<https://www.cs.usfca.edu/~galles/visualization/Algorithms.html> 树

1、索引存储在磁盘上，然后加载到内存，磁盘往往不是严格按需读取，而是每次都会预读，而且主存和磁盘以页为单位交换数据，所以在读取的数据不在主存中时，会从磁盘中读取一批数据（页）到主存中。

2、不管在哪种程序优化上，要想快速挺高性能，直接将常用的、少变更的数据直接读取到内存中，使用的时候就直接在内存上读取，而不去磁盘上读取，减少I/O操作，这样就能使程序快上10倍以上。但由于内存容量的限制，也不可能将所有的数据都放内存中。

索引数据结构：

二叉树，红黑树，Hash表，B-Tree

[二叉树](https://baike.baidu.com/item/%E4%BA%8C%E5%8F%89%E6%A0%91/1602879" \t "https://baike.baidu.com/item/%E4%BA%8C%E5%8F%89%E6%90%9C%E7%B4%A2%E6%A0%91/_blank)： 若它的左子树不空，则左子树上所有结点的值均小于它的[根结点](https://baike.baidu.com/item/%E6%A0%B9%E7%BB%93%E7%82%B9/9795570" \t "https://baike.baidu.com/item/%E4%BA%8C%E5%8F%89%E6%90%9C%E7%B4%A2%E6%A0%91/_blank)的值； 若它的右子树不空，则右子树上所有结点的值均大于它的根结点的值； 它的左、右子树也分别为[二叉排序树](https://baike.baidu.com/item/%E4%BA%8C%E5%8F%89%E6%8E%92%E5%BA%8F%E6%A0%91/10905079" \t "https://baike.baidu.com/item/%E4%BA%8C%E5%8F%89%E6%90%9C%E7%B4%A2%E6%A0%91/_blank)。

3.非叶子结点的左指针指向小于其关键字的子树，右指针指向大于其关键字的子树；

二叉树的遍历主要有三种：

（1）先(根)序遍历（根左右） 第一次经过

（2）中(根)序遍历（左根右） 第二次经过

（3）后(根)序遍历（左右根） 第三次经过 或者死胡同

红黑树（二叉平衡树），不仅是一个二叉搜索树，而且必须满足一下规则：  
     1、每个节点不是红色就是黑色。  
     2、根节点为黑色。  
     3、如果节点为红色，其子节点必须为黑色。  
     4、任意一个节点到到NULL（树尾端）的任何路径，所含之黑色节点数必须相同。

红黑树不支持数据量大的查找，数据量多的时候，红黑树的高度就很高，就需要进行很多次I/O（mysql的IO是指数据库文件的读写，也就是检索数据和插入数据。）查找，性能降低

Hash表 通过算法算出的值查找到这个值对应的行，映射到所有行，但是不能范围查询，用的比较少。单条sql查询的话，hash查询速度就很快。不支持范围查找。也叫散列算法，就是把任意值(key)通过哈希函数变换为固定长度的 key 地址，通过这个地址进行具体数据的数据结构。Hash算法也可能对不同key计算出同一个结果，这叫哈希碰撞

B树和B+树：

1、B+树的非叶子节点只保存索引(key)，不保存行记录(数据)。而B树叶子节点和非叶子节点都存储索引和

行记录。Mysql官方定义一张表有规定的大小，这样的话，同样的空间，B+树能存储更多的索引。这就意味着同样的数据，B+树比B树更”矮胖”，减少IO次数。

1. 查询额时候，由于B树子节点和非叶子节点都存在行几录，也就是说，B树查询其实是不稳定的的(好的时候，只查根节点，坏的时候，查到叶子节点)。而B+树查询最终到叶子节点，查询时候稳定
2. B树查询只能通过中序遍历查询来定位最小和最大值，而B+树通过链表就能实现，查询更方便

B+Tree与B-Tree的差异在于：  
1、B+Tree非叶子节点不存储data，只存储key；  
2、所有的关键字全部存储在叶子节点上；  
3、每个叶子节点含有一个指向相邻叶子节点的指针，带顺序访问指针的B+树提高了区间查找能力；  
4、非叶子节点可以看成索引部分，节点中仅含有其子树（根节点）中的最大（或最小）关键字；

#### ****B树的插入与删除****

#### （1）插入的步骤

1、插入一个元素时，首先在B树中是否存在，如果不存在，即在叶子结点处结束，然后在叶子结点中插入该新的元素

2、如果叶子结点空间足够，这里需要向右移动该叶子结点中大于新插入关键字的元素，如果空间满了以致没有足够的空间去添加新的元素，则将该结点进行“分裂”，将一半数量的关键字元素分裂到新的其相邻右结点中，中间关键字元素上移到父结点中（当然，如果父结点空间满了，也同样需要“分裂”操作）

3、当结点中关键元素向右移动了，相关的指针也需要向右移。如果在根结点插入新元素，空间满了，则进行分裂操作，这样原来的根结点中的中间关键字元素向上移动到新的根结点中，因此导致树的高度增加一层

Mysql的MyISAM和InnoDB两个存储引擎的索引实现方式：

MyISAM引擎使用B+ Tree作为索引结构，叶节点存放的是数据记录的地址。

MyISAM引擎的辅助索引（二级索引）和主索引在结构上没有区别，只是辅助索引的key可以重复，叶节点上存放的也是数据记录的地址。

MyISAM索引文件(myi)和数据文件(myd)是分离的，索引文件仅保存数据记录的地址。

InnoDB(ibd)中表数据本身就是按B+ Tree组织的一个索引结构，叶节点存放的就不是数据记录的地址，而是完整的数据记录。所以InnoDB这种存储方式，又称为聚集索引，使得按主键的搜索十分高效，但二级索引搜索需要检索两遍索引：首先二级索引获得主键，然后用主键到主索引中检索到数据记录。因为主键是InnoDB表记录的”逻辑地址“，所以InnoDB要求表必须有主键，MyISAM可以没有。

索引最终选择B+树的原因:  
1. hash很快，但每次IO只能取一个数  
2. AVL和红黑树，在大量数据的情况下，IO操作还是太多  
3. B树每个节点内存储的是数据，因此每个节点存储的分支太少  
4.B+节点存储的是索引+指针(引用指向下一个节点)，可以存储大量索引，同时最终数据存储在叶子节点，并且有引用横向链接，可以在2-3次的IO操作内完成千万级别的表操作。  
5. 建议索引是是自增长数字，这样适合范围查找



