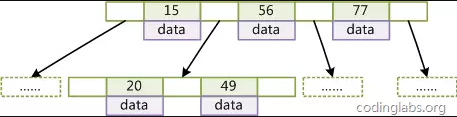
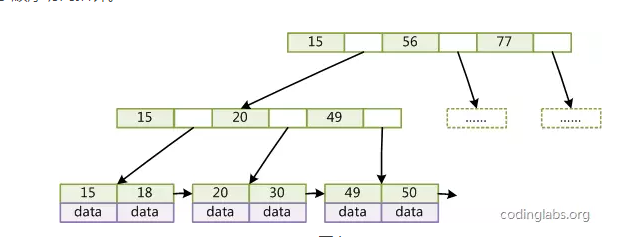
B-Tree,每个节点都有data



B+tree。只有叶子节点有data

叶子节点之间有指针连接



与B-Tree相比，B+Tree有以下不同点：、

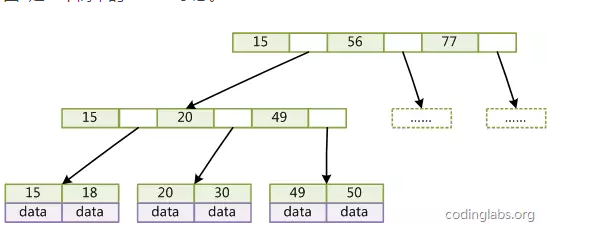
1. 每个节点的指针上限为2d而不是2d+1。

2. 内节点不存储data，只存储key；叶子节点不存储指针。

3增加了顺序访问指针。

在B+Tree的每个叶子节点增加一个指向相邻叶子节点的指针，就形成了带有顺序访问指针的B+Tree。做这个优化的目的是为**了提高区间访问**的性能，例如图4中如果要查询key为从18到49的所有数据记录，当找到18后，只需顺着节点和指针顺序遍历就可以一次性访问到所有数据节点，极大提到了区间查询效率。

（二级索引叶子节点存储的是**主键值**，故查询主键，即使索引里没有也可以覆盖）



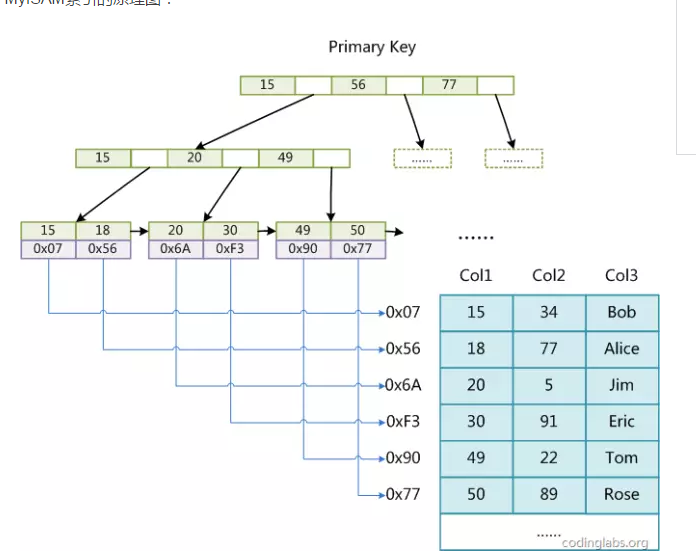
B+Tree更适合外存索引，原因和内节点出度d有关。从上面分析可以看到，d越大索引的性能越好，而出度的上限取决于节点内key和data的大小：

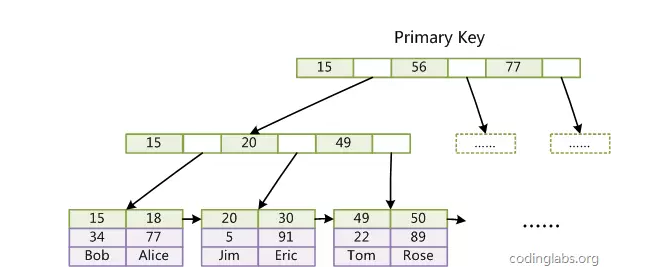
dmax = floor(pagesize / (keysize + datasize + pointsize)) (pagesize – dmax >= pointsize) 或 dmax = floor(pagesize / (keysize + datasize + pointsize)) – 1 (pagesize – dmax < pointsize) floor表示向下取整。由于B+Tree内节点去掉了data域，因此可以拥有更大的出度，拥有更好的性能。

MyIsam没有聚簇索引，索引主键索引和二级索引本质一样。

Myisam的叶子节点data指向**磁盘的位置**，（数据在磁盘中顺序存储）

Innodb的叶子节点data存储的是主键的值。





理解查询使用索引的最好的方式是通过递归去理解。

当存在某key（a,b,c）;

查询 a=n1;相当于使用了一个key(a)

a=n1 and b = n2;相当于在a=n1的查询条件下限定了一个子树，然后在子树上使用了一个key(b),同理如此递推

当 a =n1 and b=n2 and c>n3时 相当于在a =n1 and b=n2查询子树上使用了一个key(c)

而使用一个索引来做范围查询时的操作并非从根节点开始查询，而是同叶子结点（以key(c) 形式排序的）开始顺序查找