1、树的定义

树是一种数据结构，它是由n(n>=1)个有限结点组成一个具有层次关系的集合。把它叫做“树”是因为它看起来像一颗倒挂的树，也就是说它是根朝上，而叶朝下的。它具有以下特点：

A、每个结点有零个或多个子节点。

B、没有父结点的结点称为根结点。

C、每一个非根结点有且只有一个父结点。

D、除了根结点外，每个子节点可以分为多个不相交的子树。

2、为什么需要树？

因为它结合了另外两种数据结构的有点：一是有序数组，一是链表。在书中查找数据项的速度和在有序数组中查找一样的快。并且插入数据项和删除数据项和链表一样的快。

目前接触来看，有序数组查询和更新相对较快，对于插入和删除，由于会导致扩容，所以相对应的链表会更快。

有序数组：查询和更新快

链表：插入和删除快

树：结合了两者的有点，查询、更新、插入、删除都很快。

3、数的种类有哪些？

无序树：树中任意节点的子节点之间没有顺序关系，这种树称为无序树，也成为自由树。

有序树：树中任意节点的子节点之间有顺序关系，这种树称为有序树；

二叉树：每个节点最多含有两个子树的树称为二叉树；

完全二叉树：对于一颗二叉树，假设其深度为d(d>1)。除了第d层外，其他各层的节点数目均已达到最大值，且第d层所有节点从左向右连续地紧密排列，这样的二叉树被称为完全二叉树；

满二叉树 ：所有叶节点都在最底层的完全二叉树；

平衡二叉树（AVL树）：当且仅当任何节点的两颗子树的高度不大于1的二叉树。

排序二叉树：也称二叉搜索树、有序二叉树；

霍夫曼树：带权路径最短的二叉树称为哈夫曼树或最优二叉树；

B树(B-tree)：一种对读写操作进行优化的自平衡的二叉查找树，能够保持数据有序，拥有多余两个子树。

B+树：B+树通常用于数据库和操作系统的文件系统中。B+树的特点是能够保持数据稳定有序，其插入与修改拥有比较稳定的对数时间复杂度。B+树元素自底向上插入。

红黑树：是一种自平衡二叉查找树，是在计算机科学中用到的一种数据结构，典型的用途是实现关联数组。

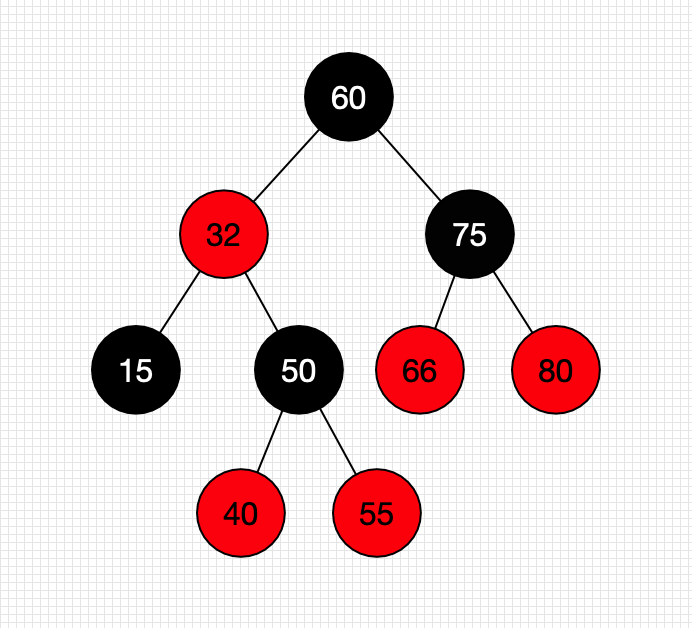
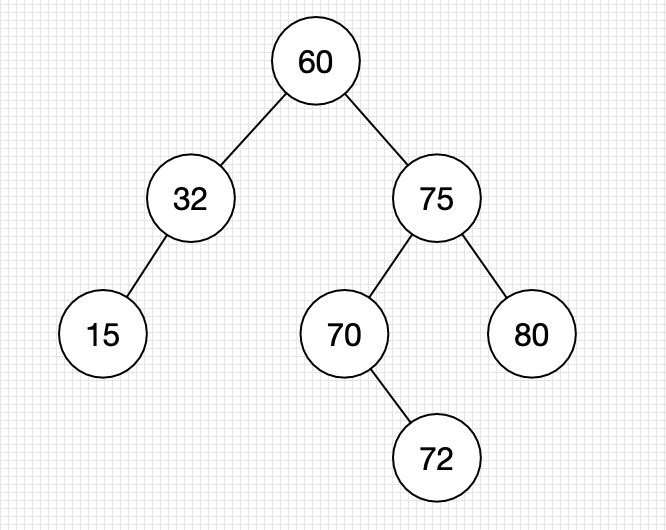
对于几种常见的树进行介绍：

1、二叉搜索树的特点：

A、每个节点包含一个值，每个节点至多有两个子树。

B、每个节点左子树节点的值都小于自身的值，每个节点右子树节点的值都大于自身的值。

C、二叉搜索树的查询时间复杂度是 log(N)，但是随着不断的插入、删除节点，二叉树的树高可能会不断变大，当一个二叉搜索树所有节点都只有左子树或者都只有右子树时，其查找性能就退化成线性的了。



二叉树 红黑树

2、平衡二叉树的特点：

平衡二叉树又被称为AVL树，这个方案很好的解决了二叉查找树退化成链表的问题，平衡二叉树保证每个节点左右子树的高度差的绝对值不超过 1。AVL 树是严格的平衡二叉树，插入或删除数据时可能经常需要旋转来保持平衡，时间复杂度最好情况和最坏情况都维持在O(logN)。但是频繁旋转会使插入和删除牺牲掉O(logN)左右的时间，不过相对二叉查找树来说，时间上稳定了很多。

3、红黑树的特点：

红黑树是另一种更加实用的非严格的平衡二叉树。红黑树更关注局部平衡而非整体平衡，确保没有一条路径会比其他路径长2倍，所以是接近平衡的，但减少了许多不必要的旋转操作，更加实用。在java8中HashMap和ConcurrentHashMap中就应用了红黑树来解决散列冲突时的查找问题。同时TreeMap也是通过红黑树来保证有序性的。

红黑树除了拥有二叉搜索树的特点外，还有以下规则：

A、每个节点不是红色就是黑色。

B、根节点是黑色。

C、每个叶子节点都是黑色的空节点，如图中的黑色三角。

D、红色节点的两个子节点都是黑色的。

E、任意节点到其叶节点的每条路径上，包含相同数量的黑色节点。

4、B-Tree的特点：

B树是一种多叉树，也叫多路搜索树。B树中每个节点可以存储多个元素，非常适合用在文件索引上，可以有效减少磁盘IO次数。B树中所有节点的最大子节点数称为B树的阶，如图所示是一棵3阶B树，也叫2-3树。

一个m阶树有如下特点：

1、非叶子节点最多有m棵子树。

2、根节点最少有两个子树，非根、非叶节点最少有m/2棵子树。

3、非叶子节点中保存的关键字个数，等于该节点子树子个数-1，就是说一个节点如果有3棵子树，那么其中必定包含2个关键字。

4、非叶子节点中的关键字大小有序

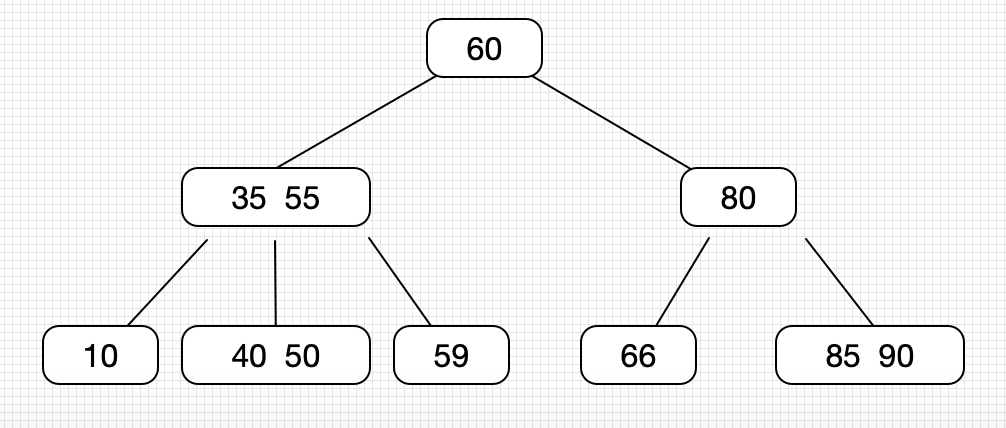
5、节点中每个关键字的左子树中的关键字都小于该关键字，右子树中的关键字都大于该管家你在。

6、所有叶节点都在同一层。

7、B树在查找时，从根节点开始，对节点内的有序的关键字序列进行二分查找，如果找到就结束，没有找到就进入查询关键字所属所属范围的子树进行查找，直到找到叶节点。

总结：

B树的关键字分布在整棵树中，一个关键字只出现在一个节点中，搜索可能在非叶节点停止；B树一般应用在文件系统中。



5、B+树的特点：

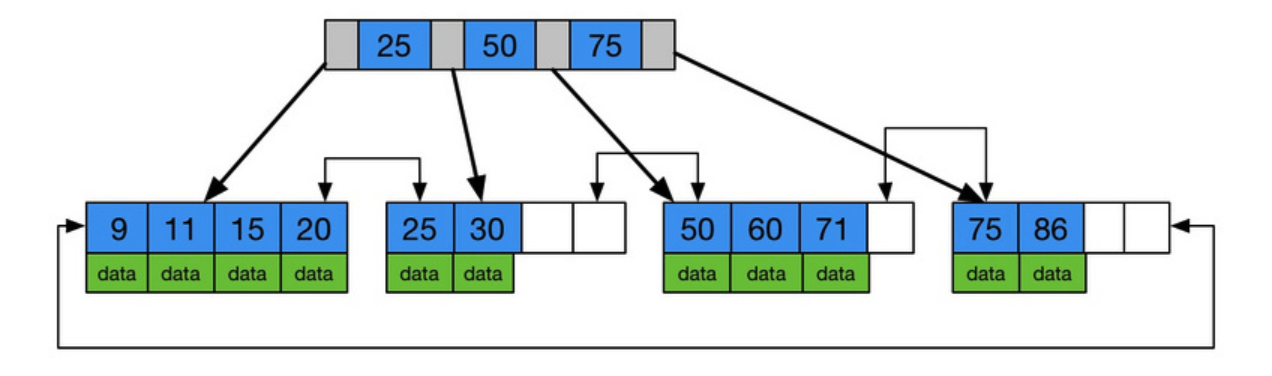
1、节点中的关键字与子树数目相同，比如节点中有 3 个关键字，那么就有 3 棵子树；

2、关键字对应的子树中的节点都大于或等于关键字，子树中包括关键字自身；

3、所有关键字都出现在叶子节点中；

4、所有叶子节点都有指向下一个叶子节点的指针。

与 B 树不同，B+ 树在搜索时不会在非叶子节点命中，一定会查询到叶子节点；另外一个，叶子节点相当于数据存储层，保存关键字对应的数据，而非叶子节点只保存关键字和指向叶节点的指针，不保存关键字对应的数据，所以同样数量关键字的非叶节点，B+ 树比 B 树要小很多。



问题：为什么B+ 树更适合索引系统，在MySQL 数据库的索引就提供了 B+ 树实现。原因有三个：

1、由于叶节点之间有指针相连，B+ 树更适合范围检索；

2、存储引擎的设计巧妙的利用了外存（磁盘）的存储结构，即磁盘的最小存储单位是扇区（sector），而操作系统的块（block）通常是整数倍的sector，操作系统以页（page）为单位管理内存，一页（page）通常默认为4K，数据库的页通常设置为操作系统页的整数倍，因此索引结构的节点被设计为一个页的大小，然后利用外存的“预读取”原则，每次读取的时候，把整个节点的数据读取到内存中，然后在内存中查找，已知内存的读取速度是外存读取I/O速度的几百倍，那么提升查找速度的关键就在于尽可能少的磁盘I/O，那么可以知道，每个节点中的key个数越多，那么树的高度越小，需要I/O的次数越少，因此一般来说B+Tree比BTree更快，因为B+Tree的非叶节点中不存储data，就可以存储更多的key。

3、B+ 树的查询效率比较稳定。任何关键字的查找必须走一条从根结点到叶子结点的路，所有关键字查询的路径长度相同，效率相当。