数据结构和算法

多路查找树：每个节点的孩子可以多于两个，而且每个节点可以存储多个元素

B树

如何内存与外存交换数据的次数频繁，会造成时间效率上瓶颈，
B树如何减少次数呢?

外存（硬盘）将所有信息分割成相等大小的页面，每次磁盘读写都是一个或多个完整的页面，对于一个硬盘来说，一页的长度可能是211到214个字节。

B树的应用中，要处理的硬盘数据量很大，因此无法一次性的载入内存，因此我们会对B树进行调整，使B树的阶(或结点的元素)与硬盘存储的页面大小相匹配，比如一棵树的阶为1001，高度为2，它可以存储10亿个关键字，我们只需要让根结点持久的保存在内存中，那么在这颗树上，寻找某个关键字最多需要两次磁盘的读取即可

缺点:在B树结构中，往返于每个结点之间意味着我们必须的在硬盘页面之间进行多次访问。每次对结点遍历时，都会对结点的元素进行遍历，这非常糟糕，有没有可能让遍历时每个元素只访问一次？答案就是B+

- 2-3树(3阶)
- 2-3-4树(4阶)
- 多阶

B+树

最大的好处：如果要随机访问，我们从根节点出发，与B树的查找方式相同，只不过即使在分支上找到了待查找的关键字，它也只是用来索引的，不能提供实际记录的访问，还是需要到达包含此关键字的终端结点

与B树的区别：每个叶子结点都会保存一个指向后一叶子结点的指针

平衡的二叉排序树(AVL)

时间复杂度O(logn),近似二分查找

是一种动态查找算法，查询，插入和删除都是O(logn)，两个特点：1、它是排序树，2、而且是平衡的

排序

- 简单算法（适合数据量较少）：冒泡、简单选择、直接插入
- 改进算法（合适数据量较大）：希尔(不稳定)，堆(不稳定)，归并，快速(不稳定)

位运算

对内存数据直接操作，so处理速度非常快

运算逻辑

- X&1=1 OR ==0 判断奇偶(x%2==1)
- X=X & (X-1) 清零最低位的1
- X& -X 得到最低位的1
- (a^b)^c=a^(b^c)
- a^0=a
- a^a=0
- A&-A=A原码的最后一个1代表的数字

符号	描述	运算规则
&	与	两个位都为1时，结果才为1
|	或	两个位都为0时，结果才为0
^	异或	两个位相同为0，相异为1
~	取反	0变1，1变0
<<	左移	各二进位全部左移若干位，高位丢弃，低位补0
>>	右移	各二进位全部右移若干位，对无符号数，高位补0，有符号数，各编译器处理方法不一样，有的补符号位（算术右移），有的补0（逻辑右移）

更为复杂的位运算操作

1. 将 x 最右边的 n 位清零 - x & (~0 << n)
2. 获取 x 的第 n 位值(0或者1) - (x >> n) & 1
3. 获取 x 的第 n 位的幂值 - x & (1 << (n - 1))
4. 仅将第 n 位置为 1 - x | (1 << n)
5. 仅将第 n 位置为 0 - x & (~(1 << n))
6. 将 x 最高位至第 n 位(含)清零 - x & ((1 << n) - 1)
7. 将第 n 位至第0位(含)清零 - x & (~((1 << (n + 1)) - 1))

优先队列

Bitmap

- bitmap 核心优势就是算集合操作
- Bitmap 由于集合操作特性，常常用于检索系统的倒排索引

heap(堆)实现

- 小顶堆
- 大顶堆

binary search tree 实现

top K

适合所有动静态数据

案例

- 求中位数
  - 动态数据：先排序就不一定准确
  - 静态数据：先排序，在求
  - 方法：需要两个堆，一个大顶堆，一个小顶堆，数据量各站一半且小顶堆的数据都大于大顶堆的数据即可，也就是先排序，动态调整的时候，大顶堆的堆顶元素就是中位数
- 合并有效小文件
- 高性能定时器

包含10亿个搜索关键字的日志文件，如何快速获取到热门的top 10的关键字？内存1G
1、先选用散列表或二叉查找树或其他支持快速查找，插入的数据结构来记录关键字和频次。
2、选用小顶堆
3、问题来了：10亿条关键字，每条平均50个字节，那么就需要5G空间，而散列表因为要避免频繁冲突，不会选择太大的转载因子，所以消耗的内容空间就更多了，怎么办？我们可以用哈希的特点将10亿关键字通过hash算法分片到10个文件中，求的hash值同10取模，得到关键字应该就是被分到的文件编号，通过小顶堆求没个文件编号的top10 在合并之后在求top10 得出结果。

并查集

主要作用就是判断是否有环