**数据结构与算法分析——Java语言描述**

**chap1 引论**

当一个函数是用它自己来定义的时候就称为是递归的。

i的k次方累加（i=1，2...N）的结果约等于N的k+1次方除以（k+1）

递归的四条基本准则：

1基准情形，必须对于某些情形，它无需递归就能解出

2不断推进，对于那些递归求解的情形，它每一次递归调用都要使情况朝着一种基准情形前进

3设计法则，假设所有的递归调用都能运行

4合成效益法则，在求解一个问题的同一实例时，切勿在不同的递归调用中做重复性的工作

基本类型的包装类对象都是不可变的

关于递归要深入理解递归的思想，将任务分解为相同的规模更小的任务，并且找到基准情形

**chap3 表、栈和队列**

Collection接口实现了Iterable接口

对正在被迭代的集合进行结构上的改变（remove、add、clear等）会使迭代器不再合法，但是如果用的是迭代器自己的remove方法等，这个迭代器合法

增强的for循环使用了迭代器

注意ArrayList和LinkedList的关于get、remove、add、set的操作复杂度不同

ArrayList中get和set花费常数时间，add和remove花费o（N）时间（因为要改变其他位置的值）

LinkedList中get和set花费o（N）时间，remove和add方法为常数时间操作

表可以分为链表和列表

栈和队列都可以看做是基于表的，栈是后进先出，队列是先进先出

**chap4 树**

节点n（1）~n（k）的路径定义为n1、n2...nk的一个序列，其中n（k-1）为n（k）的父亲

节点的深度为根节点到该节点的唯一路径的长度，因此根节点的深度为0

深度和高度是看路径长度还是路径包含的节点数要根据题目来判断！稍微有点

节点a到自身的路径长度为0

节点的高度为该节点到一个叶子节点的最长路径的长，因此叶子节点的高度为0，一棵树的高度等于根节点的高度

先序遍历：节点在它的子节点被处理前被处理

后序遍历：节点在它的子节点被处理后被处理

二叉树是一棵树，每个节点不能有多于两个的儿子

中序遍历的操作顺序是左子节点、父节点、右子节点

二叉查找树：对于树中的每一个节点X，它的左子树中所有的项的值都小于X的值，它的右子树中所有项的值都大于X的值

在二叉查找树中删除一个点X：如果X子节点只有0个，那么直接删除这个叶子节点，如果X有一个子节点，那么将X的父节点绕过X直接连向X的

子节点后删除X即可；如果X有两个子节点，那么找到左子树中的最大值或者右子树的最小值点Y，用Y的值代替X节点的值再递归的删除Y节点

空子树的高度通常定义为-1

AVL树定义为每个节点的左子树和右子树的高度最多相差1的二叉查找树

若M次操作的总的最坏时间为o（M\*f（N）），则称它的摊还运行时间为o（f（N））

B树理解为节点满足一些要求的M叉树，理解为平衡M叉树

Map中关键字必须是唯一的，但是不同的关键字可以映射到相同的值，因此值可以不唯一

完全二叉树：除了最下层外其它层都是满的，并且最下层节点都集中在最左边

满二叉树：每一层都是满的

深度优先：DFS是用来遍历或搜索树数据结构的算法。从根节点开始，在回溯之前沿着每一个分支尽可能远的探索。 —  Wikipedia

广度优先：BFS是用来遍历或搜索树数据结构的算法。从根节点开始，在探索下一层邻居节点前，首先探索同一层的邻居节点。 —  Wikipedia

二叉树的遍历如果按深度优先可以用递归实现

广度遍历可以用队列来实现

**chap5 散列**

本章讨论散列表（hash table）

散列是以常数平均时间执行插入、删除、查找的技术，但是需要元素间任何排序信息的树操作不会得到有效的支持

表的大小TableSize应该理解为散列数据结构的一部分而不是一种浮动于全局的变量

通常的查找是通过对数据域的某个部分进行的，这部分我们称为关键字。

散列函数：将关键字映射到0~TableSize-1范围内的某个数的映射

散列函数理想情况下应该计算简单并且要在表中单元内尽量均匀的分配关键字，减少冲突

散列表的大小应该以素数为好

装填因子是散列表中的元素个数和表TableSize的比

装填因子决定在一个散列表中查找一个元素的代价

散列冲突的解决方法：

1.分离链接法：即每个表中单元都存储一个链表，冲突的元素放入同一个链表中

2.探测散列表：不用链表，若元素冲突则进行探测，将冲突元素放入探测到的空单元中，探测方法可以是线性探测法或、二次探测法、双散列等

再散列：散列表中的装填因子过大时会导致性能下降，此时可以建立一个容量更大的新表并将原先表中的元素放入新表中的操作叫做再散列

注意装填因子不能过大，选择合理的散列函数

二叉查找树也能实现contains和insert操作，平均时间界为o（logN），而且支持有序存储

散列表实现contains和insert操作在装填因子不大的时候为o（1）但是不支持有序存储

若输入有序，二叉树可能运行的很差，平衡查找树的实现代价高，因此不需要有序信息以及对输入是否被排序存在怀疑时可以用散列这种结构

**chap6 优先队列（堆）**

二叉堆中若根节点位置为1，则第i个节点的左子节点位置为2i，右子节点位置为2i+1，父节点位置为i/2向下取整

堆可以声明为最小堆、最大堆

最小堆中对任何一个除了根节点以外的节点X，X父节点的值比X的值小（或等于）

在最小堆中插入节点的操作是上滤（空穴位置向上）：与父节点值比较，若小于父节点则将父节点的值填入空穴中并将空穴的位置移入父节点位置

最小堆中删除最小元操作是下滤（空穴位置向下）：将根节点取出，并将最后一个位置元素移入根节点，然后对根节点的值与两个子节点中较小者a比较，若比a小就结束，若比a大就将较a填入空穴（一开始是根节点），并将空穴位置设为a的位置

构建堆操作的复杂度为o（N），线性时间构建堆，方法是对N个元素的数组所有从倒数第二层到第一层逐层进行节点的下滤

**chap7 排序**

常用的内部排序都用到了内存可以直接寻址的事实，外部排序诸如在磁带上进行排序则寻址需要转动磁带，速度很慢！

大部分内部排序使用插入排序、希尔排序、归并排序和快速排序

插入排序适合非常少量的输入（N<20）

希尔排序适合中等规模的输入，只要增量序列合适

归并排序在最坏情况下表现为o（NlogN）,但是需要额外的空间

**chap8 不相交集类**

集合S中定义的等价关系R满足：

1自反性：aRa

2对称性：aRb当且仅当bRa

3传递性：若aRb，bRc，则aRc

元素a的等价类是S的子集，它包含于a有等价关系的所有元素

**chap10 算法设计技巧**

贪婪算法、分治算法、动态规划、随机化算法、回溯算法

NP问题是能在多项式时间内判断一个yes的实例是yes的问题。例如有一条狗，我们能在多项式时间内判断出它是狗，而不用考虑有一头猪，我们判断它不是狗要花多少时间。

属于NP问题的所有问题中有一个子集叫做NP完全问题。

NP中的所有问题都可以多项式时间规约成NP完全问题

规约是指映射，即p中任何实例都可以映射称为q中的一个实例

联机算法：计算完当前答案后才能进一步进行下一步计算，如装箱问题中每来一个就要装一个才能解决下一个的装箱

脱机算法：得到所有输入数据后才能进行计算，如装箱问题中知道所有箱子大小后再进行装箱操作

动态规划：将递归算法写成非递归的形式，让后者以表格形式将这些子问题的答案进行记录。比如斐波那契如果递归算的话很慢因为很多数重复计算了。写成非递归的表格形式避免了重复计算。理解为递归的话要在复杂情况下分解为简单情况的计算再合成，而动态规划就用表（有时可能是二维表也就是矩阵）一步一步由最简单的情况记录到所需的复杂情况。搞清楚算法的数学优化问题实质是什么，列出公式！搞清楚什么是最简单情况，书中很多例题都是长为1的情况为最简单情况。

随机化算法：算法期间，随机数至少有一次用于决策

随机化序列的构造中要使构造序列周期尽可能的长。

构造2的n次方的数可以用1<<n用左移的方法实现。xmod（2的n次方）可以用x与2的n次方-1的数进行位与计算即可！

测试一个数是不是素数可以用3到根号n之间的奇数来看能否被整除，偶数（除了2）肯定不是素数

借由数论的一些定理，可以用随机化算法以极大概率正确判断一个数是否是素数，判断错误的概率很小而且速度比上一行的算法快很多很多

在一步内删除一大组可能性的做法叫做裁剪

回溯算法比较复杂一些，可以用递归实现

回溯算法关键在于要理解回溯的过程，是怎么完成回溯的

回溯算法和动态规划一样要有一个类似全局的表格来记录所有或部分的解