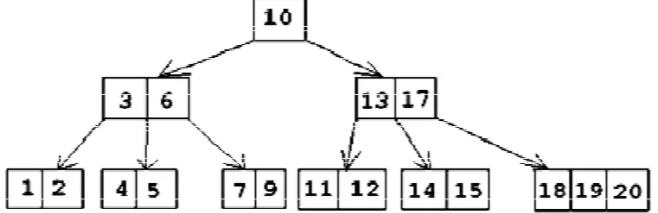
以此类推



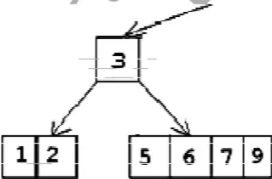
**删除操作**

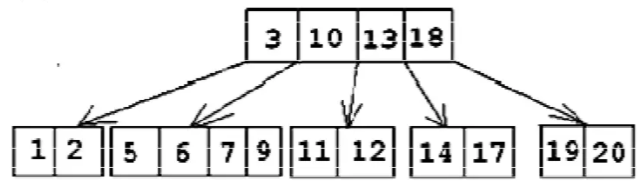
对于B-树关键字的删除，需要找到待删除的关键字，在结点中删除关键字的过程也有可能破坏B-树的特性，如旧关键字的删除可能使得结点中关键字的个数少于规定个数，这是可能需要向其兄弟结点**借关键字**或者和其孩子结点进行**关键字的交换**，也可能需要进行**结点的合并**。

①删除8,16，删除终端节点只需要判断删除后关键字的个数有没有小于最小值即2，没有直接删除，删除16的话，由于不是中端结点，取孩子来交换，只能找小于16的最大的或者大于16最小的，这里如果取15，会导致15所在结点的个数小于2，更麻烦，因此取17替换。



②删除4,

，将其与根节点合并，高度－1



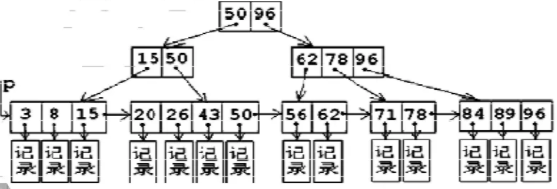
**2.B+树**

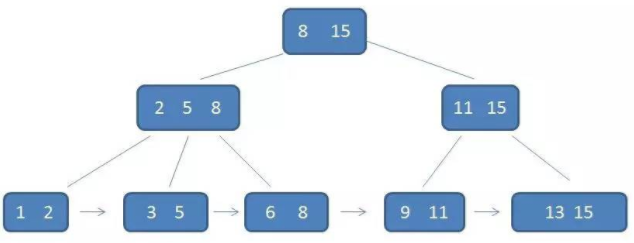
1）在B+树中，具有n个关键字的结点有n个分支，而在B-树中，具有n个关键字的结点含有n+1个关键字。

2）在B+树中，每个结点（除根结点外）中的关键字个数n的取值为ceil(m/2) <= n <=m,根结点的取值范围为1<=n<=m，b树他们的取值范围分别是ceil(m/2) -1<= n <=m-1和1<=n<=m-1。

3）在B+树中叶子结点包含信息，并且包含了全部关键字，叶子结点引出的指针指向记录。

4）在B+树中的所有非叶子结点仅起到一个索引的作用，即结点中的每个索引项只含有对应子树的最大关键字和指向该子树的指针。

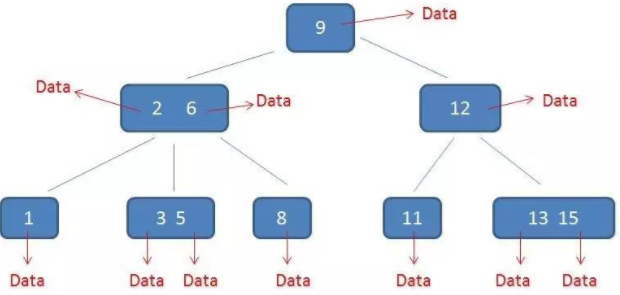




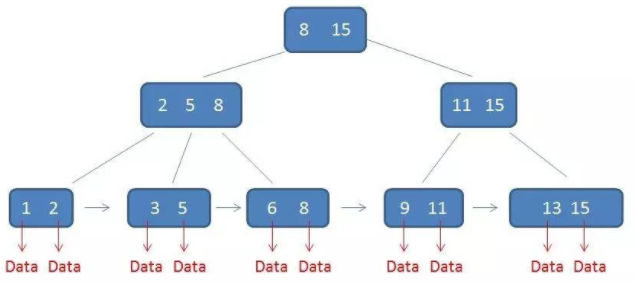
每个父节点都出现在子节点中，是子节点的最大或最小元素。例如根节点的元素8，是子节点2,5,8的最大元素，也是叶子节点6,8的最大元素。根节点15是整个b+数的最大元素，无论插入删除多少元素，始终要保持最大在根节点中；父亲元素都会包含在叶子节点中，所有叶子节点包含了全部元素信息，并且每个叶子节点都带有指向下一个节点的指针，形成链表。

**卫星数据：指的是索引元素所指向的数据记录，比如数据库中的某一行。**

B树中无论是中间结点还是叶子节点，都有卫星数据



而b+树中，只有叶子节点带有卫星数据，其余中间结点仅仅是索引。

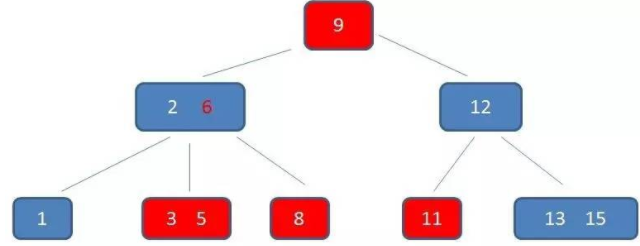


如果需要单独查找元素3，那么先8,15节点，2,5,8节点，3,5节点，看起来与b树没有区别，实际上有两点：b+树没有卫星数据，**同样大小磁盘页，可以容纳更多的元素**，这样B+树会比b树矮胖，另外，**b+树只有在叶子节点有卫星数据，因此必须查找到叶子节点**，而b树只要找到就可以返回。

范围查找:3-11

这种查找树，只要中序遍历，就是有序的

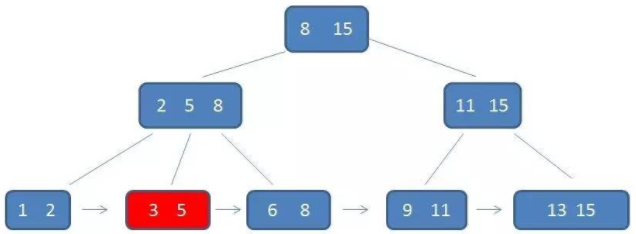
B树范围查找3-11



9->(2,6)->(3,5)查找到3之后进行中序遍历，

(3,5)->(2,6)->(8)->9->11

B+树的查找范围3-11



9->(2,6)->(3,5)查找到3之后进行链表遍历，(3,5)->(6,8)->(9,11)

所以b+树的优势在于；

1.单一节点存储更多的元素，使得查询的IO次数更少。

2.所有查询都要查找到叶子节点，查询性能稳定。

3.所有叶子节点形成有序链表，便于范围查询。