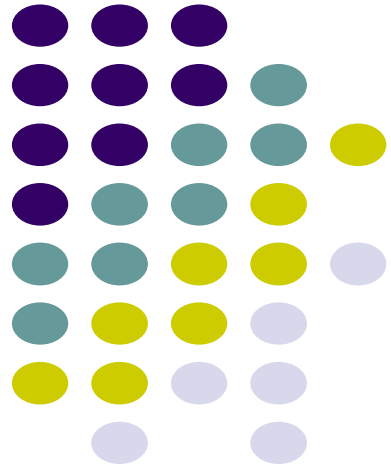
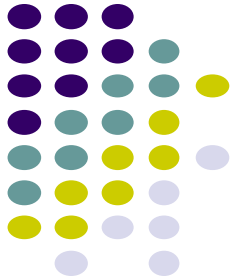
L28：B树

吉林大学计算机学院
谷方明
fmgu2002@sina.com



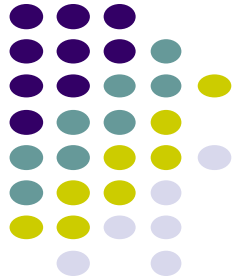
存储系统



- 内存：硅芯片构成。速度快(注：10^8级)，单价高，断电丢失
- 外存：硬盘等。速度慢，单机低，断电保存。

- 存储系统一般采用分级模式：内存+外存
  - ✓ 均摊性能较好：一次读取多个数据，均摊性能10^6/S
  - ✓ 价格低
  - ✓ 聪明的方案

磁盘（机械硬盘）



- 盘片(platter)，主轴(spindle)，磁臂，磁头(head)
- 容量大，单价低，断电保存
- 速度慢（机械运动：磁头移动和盘片旋转），一次存取时间10ms级（5400/7200/15000转每分钟RPM）。
- 读取多个数据 VS 读取1个数据，差别较小:磁头定位；磁道(track)， 扇区(sector)

内查找和外查找

- 内查找：被检索文件能被完整地存放到内存中。
- 外查找：文件很大，内存容纳不下。
- 外查找多次存取磁盘，关键是要尽量减少磁盘读取次数

多叉查找树

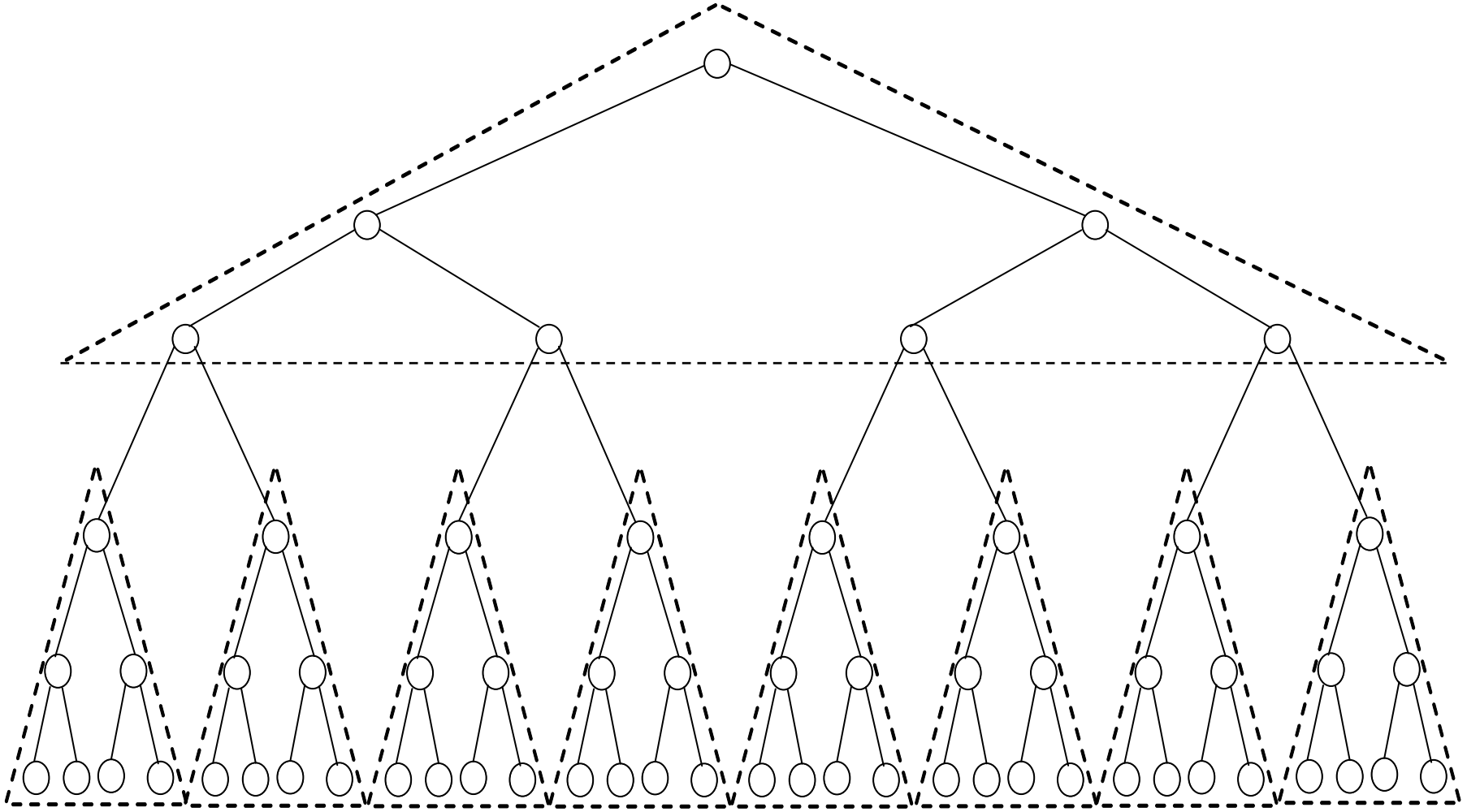


图8.31 二叉检索树形的分页

B-树（B树）

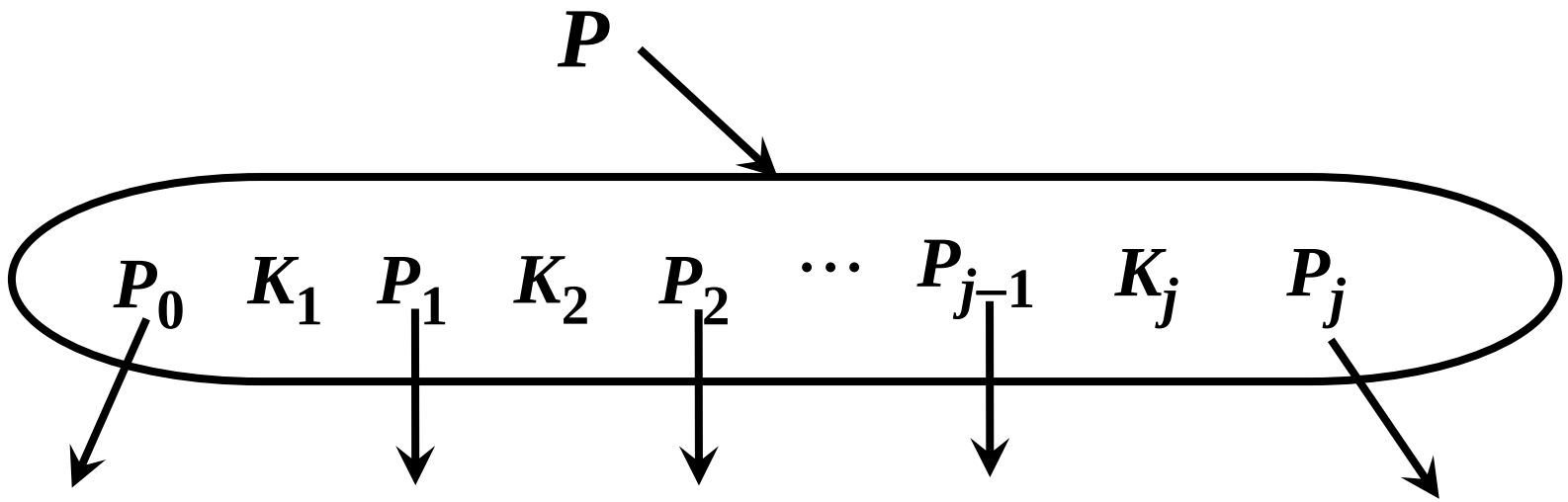
- **m阶B-树**

1. 每个结点最多有 m个儿子；
2. 每个结点最少有$\lceil m/2 \rceil$个儿子，根和叶除外；
3. 根或者是叶，或者至少要有2个儿子；
4. 具有 k 个儿子的非叶结点包含$k-1$个关键词；
5. 所有叶结点都在相同的深度上.

- 注
  - ✓ B-树是一种多叉平衡树；
  - ✓ B-树1970年由**Bayer**和McCreight 提出；

B-树的结点

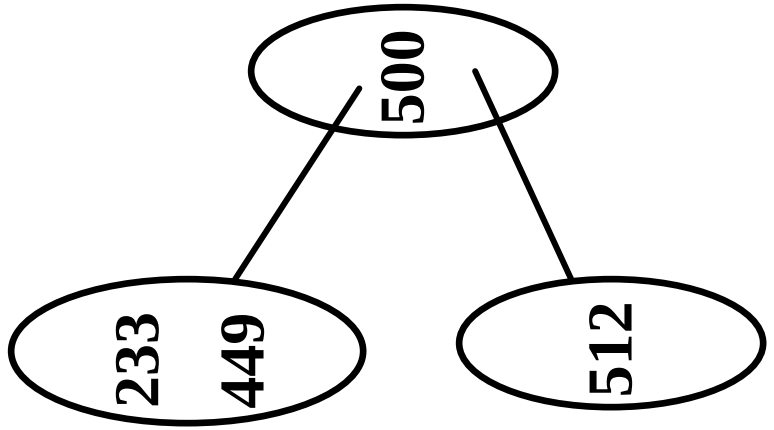
- 结点之关键词满足 $K_1 < K_2 < \ldots < K_j$,指针P_i指向一个其所有关键词都在K_i和 K_{i+1}之间的子树形



B-树的节点

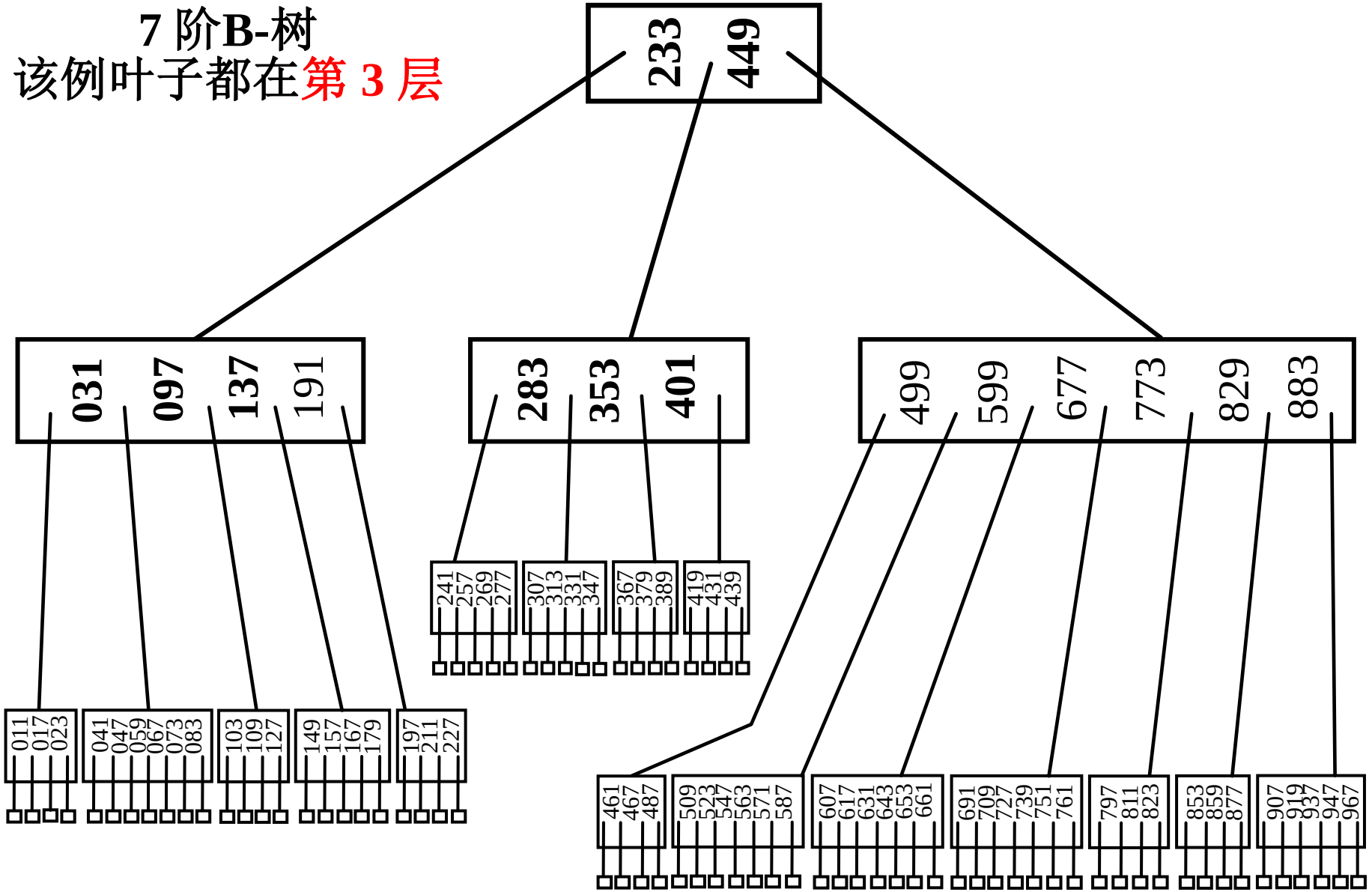
例

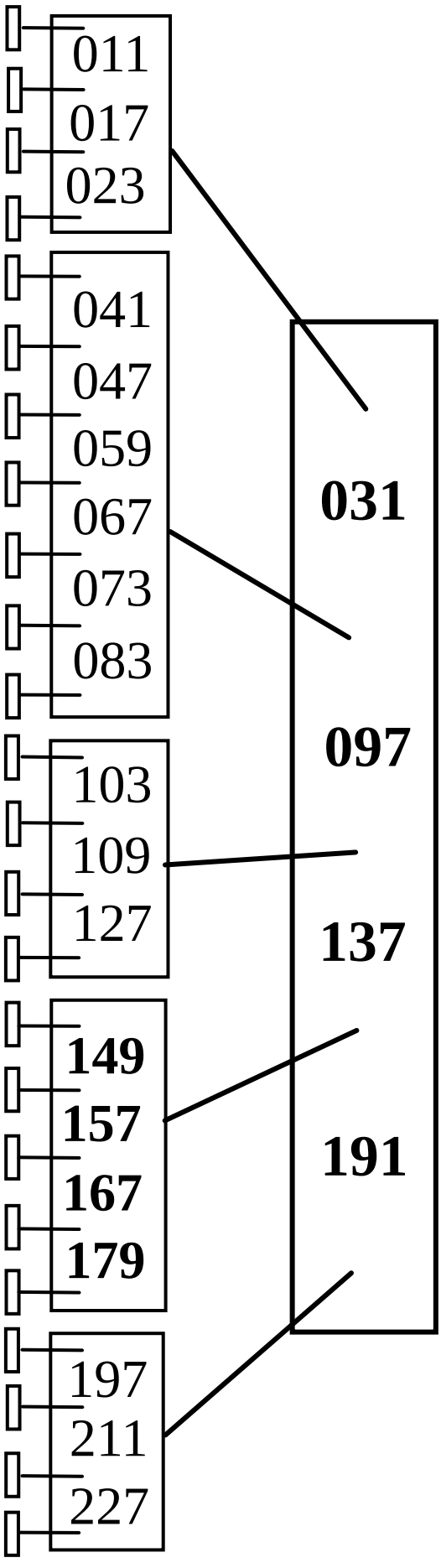
- 3阶B-树 2-3树



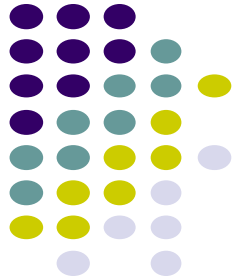
- 4阶B-树 2-4树

7 阶B-树
该例叶子都在第 3 层




011
017
023
041
047
059
067
073
083
103
109
127
149
157
167
179
197
211
227
031
097
137
191

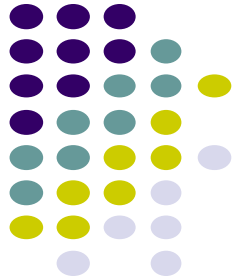
说明



- **结点保存关键字及卫星数据（或记录地址，卫星数据指非关键字的其它信息）**
  - ✓ 关键字分布在所有结点；

- **方块结点要计入高度；**
  - ✓ 方块结点可能是查找失败的位置，也可能是磁盘的链接地址

查找



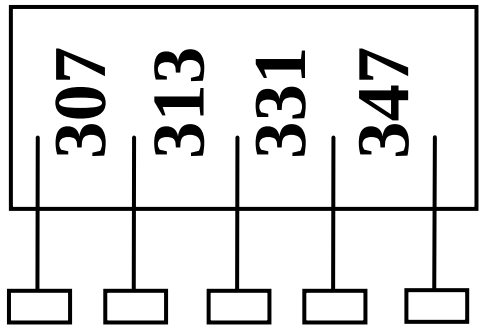
- 算法概述：
  - ✓ $K = K_i$，查找成功结束；
  - ✓ $K < K_1$，转 P_0 继续查找；$K_i < K < K_{i+1}(1 \le i < j)$，转 P_i 继续查找；$K > K_j$，转 P_j 继续查找；
  - ✓ $P = \Lambda$时，查找以失败结束；

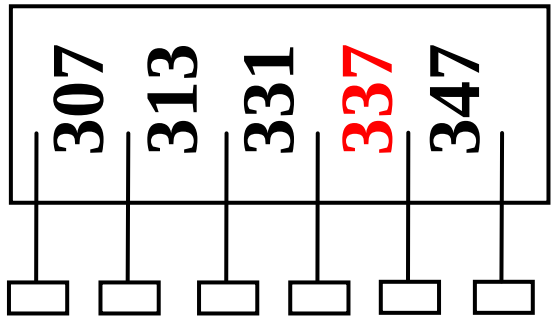
- 结点内查找：当 j 比较大时，可选择对半查找算法；当 j 较小时可采用顺序查找方法。

插入

- 插入发生在最低层的某个结点处（未找到）；

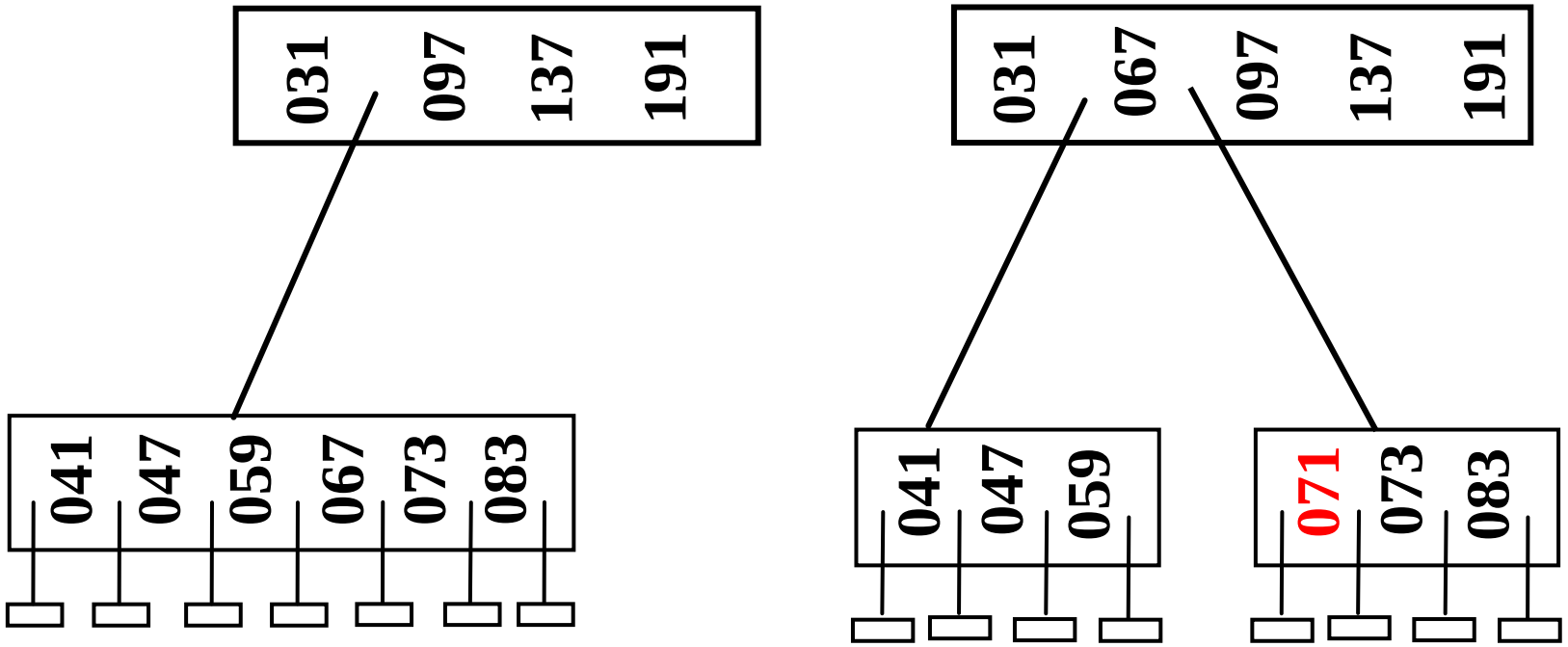
- 插入I：若插入结点的关键词个数 < m-1，把新关键词直接插入该结点中；例：插入337





插入II：分裂

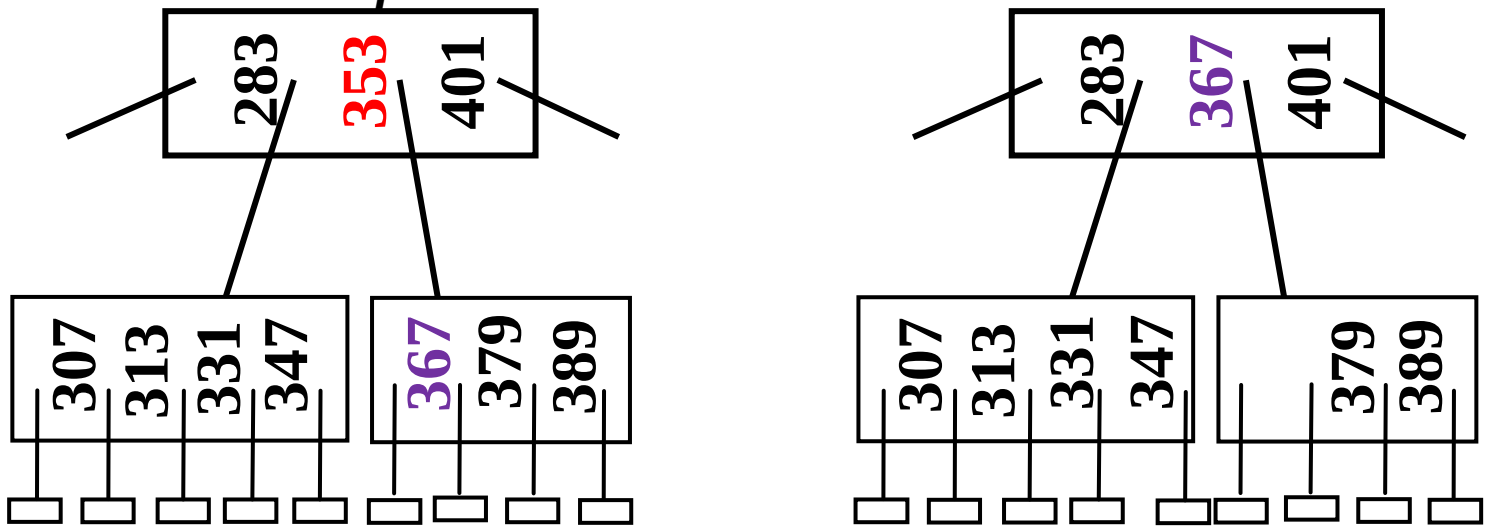
- 若插入结点的关键词个数 = m-1，则插入将造成此结点“分裂”.例：插入071



- **分裂操作：把*m*个关键词分成三组**
  - ✓ 第一组从左向右数，包含$\lceil m/2 \rceil - 1$个关键词；
  - ✓ 第二组从右向左数，包含$\lfloor m/2 \rfloor$个关键词；
  - ✓ 中间的关键词$K_{\lceil m/2 \rceil}$将被提升到上一层结点中，开始新的插入。
- **“分裂”可能会向上传播。在极端情况下，分裂会一直波及到根结点，而这恰恰是B树长高的唯一方式。**

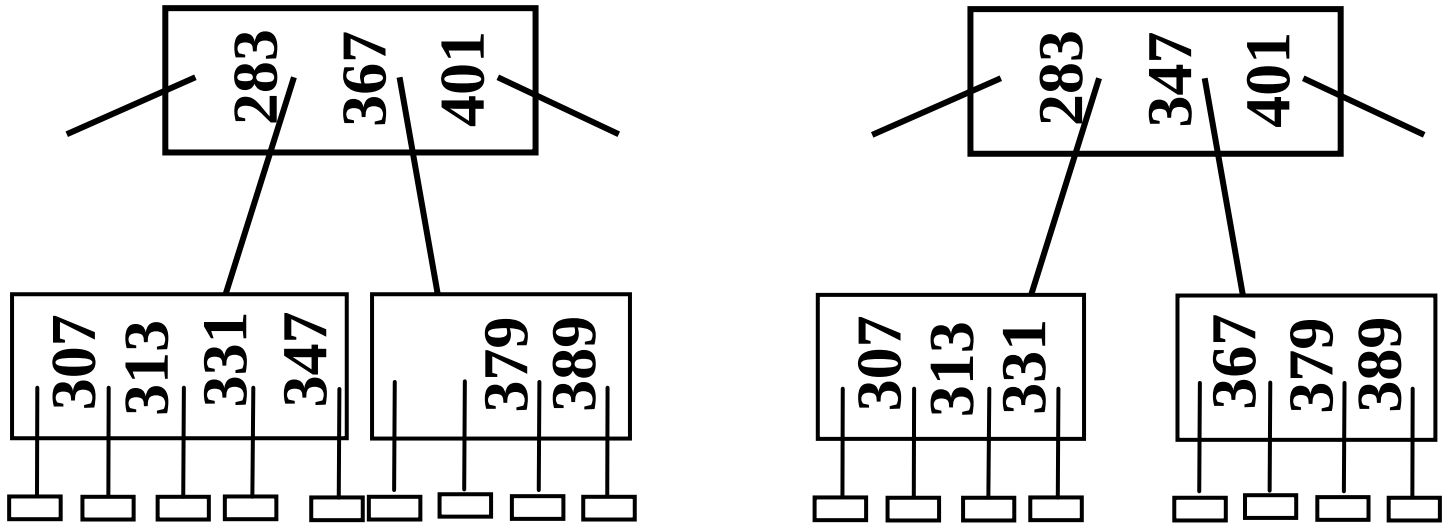
删除

- 先找到要删除的关键词K_i的位置。

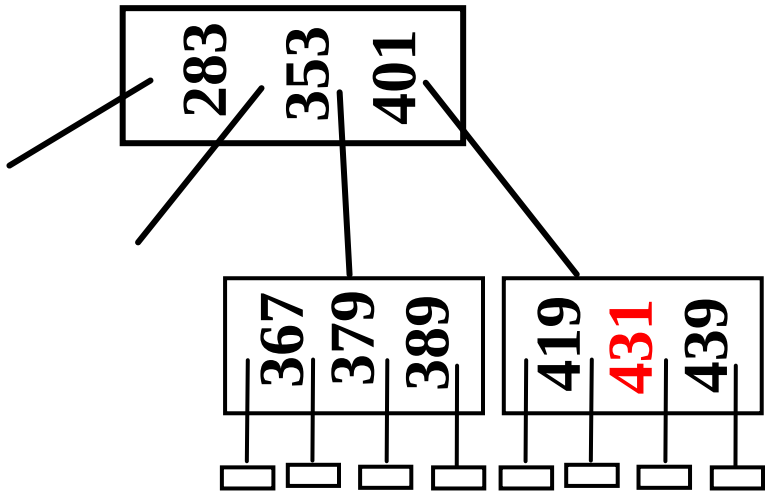


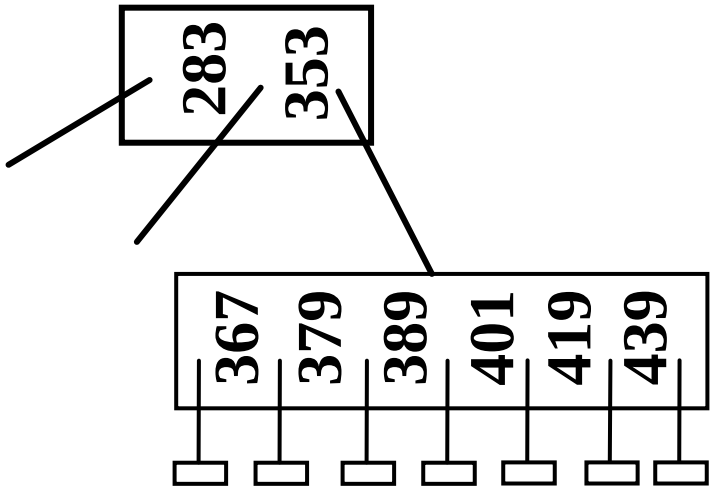
- 第一步：换。若待删除结点不在最低层，则用最接近K_i且大于K_i的关键词K*来交换，然后从K*所在的结点中删除K*。因为中间诸层的关键词还起引导向下查找的作用，不能为空。例：删353

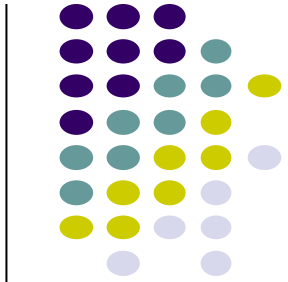
- **第2步：借。当删除关键词后。要检查该结点目前包含的关键词个数是否小于⌈*m*/2⌉ - 1 .**
  - ✓ 若否，则直接删除。
  - ✓ 若是，则要从相邻的兄弟结点中借关键词。最好两个结点的大小一样（关键词数目相同）。



- 第三步：并。若相邻（同父）结点的关键词个数都等于⌈*m*/2⌉–1，则需要“合并”。例：删431
- “合并”操作：即把两个兄弟结点的关键词连同在父结点中指向这两个结点的指针之间的关键词按递增顺序排列在一个新结点中。







- “合并”可能会向上传播。若“合并”操作使上一层结点（父结点）包含的关键词个数 $< \lceil m/2 \rceil - 1$，则又造成上一层结点要借关键词或要进行“合并”操作。“合并”可能会导致上一层发生“合并”，使“合并”不断向上传播，一直波及到根结点，从而使整个B树减少一层。

B-树的高度

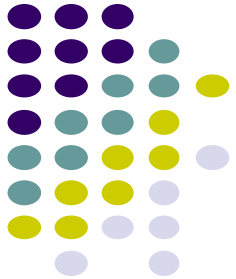
- N个关键词，N+1个叶结点
- 最低: $N+1 \le m^h$, $h \ge \log_m(N+1)$
- 最高

层数	0	1	2	3	…	h
结点数	1	≥2	$\ge 2\lceil m/2 \rceil$	$\ge 2\lceil m/2 \rceil^2$	…	$\ge 2\lceil m/2 \rceil^{h-1}$

$$N+1 \ge 2\lceil m/2 \rceil^{h-1} \qquad h \le 1 + \log_{\lceil m/2 \rceil}\left(\frac{N+1}{2}\right)$$

- $h = \Theta(\log_m n)$

算法分析



- **查找： O ($\log_m N$) * logm = O (logN)**
  - ✓ 结点内二分查找
- **插入：分裂次数O($\log_m N$)，每次最坏涉及O(m)个关键词，整体 O(logN * m/logm)**
- **删除和插入类似；**

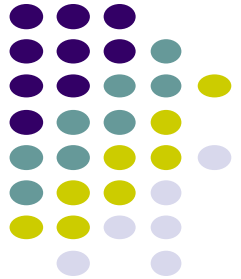
- **B树可作为平衡树的替代（特有平衡机制）**
  - ✓ m不能太大，m=3或4

B树是外查找的自然选择

- 内查找中， B树可作为平衡树的替代
  - ✓ 此时m不能太大，m=3或4

- 多叉树，尤其B树，是外查找的自然选择
  - ✓ m=1001时，高度为2的B树最多存储10亿个关键字，查找最多需要2次磁盘访问。
  - ✓ 磁盘页的大小限制了B树的阶数。
  - ✓ 通常B树结点大小=磁盘页的大小(page=4/8kb)

合理的m（教材的例子）

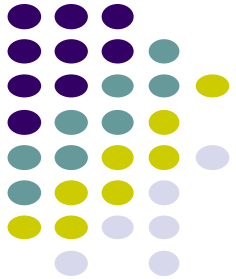


- 设读进一个包含***m***个分枝的页结点要用**72.5 + 0.05*m***毫秒的时间（磁头定位+结点读取）。
- 每个页结点在内存中处理需***a* + b log*m***毫秒(二分)
- 一般***a* << 72.5**，可忽略。从而，每个记录**ASL**

$$
\begin{aligned}
t_a &\approx \log_m N * (72.5 + 0.05m + b\log_2 m) \\
&= \frac{\log_2 N * (72.5 + 0.05m + b\log_2 m)}{\log_2 m} \\
&= \log_2 N \ [(72.5 + 0.05m)/\log_2 m + b]
\end{aligned}
$$

- 在***m* ≈350**时，***ta***取最小值，当**200 < *m* < 500** 时，***ta***取近似最佳值。
  - ✓ m=256, ta = (72.5+12.8) /8 ≈ 9.48；m=512, ta=10.9, m=1024, ta = 12.37

B-树的改进



- B树的结点中，除了关键字和指针外，还保存了记录的其它数据或记录的地址，占用了空间，限制了m。

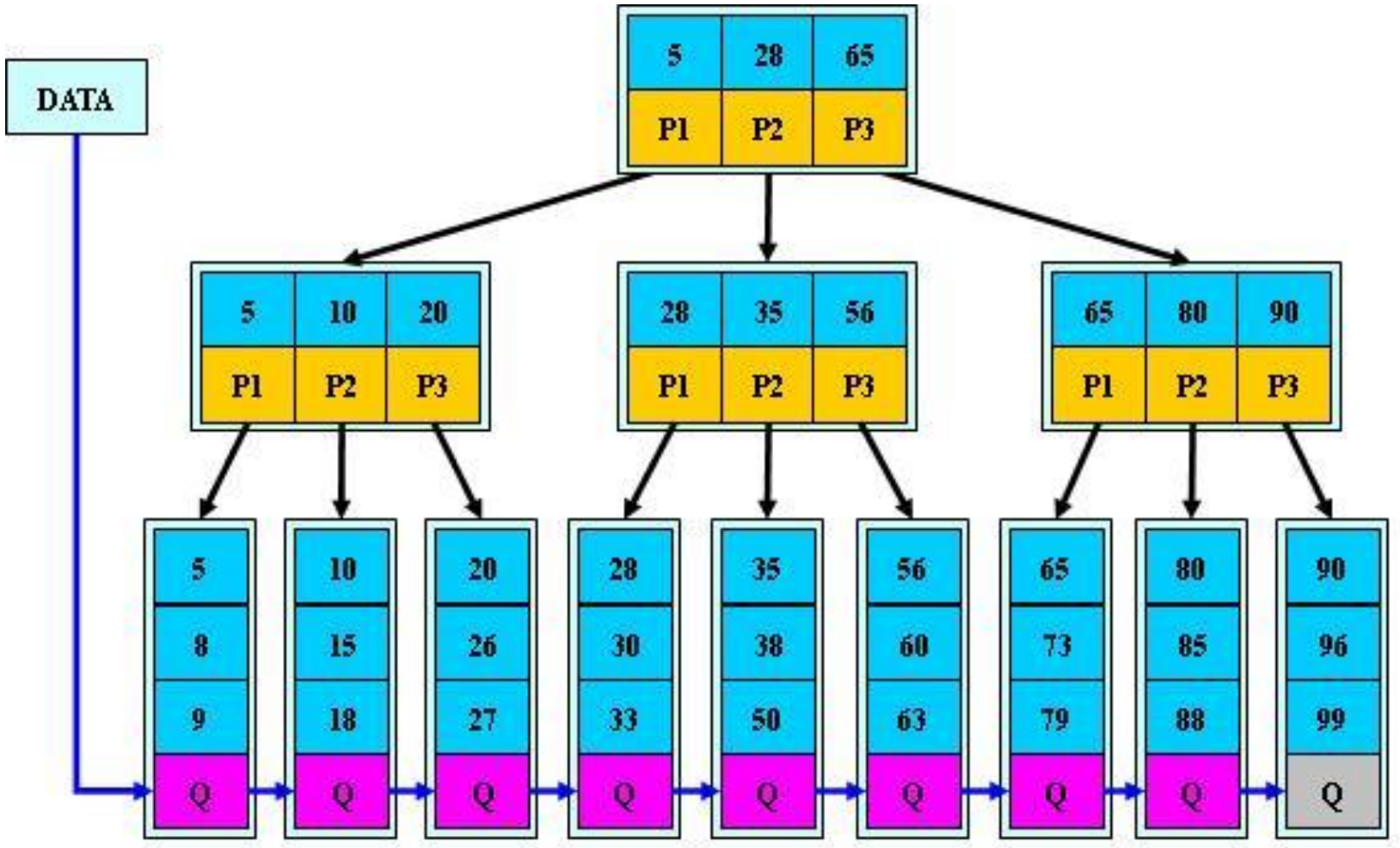
- 特殊操作的需要。例如，以递增序访问B树的所有结点。
  - ✓ 用中序遍历，但对非叶结点，一次只能显示一个关键词，然后就要访问其他页。
  - ✓ 增加磁盘读取次数；也会引起内存颠簸。

B+树定义

- m阶B+ 树：

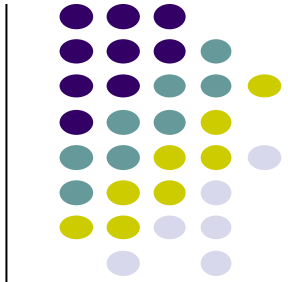
1. 每个结点最多有 *m*个儿子；
2. 每个结点最少有⌈*m*/2⌉个儿子，根和叶除外；
3. 根或者是叶，或者至少要有2个儿子；
4. 有*n*棵子树的结点有*n*个关键词；
5. 叶子结点包含全部关键词及指向相应记录的指针，并且叶子结点按关键词自小而大的顺序链接；
6. 分支结点（索引块）仅包含各个子结点（下级索引块）中最大/最小关键词及指向子结点的指针。

B+树示例


DATA
5
28
65
P1
P2
P3
5
10
20
P1
P2
P3
28
35
56
P1
P2
P3
65
80
90
P1
P2
P3
5
8
9
Q
10
15
18
Q
20
26
27
Q
28
30
33
Q
35
38
50
Q
56
60
63
Q
65
73
79
Q
80
85
88
Q
90
96
99
Q

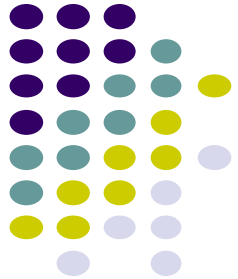
B+树与B树的区别

- B+树中，有*n*棵子树的结点含有*n*个关键词；B树中，有*n*棵子树的结点含有*n-1*个关键词；
- B+树中，叶结点中包含了全部关键词，其它结点中的关键词都包含在叶结点中；而在B树中，叶子结点包含的关键词与其它结点包含的关键词是不重复的；
- B+树中，非叶结点仅起索引作用，即结点中每个索引项只含有对应子树的最大关键词和指向该子树的指针，不含有该关键词对应记录的存储地址。而B树中，每个关键词对应一个记录的存储地址；



- B+树在随机查找时，如果非叶结点上的关键词等于给定值，并不终止，而是继续向下直到叶结点。因此，在B+树中，不管查找成功与否，每次查找都是走了一条从根到叶结点的路径。
- B+树上有两个头指针，一个指向根，另一个指向最小关键词的叶子结点，
- B+ 树进行两种查找运算：一种是从最小关键词开始进行顺序查找，另一种是从根结点开始进行随机查找。

B+树比B树更适合作索引



- B+树的磁盘读写代价更低。 B+树的分支结点没有指向记录的指针，分支结点(硬盘页)能容纳更多的关键字。相对来说读写次数也就降低了。
- B+树方便扫库，B树必须用中序遍历的方法按序扫库，而B+树直接从叶子结点挨个扫一遍。
- B+树易于支持Range-Query（顺着链块找），而B树则很麻烦。这是数据库选用B+树的主要原因。

B*树：B+树的变体

- B*树在B+ 树非根和非叶结点再增加指向的兄弟指针；
- B*树定义了非叶子结点关键字个数至少为(2/3)*m，即块的最低使用率为2/3（代替B树的1/2）。