# AVL树，红黑树，B树，B+树，Trie树都分别应用在哪些现实场景中

1、AVL树和红黑树简单来说都是用来搜索的呗。  
 AVL树：平衡二叉树，一般是用平衡因子差值决定并通过旋转来实现，左右子树树高差不超过1，那么和红黑树比较它是严格的平衡二叉树，平衡条件非常严格（树高差只有1），只要插入或删除不满足上面的条件就要通过旋转来保持平衡。由于旋转是非常耗费时间的。我们可以推测AVL树适合用于插入删除次数比较少，但查找多的情况。

红黑树：平衡二叉树，通过对任何一条从根到叶子的简单路径上各个节点的颜色进行约束，确保没有一条路径会比其他路径长2倍，因而是近似平衡的。所以相对于严格要求平衡的AVL树来说，它的旋转保持平衡次数较少。用于搜索时，插入删除次数多的情况下我们就用红黑树来取代AVL。  
（现在部分场景使用跳表来替换红黑树，可搜索“为啥 redis 使用跳表(skiplist)而不是使用 red-black？”）

1. B树，B+树：它们特点是一样的，是多路查找树（不一定是二叉树）

一般用于数据库系统中，为什么，因为它们分支多层数少呗，都知道磁盘IO是非常耗时的，而像大量数据存储在磁盘中所以我们要有效的减少磁盘IO次数避免磁盘频繁的查找。  
 B+树是B树的变种树，有n棵子树的节点中含有n个关键字，每个关键字不保存数据，只用来索引，数据都保存在叶子节点。是为文件系统而生的。

1. Trie树：又名单词查找树，一种树形结构，常用来操作字符串，用于字符串的查询。也可以用于基于词典的敏感词检测。

它是不同字符串的相同前缀只保存一份。相对直接保存字符串肯定是节省空间的，但是它保存大量字符串时会很耗费内存（是内存）。类似的有前缀树(prefix tree)，后缀树(suffix tree)，radix tree(patricia tree, **compact prefix tree**)，crit-bit tree（解决耗费内存问题），以及前面说的double array trie。简单的补充下我了解应用前缀树：字符串快速检索，字符串排序，最长公共前缀，自动匹配前缀显示后缀。  
后缀树：查找字符串s1在s2中，字符串s1在s2中出现的次数，字符串s1,s2最长公共部分，最长回文串。