第三章 栈和队列

利用栈计算中缀表达式（一般表达式）

- **数字**：读取完整数字（处理多位数），压入 OPND。
- **左括号 (**：压入 OPTR。
- **右括号)**：弹出 OPTR 栈顶运算符并计算，直到遇到 (。
- **运算符**： 若当前算符优先级 ≤ 栈顶，先计算栈顶运算并将结果压入 OPND，**直到优先级大于栈顶**，再压入当前运算符，读下一个字符

  若大于，直接压栈，读下一个字符

  ***、/** > **+、-** > 左括号**(** > 右括号**)**、**#** 最低优先级

利用栈计算后缀表达式（逆波兰表达式）

一个工作栈

遇到操作数，进栈；遇到运算符，连续从栈中退出两个操作数，进行计算

中缀表达式转换成后缀表达式

1. 若 ch 是**操作数**，则**直接输出**，读下一个字符 ch；
2. 若 ch 是**运算符**，比较 ch 的优先级和栈顶运算符的优先级：
   1. 若 ch 的优先级高，则 ch 进栈，读下一个字符 ch；
   2. 若 ch 的优先级低，则栈顶运算符退栈并输出，直到 ch 优先级大于栈顶，入栈，读下一个字符；
   3. 若两者的**优先级相等**，则**栈顶运算符退栈**，**若退出的是“(”，读下一个字符 ch**

递归的 Master 定理：

条件	复杂度	理解
$\log_b(a) > d$ **递归主导**	$\mathbf{O(n^{\log_b(a)})}$	拆分子问题的代价远高于递归外的操作，时间复杂度由拆分次数主导
$\log_b(a) = d$ **内外均衡**	$\mathbf{O(n^d \cdot \log n)}$	递归拆分与外部操作复杂度相当，每一层需叠加 O(n^d)操作，共 logn 层
$\log_b(a) < d$ **外部操作主导**	$\mathbf{O(n^d)}$	递归外操作复杂度远高于拆分，时间复杂度由外部操作主导

链式栈不设 base，由结点的 next 是否为 NULL 确定是否到头/为空栈

顺序栈 base 指向栈底，top 指向数组中下一个空闲位置

链队列设有头节点，front 指向头结点，rear 指向最后一个结点

第四章 串

KMP 算法

1.求 next 数组和 nextval 数组

j	0	1	2	3	4	5	6	7	8	9	10
P	a	b	c	d	a	b	c	d	a	b	c
next[j]	-1	0	0	0	0	1	2	3	4	5	6
nextval[j]	-1	0	0	0	-1	0	0	0	-1	0	0

2.从头开始匹配，

若主串的第 i 个字符 与模式串的第 j 个的字符失配，那么，

- 在 nextval[j]== -1 时，主串后（i++）移指针，模式串重头开始
- 否则，(主串的指针不动)主串的第 i 个字符下一次将与模式串的第 next[j]个的字符匹配

若当前 i、j 所指字符匹配成功，i++，j++；

第五章 数组和广义表

存取 n 维数组中任意位置的元素的时间相等

广义表

表头表尾分析法

- 任何一个非空广义表LS = (α1, α2, …, αn)均可分解为：
  - 表头 Head(LS) = α1
  - 表尾 Tail(LS) = (α2, …, αn)

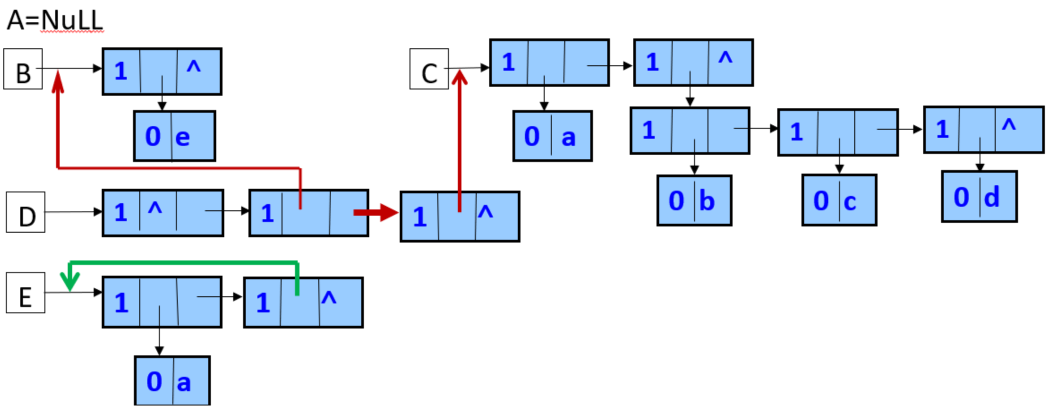
注意 tail 取出来的是个子表，最外层要加一对括号

如 **tail(L2) = ((banana, orange))** $L_2 = ((apple, pear), (banana, orange))$

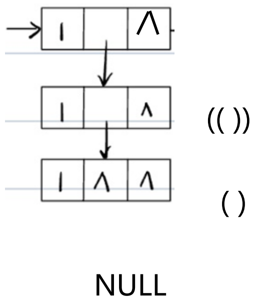
空表的表头和表尾不存在；单个表元素的表尾为空

子表分析法

A( )
B(e)
C(a, (b, c, d))
D(A, B, C)
E(a, E)



((( )))的画法：



深度为 3

广义表的深度：**所含括弧的重数，空表（ ）的深度为 1，<u>原子的深度为 0</u>**

第六章 树

树的度（叉数）：树中各个**结点的度的最大值**，通常将度为 m 的树称为 m 叉树/m 次树

树的宽度：统计树中**每一层的结点数量**，取**最大**的数量作为树的宽度

结点的层次/深度：根节点在第 1 层，其孩子的深度 + 1

结点的高度：叶结点的高度为 1；非叶结点的高度等于它的**孩子结点高度中的最大值加 1**

（其孩子中最深的叶节点到该节点的路径长度加 1）

树的层次/深度/高度：树中(叶子)结点的最大层次　　根节点深度为 1

有序树（树中结点的各个子树是有次序的，不能互换）VS 无序树

二叉树是有序树：二叉树的子树**有左右之分**，其次序不能任意颠倒

二叉树的性质：

性质 1：若二叉树结点的层次从 1 开始，则在**二叉树的第 i 层(i≥1)最多有 2^{i-1} 个结点**

性质 2：深度为 k(k≥1)的二叉树**最少有 k 个结点，最多有** 2^k **-1 个结点**

性质 3：对任何一棵二叉树，如果其叶结点有n_0个，度为 2 的非叶结点有n_2个，则有：$\boldsymbol{n_0}$ **=** $\boldsymbol{n_2}$ **+ 1**

性质：**二叉树的分支数 e 等于二叉树中所有结点的度的总和 $e=n_1+2\cdot n_2...+m\cdot n_m$**

二叉树的结点数等于所有结点的度的总和+1　$n=e+1$

$n = n_0+n_1+n_2...+n_m$

满二叉树结点编号：第 k 层第 j 个结点编号 **$C(k, j)=2^{k-1} - 1 + j$**

前面 k-1 层共有 **2^{k-1}-1 个节点**

性质 4：具有 n (n≥0) 个结点的**完全二叉树的深度为$\lfloor \log_2 n \rfloor$ + 1)**

6.18 试讨论，能否在一棵中序全线索二叉树上查找给定结点*p 在后序序列中的后继

若*p 是根节点，则没有后继；

若*p 是根节点是其双亲结点的右孩子，则后继是双亲结点；

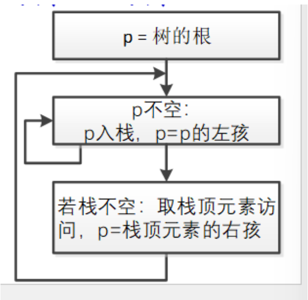
若*p 是根节点是其双亲结点的左孩子，则后继是其兄弟结点最左下的子孙。

二叉树的遍历

设访问根结点记作 D，遍历根的左子树记作 L，遍历根的右子树记作 R

遍历方式	访问顺序	实现方法
先序遍历	DLR	访问根节点-递归遍历左子树-递归遍历右子树
中序遍历	LDR	
后序遍历	LRD	

遍历算法的非递归实现（中序遍历）



层次遍历二叉树!!!

- 利用二叉树的先序序列和中序序列可以唯一确定一颗二叉树；
- 利用二叉树的后序序列和中序序列可以唯一确定一颗二叉树；
- 利用二叉树的先序序列和后序序列不能唯一确定一颗二叉树；

（两个）结点的路径长度：从一个结点到另一个结点的路径上分支的数目

树的路径长度：从树**根**到**每个结点**的**路径长度之和**

结点的带权路径长度：从**根结点**到该结点的**路径长度（分支数）**与**该结点的权**的**乘积**

树的带权路径长度：树中所有**叶子结点**的带权路径长度之和

构造 Huffman 树(严格二叉树，使用三叉静态链表)

1.先构造一个 n 棵二叉树的集合 **F** = $\{T_0, T_1, ..., T_{n-1}\}$ （**其实一开始就是 n 个结点**）

2.选取**两棵根结点的权值最小**的二叉树，做为左、右子树构造一棵新的二叉树

新树的根结点权重为两个子树权重之和

3.在 F 中删去这两颗树，把新树**加入到 F** 中。重复 2.，直到 F 中只剩一棵树

（F 数组只存储根节点，以根节点代表一颗子树）

森林转二叉树：

1. 先把每棵树转换为二叉树（孩子-兄弟树）

2. 依次把后一棵树的根结点连接为前一棵树的右孩子结点