查找

索引

一种尽可能减低磁盘I/O次数的索引组织方式。采用B树（B-树）这一多路平衡查找树（B+树为其变体）。

	B树（B-树）	**B+树**
	子树个数_；_关键字个数	子树个数_；_关键字个数
根节点	2 ~ m ； 1 ~ (m-1)	2 ~ m ； 2 ~ m
中间节点	⌈3/2⌉~ m ； ⌈m/2⌉-1 ~ m-1	⌈m/2⌉ ~ m ； ⌈m/2⌉ ~ m
节点重复	否	是
查找	只有随机查找	随机查找和顺序查找
关键字	各节点包含的含剪子不重复	叶子结点包含全部关键字，非叶子节点中出现的关键字也会出现于叶子节点
存储信息	节点中都包含了关键字对应记录的存储地址	叶子节点包含信息；所有非叶子节点仅作为索引，且每个索引项只含有对应子树最大关键字和指向该子树的指针，不含有该关键字对应记录的存储地址
n个关键字	对应n+1个子树	对应n个子树

B树

树中每个节点的大小为一个磁盘页，结点中所含关键字及其孩子数目取决于页的大小。度为m的B树称为m阶B树，是满足以下性质的m叉树（或空树）：

1. 根节点至少有两棵子树、至多有m棵子树（或者根节点为叶子节点）；
2. 除根结点外，所有非终端节点至少有⌈m/2⌉棵子树，至多有m棵子树；
3. 所有叶子节点都在同一层。

4. 每个结点包含(n, A0, K1, A1, k2, ... , Kn, An)，其中：
    1. n是节点中关键字个数，⌈m/2⌉-1 ≤ n ≤ m-1，n+1为子树棵树——用关键字分割子树；
    2. Ki为关键字，Ki < K(i+1)；
    3. Ai是指向孩子节点的指针，A(i-1)所指向的子树中所有节点的关键字都小于Ki，Ai则均大于Ki。

B树查找

从根节点T开始，在T所指结点的关键字向量key[1...keynum]中查找给定值K（折半查找）：

1. 若key[i] == K (1 ≤ i ≤ keynum)，查找成功返回结点和关键字位置；
2. 否则，将K与key中各个值比较，以选定查找子树：
    - 若K < key[1]：T = T->prt[0]；
    - 若key[i] < K < key[i+1] (i = 1,2,..keynum)：T = T->ptr[i]；
    - 若K > key[keynum]：T = T->ptr[keynum]；
3. 若均不满足，跳转1，知道T是叶子节点且未找到相等关键字，查找失败。

查找分析：

对于第h层：

- 最多节点数为m^(h-1)，最多关键字数为(m-1)m^(h-1)；
- 最少节点数为2*(⌈m/2⌉)^(h-2)，最少关键字数为2(⌈m/2⌉-1)(⌈m/2⌉)^(h-2)。

B树插入

插入时首先在最低层的叶子节点添加一个关键字，然后有可能"分裂"，插入过程如下（**插入看上界，超过要分裂，根分高一层**）：

1. 在B树种查找关键字K，若找到则表明已存在，否则K的查找操作失败与某个叶子节点；
2. 随后将K插入该叶子节点，插入时：
    - 若叶子节点关键字数 < m-1：直接插入；
    - 若叶子节点关键字数 == m-1：结点"分裂"。
3. 根节点分列式，由于没有父节点，则建立一个新根，B树增高一层。

B树删除

对于删除一个关键字K：

1. 找到其所在结点N并删除关键字K。
2. 如果N不是叶子节点，设K是N的第i个关键字，将指针A(i-1)所指子树中的最大关键字K'（或最小关键字）放在(K)的位置；
3. 然后删除K'，而K'在叶子节点上。

删除看下界，若自己够就从自己删除，树不调整；
自己不够，找左兄弟的最大值，或找右兄弟的最小值——兄弟上，父亲下；
若都不够，让自己、左兄弟（或右兄弟）、父亲三方合并，此时父亲也进行上述操作，逐级向上递归。

B+树

只有叶子节点存储信息，非叶子部分均为索引，同时支持顺序查找和随机查找。

散列表Hash

在记录存储地址和它的关键字之间简历一个确定的对应关系，不经比较，一次存取获得查找元素的查找方式。

基本概念：

- **哈希函数**：在关键字与存储地址之间建立关系，从关键字空间到存储地址空间的一种映像，从而得出哈希地址（哈希值）。
- **哈希表**：用哈希函数的映像记录在表中的地址，并将记录置入此地址，构成哈希表。
- **冲突**：不同关键字但生成了相同哈希值的情况。
- **同义词**：相同哈希值的两个不同关键字称为同义词。
- **哈希查找**（散列查找）：利用哈希函数进行查找。
- **散列表设计**：
    1. 空间范围，确定散列函数的值域；
    2. 构造合适的散列函数，使对于所有可能元素的哈希值均在散列表的地址空间范围，且冲突尽可能小；
    3. 设计合适的冲突处理方式。
- **哈希表评估因素**：
    1. 散列函数构造是否简单；
    2. 能否均匀将关键字映射到地址空间（冲突尽可能少）。

哈希函数构造方式

直接定址法：使用一元线性方程生成哈希值，关键字个数和地址个数一样，不会发生冲突，但由于占用空间过高，实际很少使用。

除留余数法：对关键字取余得到哈希值。取余数的大小不大于哈希表长度。是一种简单常用的构造方式。

冲突的处理方式

开放定址法：

冲突发生时，可以（由某种给定的方式，但得确保能够被找到）放置于值域的任何位置。
其公式为$Hi(key) = (H(key)+di) \% m, i = 1,2,..,k (k\leq m-1)$，
H(key)为哈希函数，m为散列表长度，di为第i此探测时的增量，Hi(key)是经过第i此探测后得到的散列表地址。

对于di的算法：

1. 线性探测法：发生冲突时，从发生冲突的位置一次向后探查。只要表中未满，总会找到位置。但每个冲突记录被散列到冲突最近的空地址，增加了更多冲突，容易“聚集”，ASL增大。查找失败的比较次数将从哈希值出发，一直到空位置为止。

2. 二次探查法：di的增量为$1^2, -1^2, 2^2, -2^2,..., \pm k^2 (k\leq \lfloor m/2 \rfloor)$，相较于当前所在位置做平方勘察。采用较大的跨距跳跃到散列表，不容易“聚集”，但不能保证准确使用到所有空间。

再哈希法：

备置多个哈希函数，冲突时使用另一个哈希函数，直到没有冲突发生。不容易“聚集”，但会增加计算时间。

链地址法：

哈希值相同的关键字存储在一个单链表，并用一维数组存放头指针。

哈希查找分析

查找效率基于ASL，关键字和给定值比较次数基于：哈希函数，处理冲突的方式，哈希表的装填因子α = (表中填入记录数)/(哈希表长度)。

ASL成功 = 比较次数 / 元素个数；ASL失败 = 比较次数 / 失败的位置量个数