算法、数据结构+其他

Friday, April 13, 2018 5:20 PM

堆、栈（堆栈）、队列

操作系统：

使用栈就象我们去饭馆里吃饭，只管点菜（发出申请）、付钱、和吃（使用），吃饱了就走，不必理会切菜、洗菜等准备工作和洗碗、刷锅等扫尾工作，好处是快捷，但自由度小；

使用堆就象是自己动手做喜欢吃的菜肴，比较麻烦，但是比较符合自己的口味，而且自由度大。

数据结构：

这里的堆实际上指的就是（满足堆性质的）优先队列的一种数据结构，可以被看成是一棵树，如：堆排序，第1个元素有最高的优先权；

栈实际上就是满足先进后出性质的数学或数据结构，可以被看成是一个桶。

虽然堆栈是连起来叫，但是他们还是有很大区别的，连着叫只是由于历史的原因

队列（数据结构）：

（1）队列是一种特殊的线性表，特殊之处在于它只允许在表的前端（front）进行删除操作，而在表的后端（rear）进行插入操作，和栈一样，队列是一种操作受限制的线性表。进行插入操作的端称为队尾，进行删除操作的端称为队头。

（2）队列中没有元素时，称为空队列。

（3）建立顺序队列结构必须为其静态分配或动态申请一片连续的存储空间，并设置两个指针进行管理。一个是队头指针front，它指向队头元素；另一个是队尾指针rear，它指向下一个入队元素的存储位置。

（4）队列采用的FIFO(first in first out)，新元素（等待进入队列的元素）总是被插入到链表的尾部，而读取的时候总是从链表的头部开始读取。每次读取一个元素，释放一个元素。所谓的动态创建，动态释放。因而也不存在溢出等问题。由于链表由结构体间接而成，遍历也方便。

进程和线程

关系：

（1）一个线程只能属于一个进程，而一个进程可以有多个线程，但至少有一个线程；

（2）资源分配给进程，同一进程的所有线程共享该进程的所有资源；

（3）处理机分给线程，即真正在处理机上运行的是线程；

（4）线程在执行过程中，需要协作同步。不同进程的线程间要利用消息通信的办法实现同步。线程是指进程内的一个执行单元,也是进程内的可调度实体。

区别：

（1）调度：线程作为调度和分配的基本单位，进程作为拥有资源的基本单位；

（2）并发性：不仅进程之间可以并发执行，同一个进程的多个线程之间也可并发执行；

（3）拥有资源：进程是拥有资源的一个独立单位，线程不拥有系统资源，但可以访问隶属于进程的资源.；

（4）系统开销：在创建或撤消进程时，由于系统都要为之分配和回收资源，导致系统的开销明显大于创建或撤消线程时的开销；

其他：基本概念、进程间通信（管道、信号、消息队列、共享内存等，共享内存最常用）、同步|互斥等

数组和链表（增、删、改、查+翻转（迭代|递归））

数组静态自动分配内存，连续、固定，内存空间要求高，可能浪费内存，不能动态扩展；链表动态手动申请分配内存，不连续、大小不固定，内存利用率高，扩展灵活；

数组元素在栈区；链表元素在堆区；

数组查找效率高；链表增、删、改效率高；

查找，数组时间复杂度O(1)；链表时间复杂度O(n)；

插入|删除，数组时间复杂度O(n)；链表时间复杂度O(1)。

注：

1、数组、链表可以用来实现堆栈和队列，具体来说就是用数组（结构体：元素、下标）和链表（结构体：data、ponit）的增、删、改、查来实现堆栈和队列的入栈、出栈和入队、出队；

2、堆栈先入后出（在栈顶操作），队列先入先出（尾入头出）；

3、若用链表实现堆栈，则栈顶指针（top）必须在链头，要考虑栈空，不必考虑栈满；

4、若用数组实现堆栈，要考虑栈满和栈空；

5、若用链表实现队列，则队头（front）和队尾（rear）指针必须分别在链头和链尾，要考虑队空；

6、若用数组实现队列，要考虑队满和队空；

二叉树（用一个data（节点）,两个pointer（分别指向两个子节点）的链表表示）

1、前、中、后续遍历，可以用递归，也可以用非递归（堆栈）实现。

（非递归实现的前、中序遍历较简单，后序遍历较麻烦，当然也可以像某些大神一样写成通用形式（入栈时包含2个信息：一个是节点，一个是第几次遇到），只需要调整打印语句位置即可实现3种遍历。本质是相同路线多次遇到节点，第几次遇到时打印出来。）

2、层次遍历用队列实现（先入一个根节点，然后循环执行出1（父节点）入2（子节点））；

二叉搜索树—>平衡二叉树：AVL|红黑树|B、B+树、哈夫曼树 等等。。。。。。

1、二叉搜索树要求每一个节点的左子树上的所有节点值都小于该节点，且右子树上的所有节点值都大于该节点；

2、二叉搜索树的查询（指定某一元素、最大、最小）和插入、删除均可用（双指针）链表用递归或循环实现。查询时间复杂度为O（h),h为树的高度，树越平衡，时间复杂度越好，可以达到logn；（建议查询用循环，插入、删除用递归）；

(例：判别两个序列是否是同一二叉搜索树？三种方法：分别建2、1、0个树判别.....)

（当二叉搜索树进行动态查找（进行插入、删除）时，树会变得不平衡，查找效率会变得很低，此时就要用自平衡树——**AVL|红黑树|B、B+树；**）

3、平衡二叉树要求任何一个节点的平衡因子<=1（左右子树高度之差的绝对值），查询时间复杂度为logn;

4、AVL树查找比RB树好，插入和删除较差，特别是删除；

5、当自平衡树的规则被打破时，会通过一定手段（旋转、变色等）调整以保持平衡；

6、AVL——旋转；红黑树——旋转、变色；

7、红黑树规则（若插入、删除导致打破规则，则调整）：

a.节点是红色或黑色;

b.根节点是黑色;

c.每个叶子节点都是黑色的空节点（NIL节点）；

d 每个红色节点的两个子节点都是黑色。(从每个叶子到根的所有路径上不能有两个连续的红色节点);

e.从任一节点到其每个叶子的所有路径都包含相同数目的黑色节点;

8、以上树均为二叉搜索树，查找方式一样，只不过对不同的树用同样的查找方式达到的时间复杂度不同；

9、以上不同树的插入、删除方式不同，如RB树插入和删除包含很多情况，不同情况又有不同处理方式；

10、AVL树——Windows NT内核中广泛存在；

RB树——广泛用于C++的STL中,map和set都是用红黑树实现的；

B|B+树——主要用在文件系统以及数据库做索引；

哈夫曼树——用于哈夫曼编码—数据通信非等长压缩编码，提高通信有效性；

哈希表（动态查找—以空间换时间）

两个关键点：1、散列函数（计算位置）；2、（解决）冲突；

应用：1、文件校验；2、数字签名；3、鉴权协议；如：MD4、MD5、SHA-1；

散列函数（计算简单，地址分布均匀、冲突少）：

Key-数字——直接定址法、除留余数法、数字分析法、折叠法、平方取中法。。。。。。

Key-字符——ASCII加和法、前3个字符移位法、移位法（+求余）

冲突解决：开放地址法、分离链接法；

开放地址法：线性法（i）、平方法(正负i^2)、双散列（偏移也是一个散列）、再散列（装填因子：0.5—0.85合适，太大，就扩充散列表，全部重新计算位置）；

线性法易造成“聚集”；平方法可能找不到空位（当表大小为4k+3的素数时，一定可以找到位置）；

分离链接法：冲突位置的元素用一个链表串起来；

1、性能分析：平均查找次数|ASL（成功|ASLs、不成功|ASLu）；

2、装载因子（α）越大，冲突越大，α需要保证较小，怎么使其较小，就是数组空间较大，但是里面却放较少的key，即哈希表是空间换时间；

3、哈希表时间复杂度O（1），即使发生冲突，若散列函数和冲突处理的好，冲突次数也很小，其时间复杂度与问题规模无关！！！

4、哈希表适合用在动态查找、关键字直接比较计算量大的问题上，因为哈希表的核心就是：将key转化为number，直接定位置！！！（如：变量（字符串）管理—定义-插入、调用-查找）

5、哈希表为随机存储，（动态）查找、插入、删除很方便，但是不能做顺序查找key、范围查找、最大|最小值查找等；

6、开放地址法—哈希表为：数组，存储|查找效率高、较方便，但是容易产生“聚集”；

7、分离链接法—哈希表为：数组+链表，存储|查找效率低，删除较方便（不用记录），α较大时，时间效率严重下降；

Search and Sort

1、查找:顺序、二分、树、哈希表查找等；

2、排序：简单排序（冒泡、选择、插入）、希尔、堆、归并、桶、基数和快速排序等；

3、关注点：算法程序、时间复杂度（平均、最坏）、空间复杂度、稳定性；

4、一般应用：一个无序数组，限制算法时间、空间复杂度、稳定性，用排序+查找解决问题；（标配：（先）快排+（后）二分——总体时间复杂度仍为nlogn,可以满足大多数需求）；