# 第2&3章 线性表

1. 算法题：函数头声明中，&变量和\*变量的区别，例如

void InThread(Tree &p) void Inorder(Node \*q)

主函数将一个指针p的地址(指针的指针)传入InThread函数，在InThread函数内部对指针p的修改将会影响到主函数中的指针p，比如原本指针p指向结点A，在InThread中将指针p修改为指向另一个结点B，之后主函数中的指针p就指向B了。

然而在Inorder函数中就没这情况。将指向结点A的指针q传入Inorder函数。如果在Inorder函数中将p1赋值给q，p1为指向另一个结点C的指针。结束函数调用后，主函数中的q依然指向原先的结点A。换句话说，相当于在Inorder函数内部新申请了一个同类指针p2，再令p2=q，然后操作p2。但如果对q指向的结点进行操作还是会产生永久影响的。比如q->data = t，结束调用后主函数中q指向的结点A中的data就等于t了。

更新：在C++语法中函数声明中用 &变量名 表示引用。例如PTA的A1035中的用法

1. …
2. n个不同元素依一定次序进栈，不同的出栈序列为Catalan数

# 第4章 树

1. 树中一个结点的子结点个数为结点的度，其中最大的为树的度
2. 二叉树与度为2的树区别，二叉树可以空，度为2的树必须有3个结点。二叉树中父节点的孩子始终区分左右。度为2的树中父节点只有一个孩子时不区分左右。
3. 树中除根节点外所有结点都有个爹。所有结点的度是几就是几个结点的爹。
4. 二叉树中N0=N2+1。第k层最多2k-1个结点。满共2h-1个结点
5. 结点n所在层数，有n个结点树的高度：向下取[log2n]+1或向上取[log2(n+1)]
6. 完全二叉树倒数第二层也可以有叶结点。

典型题型：利用上述结论，完全二叉树，已知某层叶结点个数，求最多/最少结点数。已知结点总数，求叶结点个数。

1. 先序、中序、后序、层序遍历。递归算法、借助栈、队列的非递归算法
2. 普通二叉树的线索化。利用线索二叉树实现二叉树非递归遍历。
3. 递归遍历求深度，求叶结点个数，判树异同，寻父结点子结点等P174底部
4. 先序/后序/层序+中序序列才能推断出二叉树的唯一结构。中序才是扛把子
5. 先序+后序可以确定结点间祖孙关系。先序中a…b，后序中b…a，则a是b的祖先。
6. 从非递归遍历的角度来看。先序序列相当于入栈顺序，中序序列相当于出栈顺序。又因为先序序列加中序序列可以唯一确定一棵二叉树。所以已知一个先序序列，可以得到几个不同的二叉树的问题可以用Catalan数 ，n为结点个数。
7. 树转换为二叉树：二叉树中结点的左儿子是该结点在树中第一个儿子，右儿子是该结点在树中的相邻兄弟。森林同理，另外把所有根都串到前一棵根的右儿子上。
8. 树或森林中有n个非终端结点，二叉树中就有n+1个右指针为空结点。每一个非终端结点，它的所有儿子结点在转换为二叉树后，最后一个孩子的右指针为空。
9. 树的先根遍历对应二叉树先序遍历，后根遍历对应二叉树中序遍历
10. 并查集的存储结构：用数组来存储，数组元素下标代表元素名。根结点下标代表集合名。根结点元素为集合元素数量的负数。
11. 二叉搜索树的删除，如果被删结点有左右两个子树，则令其左子树中最大元素(中序遍历中直接前驱)或右子树中最小元素(中序遍历中直接后继)来代替。
12. 平衡二叉树，找离新结点最近的不平衡结点来调整。LL旋转是结点A的左儿子的左子树上插入了新结点。LR旋转是A的左儿子的右子树上插入新结点。RR、RL同理。
13. 深度h的平衡二叉树最小结点数递推式，N0 = 0, N1 = 1, N2 = 2, Nh = Nh-1 + Nh-2 + 1
14. 树中所有叶结点的带权路径长度之和为该树的带权路径长度WPL，WPL最小的二叉树称为Huffman树。叶结点权值越高离根结点越近。哈夫曼树中只有N0和N2
15. 没有一个编码是另一个编码的前缀。哈夫曼编码利用哈夫曼树构造出的不等长二进制编码。使用频率越高的编码放在离根结点越近的叶结点上。

# 第5章 图

1. 连通分量，生成树，无向图顶点的度之和=2e，邻接矩阵压缩

无向图确保一定连通，最少边数 = = (n-1)(n-2)/2+1

1. 邻接矩阵结构。顶点i的度 = 第i行非0个数(出度) + 第i列非0个数(入度)

邻接表结构。

邻接多重表结构。为无向图优化的邻接表。合并了重复的边结点。

十字链表结构。为有向图优化的邻接表。很容易求得顶点的入度和出度。

1. BFS、DFS 手工模拟。算法实现。

空间复杂度都是O(V)。采用邻接矩阵时间复杂度O(V2)，采用邻接表时间复杂度O(V+E)

1. 最小生成树，Prime-O(V2)，Kruskal-O(ElogE)
2. 最短路径，Dijkstra，Floyd O(V3)
3. AOV拓扑排序，算法细节，采用邻接表时间复杂度O(V+E)，序列唯一
4. AOE关键路径：关键路径中的所有活动都没有机动时间

机动时间D<i, j> = Latest[j] - ( Earliest[i] + <i, j> )

# 第6章 查找

1. B树又称多路平衡查找树，就是二叉搜索树的升级版，每个结点里可以有多个关键字，而且平衡因子全为0。所有叶结点都在同一层。m阶B树不一定要有一个结点有m个子树。根结点不是终端结点时至少有两个子树。除根结点外所有非叶结点至少有m/2向上取整个子树。B树的插入和删除规则就是建立在不破坏以上定义的前提下。
2. B+树就是文件索引块的抽象数据结构。
3. 若散列函数H(key) = key % p，则查找失败的概率是根据key % p可能的结果，即从0到p-1，每种结果等可能分布来计算的。

## 字符串模式匹配

1. 预备知识：模式串P，主串S，模式串P作为要查找的目标，在杂乱的主串S中找到一串和模式串P一模一样的子串，并返回在主串S中其开头的位置。

暴力算法：从主串的第一位S[0]开始和模式串一位一位对照。如果完全相同，则查找成功。如果有一位不相同，则主串和模式串都回溯，从主串的第二位S[1]开始和模式串一位一位对照。时间复杂度O(m×n)

1. KMP算法：推荐July的“从头到尾彻底理解KMP”。

和暴力算法区别：仅模式串P回溯，主串S不回溯。时间复杂度简化为O(m+n)。能简化的原因是模式串中常有一段前缀和后缀相同，取其中最长的称为最长前缀后缀。比如ABCDA最长前缀后缀为A，ABCDAB为AB，ABABA为ABA。如果S[k]与P[j]匹配失败，下一次比较的位置中S[k]不动，P回溯到某个位置。这个位置用next数组来表示，即P向右移动j-next[j]个位置。next数组中元素next[j]的含义就是模式串P中匹配失败的位置的前一位的最长前缀后缀的长度-1。此外next[0] = -1，next[1] = 0。

# 第7章 排序

以下默认从小到大排序

## 插入排序

1. 直接插入：时间O(n2)，空间O(1)，稳定，非全局有序，适用链表。
2. 折半插入：同上，区别仅每趟查找插入位置时用折半查找。
3. 希尔：相隔一定的步长的元素单独看作一列，分别进行插排。步长按一定规律渐减至1。

时间大约O(n1.3)，空间O(1)，不稳定。

## 交换排序

1. 冒泡：时间O(n2)，空间O(1)，稳定，全局有序。
2. 快速：将序列中一个元素作为枢纽元素。序列前后各建一个指针向中心移动，后指针遇到小于枢纽的元素将其移动序列前部分，前指针遇到大于枢纽的元素将其移动到后部分。最后留出的空位就是枢纽元素在该序列中的最终位置。用分治的思想，将枢纽元素前后部分各看作新的序列，递归进行快排。

时间平均O(nlog2n)，序列基本有序时最坏O(n2)，空间O(log2n)，不稳定。

## 选择排序

1. 简单选择：时间O(n2)，空间O(1)，不稳定。比较次数与初始状态无关n(n-1)/2。
2. 堆排序：简单选排的进阶版。建一个大顶堆，每趟从堆顶拿出一个最大元素放至堆最末尾。然后排除堆最末尾调整堆。反复至最后。

时间O(nlog2n)，空间O(1)，不稳定。

1. 盲点：堆调整(建堆，删除元素)过程中关键字比较次数，容易忽略兄弟间的比较

## 归并排序

1. 将两路或两路以上有序列合并成一个新的有序列。对于某个无序列来说，采用分治的思想，先将相邻的两个元素两两组成有序列，再将相邻的两个含有两元素的有序列两两组成有序列，最后整体成一个有序列。可以用于外部排序。

时间O(nlog2n)，空间O(n)，稳定。比较次数的数量级与初始状态无关

1. N个元素进行m趟k路归并后完成，则km = N

## 基数排序

基于关键字各位的大小进行多趟排序，一趟按各元素的一位大小放入一个序列。有几位就进行几趟。比如低位优先，先将个位进行分类收集，再进行十位，然后百位…

时间O(d(n+r))，空间O(r)，稳定

## 外部排序

1. 最佳归并树将Huffman树的思想推广到m叉树。归并段数量各异并不能保证每次归并都是m-路归并。比如3-路归并8个归并段，最后一次只能是2-路归并，所得WPL也不是最小值。但是添加长度为0的虚段可破。

严格m叉树中n0 + nm = mnm + 1。若u = (n0 – 1) % (m – 1) = 0，则正好可以构造严格m叉树。而u不为0时，则需要添加m – 1 – u个虚段即可。