**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）**

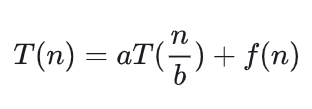
一个工作栈

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 |
| logb(a) > d **递归主导​** | **O(nlogb(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 |
| logb(a) = d **内外均衡​** | **O(nd·logn)​** | 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 |
| logb(a) < d **外部操作主导** | ​**​O(nd)​** | 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**

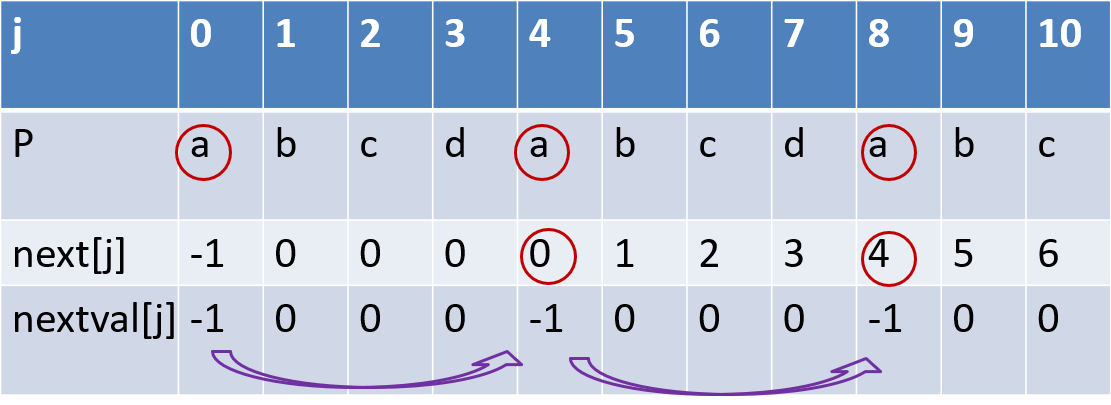
**顺序栈base指向栈底，top指向数组中下一个空闲位置**

**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

****

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

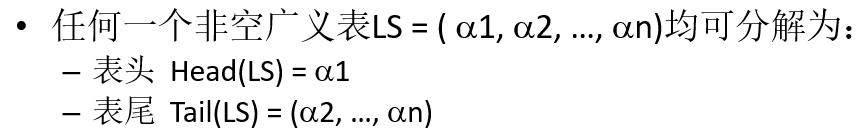
若当前i、j所指字符匹配成功，i++，j++；

**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**

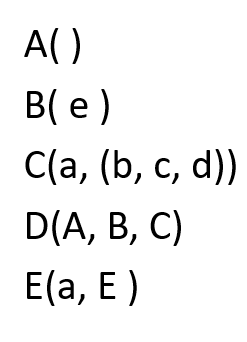
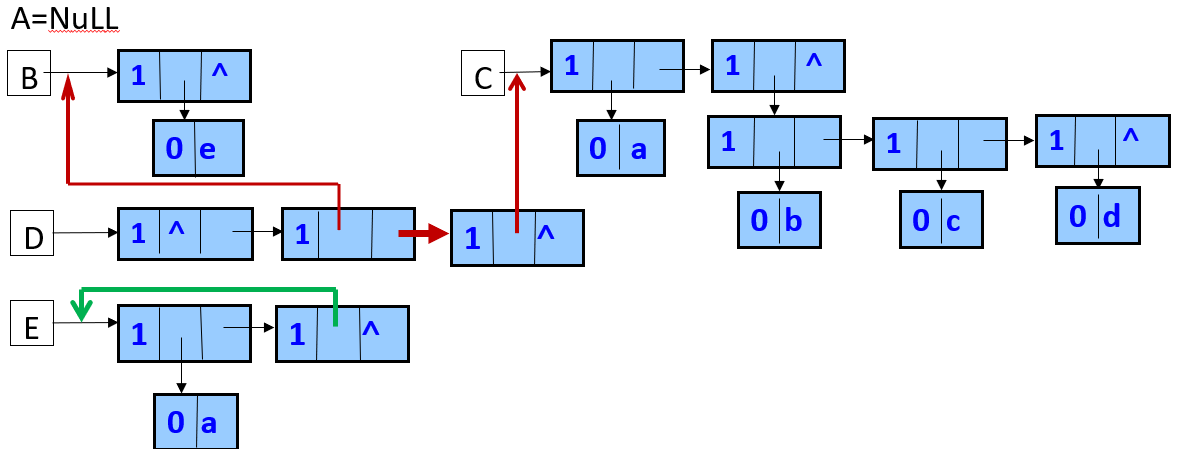


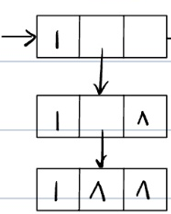
注意tail取出来的是个子表，最外层要加一对括号

如**tail(L2) = ( (banana, orange) )**

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：每个逗号间是一个子表，原子表仅有一个元素结点，非原子表存着一行表指针**



**（（（ ）））的画法**：

(( ))

( )

NULL

深度为3

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

**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 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 -1个结点~~**

**~~性质3~~**~~：对任何一棵二叉树，如果其叶结点有个，度为 2 的非叶结点有个，则有：~~ **~~= ＋ 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·n~~~~2~~~~...+m·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)=2k-1 - 1 + j**

前面k-1层共有**2k-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 + 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；

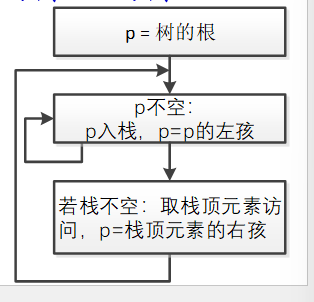
若\*p是根节点是其双亲结点的左孩子，则后继是其兄弟结点最左下的子孙。

**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

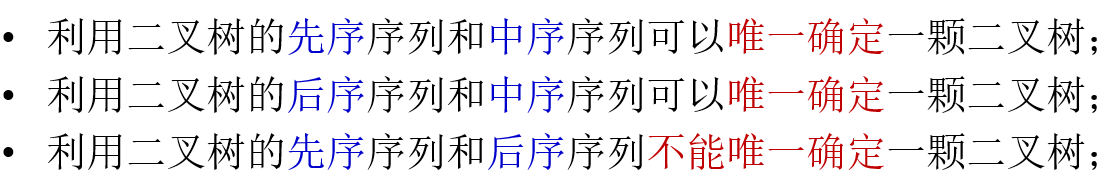
|  |  |  |
| --- | --- | --- |
| 遍历方式 | 访问顺序 | 实现方法 |
| 先序遍历 | DLR | 访问根节点-递归遍历左子树-递归遍历右子树 |
| 中序遍历 | LDR |  |
| 后序遍历 | LRD |  |

遍历算法的非递归实现（中序遍历）



**层次遍历二叉树！！！**

**给定完全二叉树的中序遍历序列可以唯一确定一颗完全二叉树：**



**（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** = {, , …., } （**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

