1 主码:每一条记录的唯一标识，辅码:可能出现重复值的码 <- 多数索引利用辅码索引
2 索引:关键码和记录位置关联起来，键值对，指针指向主文件的完整记录 -> 索引文件
3 主文件/索引文件，稠密索引:主文件不需要排序，稀疏索引:主文件必须排序，分块索引
4 线性索引: 索引文件的指针指向主码的起始位置
5 可以对边长的记录访问，支持二分检索，可以使用二级线性索引，二级->一级->主文件(四级页表)
6 静态索引:文件创建时生成，系统运行过程中索引结构不改变，再组织时才能改变索引结构
7 倒排索引: 按属性值建立索引，倒排文件，倒排表(属性值,记录指针)
8 支持基于属性的高效检索，降低更新的效率
9 正文文件的倒排，词索引(抽取关键词)/全文索引(对正文中的每一个字符串建立索引，人工/自动抽取关键词，停用词，抽词干，切词，建立关键词的索引，更进一步，建立对关键词的索引(trie, hash))
10 高效检索，检索词有限，空间代价很高
11 m 阶 B 树: 或者为空，或者,每个节点至多有 m 个子节点，根节点至少有两棵子树，非叶节点至少有[m / 2](向上取整)棵子树，叶节点都位于同一层.
12 树高平衡，叶节点同层，关键码不重复，父节点关键码是子节点的分界，局部性原理
13 检索长度:最多 h + 1 次读盘(最终访问主文件)
14 插入过程: 找到最底层插入，如果溢出，结点分裂，关键码插入父节点，迭代
15 分裂方法: 前[m / 2] - 1 个结点在左, [m / 2] + 1 开始的结点在右，中间的结点上升
16 访外次数: 约定:内存足够大，向上分裂时，不会再次读入同一个结点，最少写盘次数 1，考虑修改主文件,再加一次，最多访外次数: 3h + 1
17 删除: 不在叶节点层，把它和后继交换，再删除它，删除后若有溢出，向兄弟节点解关键码，借不到，合并(此时必不会溢出)
18 B+树: 在叶节点上存储信息的树，所有关键码都在叶节点，每层的关键码都是下一层相应结点的复写
19 每个结点至少有[m / 2]个子节点，至多有 m 个子节点，根节点至少有 2 个子节点，有 k 个子节点的结点必有 k 个关键码
20 非叶结点可以看成高层索引，查找到叶结点层才停止(否则不知道指针值)，叶节点一般形成双链表
21 插入分裂，类似 B 树，分成左[m / 2](上)和右[m / 2](下)
22 删除结点，也和 B 树类似，上层的副本可以保留，也可以作为分界的关键码
23 包含 N 个关键码的 B 树，有 N+1 个外部空指针，第 k 层至少 $2 * [m / 2]^{k-1}$ 个结点
24 存取次数 $k <= 1 + \log_{[m / 2]} (n + 1) / 2$
25 结点分裂次数 p - 1 / n - 1(除第一个结点外，所有结点都是分裂来的)，又 n >= 1 + ([m / 2] - 1) *(p - 1)，有 s = p - 1 / n - 1 <= 1 / ([m / 2] - 1)
26 RB-tree: BST 树，且满足，结点颜色红与黑，根节点和空树叶结点都为黑色，不允许红红结点连续，路径上黑结点数目相同
27 阶: 从某结点(不计)开始,到叶节点的黑色结点数量，根节点的阶成为树的阶
28 红黑树是满二叉树, k 阶红黑树高度介于 k + 1 ~ 2k + 1 之间，内部结点最少时为全是黑结点时
29 n 个内部结点红黑树的最大高度 2 *log(n + 1) + 1
30 插入算法: 约定:新插入节点标记为红色，如果父节点是红色，双红调整，叔父结点是红色:父祖换色，叔父结点是黑色，进行旋转(保持中序性质)，新的根节点为黑色，左右兄弟为

红色
31 删除算法: 调用 BST 的删除算法, 找到它的后继结点(不改变它的颜色), 删除
如果它有一个外部结点, 直接删除, 有两个外部结点, 如果他是红色, 直接删除
如果它本身是黑色
如果兄弟节点是红色, 兄弟旋上去, 原父节点变成红色, 继续调整
兄弟节点是黑色, 并且没有红色子节点, 兄弟染成红色, 父亲染成红色, 继续调成
兄弟节点有红色子节点, 把红色子节点旋到根节点, 根节点不变色, 子节点全变成黑色
32 平均/最差检索时间复杂度: O(logn), set multiset, map, multimap 有应用