**数据结构和算法**

**一.基本概念和算法**

1.数据结构

1.1 基本概念

（1）**数据**：对客观事物的符号表示，能被输入到计算机中并被计算机程序处理的符号的总称。

（2）**数据元素**：数据的基本单元，通常作为一个整体进行考虑和处理。一个数据元素可由若干个**数据项**组成。

（3）数据对象：性质相同的数据元素的集合。是数据的子集。例如整数对象、字母字符数据对象等。

（4）**数据结构**：相互之间存在一种或多种特定关系的数据元素的集合。

1.2 分类

（1）集合：数据元素之间只有“属于同一集合”的关系。

（2）线性结构：结构中的元素存在一个对一个的关系。

（3）树状结构：结构中的元素存在一个对多个的关系。

（4）图状结构（网状结构）：结构中的元素存在多个对多个的关系。

1.3 结构

（1）逻辑结构：描述数据之间的逻辑关系。

（2）物理结构（存储结构）：数据结构在计算机中的表示（映像），包含数据元素的表示和关系的表示。

（3）元素之间的关系的表示方法：顺序映像和非顺序映像，对应**顺序存储结构**和**链式存储结构**。

1.4数据类型

（1）**数据类型**：一个值的集合和定义在这个值集上的一组操作的总称。

（2）**抽象数据类型**：Abstract Data Type，简称ADT，指一个数学模型和定义在该模型上的一组操作，和计算机的实现无关。通常包含**定义、表示和实现**三部分。

（3）ADT的表示：（D,S,P） D表示数据对象，S是D上的关系集，P是对D的基本操作集。

（4）ADT的定义格式

ADT抽象数据类名{

数据对象：<数据对象的定义>

数据关系：<数据关系的定义>

数据操作：<基本操作的定义>

} ADT抽象数据类名

其中数据对象的定义和数据关系的定义用伪代码描述，基本操作的定义格式为：

基本操作名（参数表）

初始条件：<初始条件描述>

操作结果：<操作结果描述>

**二.线性表**

2.1 定义：若干个元素的序列。前驱元素和后继元素。

2.2 线性表的顺序表示：用一组地址连续的存储单元依次存储线性表的元素，元素在物空间上相邻。（数组）

在数组任意位置插入或删除元素，时间复杂度：O（n）

取第i个元素的时间复杂度为O（1）

2.3 线性表的链式表示：一个节点包含两部分—数据域和指针域，数据域存储节点的数据，指针域存储其直接后继元素的位置（称为指针或链）。若每个节点只包含一个指针域，则称为**线性链表**或**单链表**。有时，我们在单链表第一个节点之前设置一个节点，用来存储线性表的长度等附加信息，称之为**头结点**。数组描述的链表称为**静态链表**。**循环链表**：最后一个节点的指针域指向头结点，整个链表形成一个环。**双向链表**：节点有两个指针域，一个指向前驱元素，另一个指向后继元素。

在链表任意位置掺入或删除元素，时间复杂度：O（n）

查找第i个元素的事件复杂度为O（n）

**三.栈和队列**

栈和队列都是特殊的线性表，基本操作都是线性表的子集，是操作受限的线性表。

3.1 **栈**

（1）定义：仅在表尾插入或删除元素的线性表。表尾称为**栈顶**，表头称为**栈底**。

（2）栈的修改时按照后进先出（LIFO）的原则进行的。

3.2 **队列**

（1）队列是一种先进后出（FILO）的线性表。它只允许在表的一段插入数据，另一端删除数据。允许插入数据的一端称为**队尾**，允许删除数据的一端称为**队头**。

（2）**双端队列**，限定在表的两端插入和删除数据的线性表；**输出受限的双端队列**，即允许两个端点插入数据，但只允许一个端点删除数据；**输入受限的双端队列**，即允许两个端点删除数据，但是只允许一个端点插入数据；限定双端队列某个端点插入的数据只能从某个端点删除，该双端队列就蜕变成两个栈底相邻的栈了。

**四.树**

4.1定义和基本术语

（1）定义：树是n个节点的集合，在任意一个非空树中，有且仅有一个特定的称为根的节点，当n>1时，其余节点可分为m个互不相交的集合，且每个集合都是一棵树（称为根的子树）。（组成：**一个根节点个m棵子树**）

（2）**节点**：包含数据和指向其子树的分支。

（3）**节点的度**：节点拥有的子树数量。度为0的节点称为**叶子**（终端节点），不为0的称为**分支节点**（非终端节点）。各节点度的最大值称为**树的度**。

（4）**树的深度**：树中节点的最大层次称为**树的深度**。

（5）**有序树**：树中节点的各子树从左向右是有次序的，则称为该树为有序树，否则称为无序树。

（5）森林：互不相交的树的集合。

4.3 **二叉树**

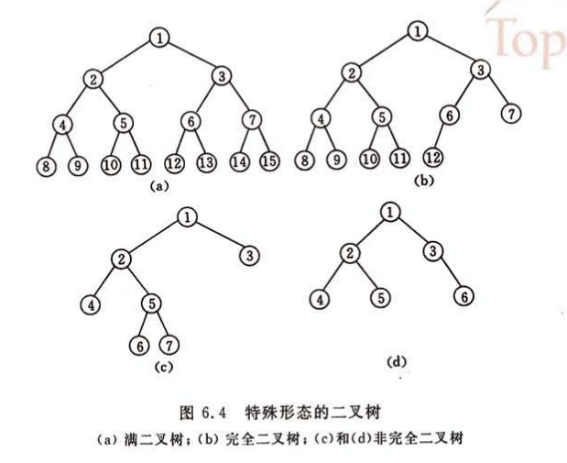
（1）定义：任意一个节点的最多有两棵子树（节点的度<=2），且二叉树的子树有左右之分。

（2）**性质1**：在二叉树的第i层上最多有2^(i-1)个节点。

（3）**性质2**：深度为k的二叉树最多有2^k-1个节点。

（4）**性质3**：如果二叉树终端节点数为n0，度为2的节点数为n2，则n0=n2+1。

**满二叉树**：深度为i的二叉树，如果总节点数为2^i-1，则称为满二叉树。

**完全二叉树**：对二叉树编号，约定从上到下，从左往右；如果一个有n个节点的二叉树，各节点的编号能够和一颗同深度的满二叉树的前n个节点一一对应，则称之为完全二叉树。

完全二叉树特点：叶子节点只会出现在最下层或倒数第二层；最下层的叶子节点集中在最左侧且连续分布；倒数第二层的叶子节点出现在最右侧且连续分布；倒数第二层只有一个子节点的节点，其子节点必定是左子节点。

（5）**性质4**：具有n个节点的完全二叉树的深度为[log2n]+1

注：[r]表示不大于r的最大整数

（6）**性质5**：有n个节点的完全二叉树，对其从上到下，从左到右编号，对任意节点i有：**i>1时，其父节点是[i/2]；如果2i>n，则节点i无左节点；2i+1>n则节点i无右节点**。

（7）二叉树遍历：先（根）序遍历（根-左->右），中（根）序遍历（左-根-右），后（根）序遍历（左-右-根）。

4.4 **二叉树的存储结构**

（1）顺序存储结构：将**完全二叉树**节点从上到下，从左到右编号，编号为i的节点存储在数组中索引为i-1的元素中。对于普通二叉树，缺少的节点对应的数组元素为null（会浪费大量的存储空间）。

（2）链式存储结构：一个节点包含数据、左右节点（和父节点）的引用，只有左右节点的引用称为二叉链表，有父节点的称为三叉链表。有n个节点的二叉链表中有n+1个空链域。

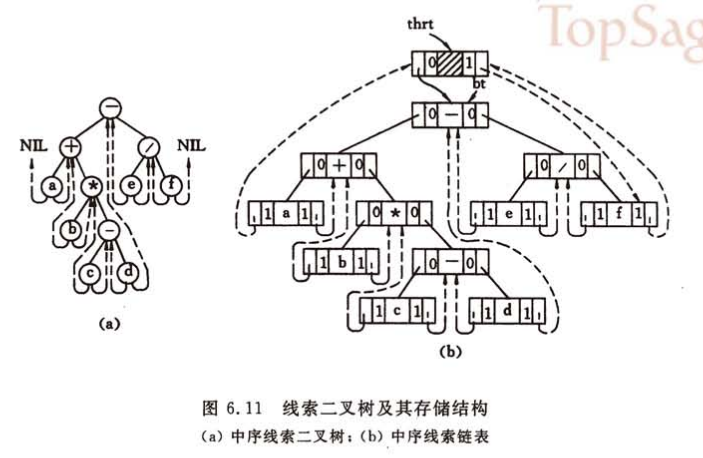
（3）**线索二叉树**：利用二叉链表中的n+1个空链域。节点新增两个属性，标记引用的是左右节点还是前驱后继节点。

lchild,ltag,data,rtag,rchild

ltag=0表示lchild指向左孩子，ltag=1表示lchild指向前驱节点；

rtag=0表示rchild指向左孩子，rtag=1表示rchild指向前驱节点。

这种结构的二叉链表称作**线索链表**，其中指向前驱和后继元素的指针称为**线索**，加上线索的二叉树称为**线索二叉树**。对二叉树遍历使其变成线索二叉树的过程叫做**线索化**。

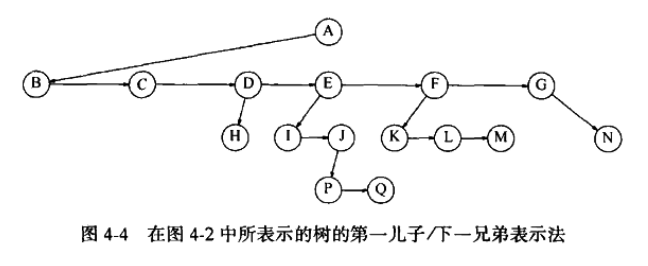
（4）二叉树的线索化：在遍历的过程中修改空指针的过程；以中序遍历为例，pre指向刚访问过的节点，如果当前节点的左孩子为null，则将做孩子指向pre节点，如果pre节点右孩子为null，则将右孩子指向当前节点。

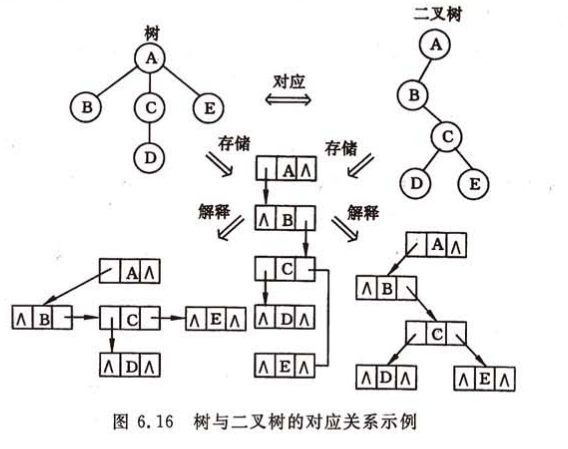
（5）线索二叉树的遍历：先找到第一个节点，依次找到后继节点直至为空为止。中序遍历，如果当前节点rtag=1,则通过rchild直接访问其后继元素，如果rtag=0时，则遍历右子树时最先访问的节点（右子树最左下的节点）是其后继元素；……遍历左子树时最后访问的节点（左子树最右下）是其前驱元素。后序遍历：若节点X是二叉树的根，则后继元素为null；若X是其父节点的右孩子或左孩子但父节点没有右孩子，X的后继元素就是父节点；若X是其父节点的左孩子，且父节点有右孩子，则X的后继元素双亲右子树遍历的第一个节点；

**4.5 树的存储结构**

（1）双亲表示法：如果以数组存储树的节点，每个节点存储其父节点的位置；查找节点的孩子时需要遍历整个数组。以链表存储时查找子节点困难。

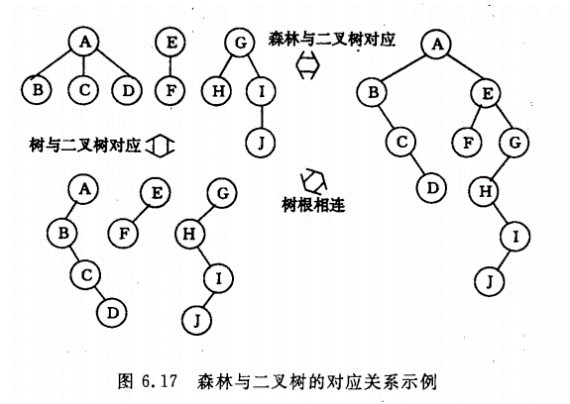
（2）孩子表示法：每个节点存储所有子节点的位置。

（3）**孩子兄弟表示法**：每个节点存储第一个子节点和下一个兄弟节点的引用，见右图。这种存储结构查找子节点和兄弟节点都比较方便。

**4.6森林**

（1）利用孩子兄弟表示法，任何树都可以使用二叉链表作为存储结构。任何树都有一颗二叉树与之对应。见右图。

（2）森林和二叉树的对应关系：一棵树对应的二叉树右子树必为空，如果把第二颗树的根节点看成第一棵树根节点的兄弟节点，则可以导出森林和二叉树之间的关系。森林的遍历和二叉树的遍历方式相同，见下图。

**5.7 赫夫曼树和应用**

(1)**路径长度**：从树中一个节点到另一个节点之间的分支构成这两个节点之间的路径，路径上分支数目称为路径长度。

（2）树的路径长度：树根到每个节点的路径长度之和。

（3）**树的带权路径长度**：节点的**带权路径长度**为从该**节点到树根**之间路径长度和节点的权重的乘积。树的带权路径长度为树中所有**叶子节点**的带权路径长度之和，通常记做WPL= 。

（4）**最优二叉树或赫夫曼树**：带权路径长度最小的二叉树称为最优二叉树。赫夫曼树中没有度为1的节点（这一类树称为严格的或正则的二叉树），原因见赫夫曼算法；有n个叶子节点的赫夫曼树共有2n-1个节点。

（5）赫夫曼算法：用于构造赫夫曼树的一般算法。

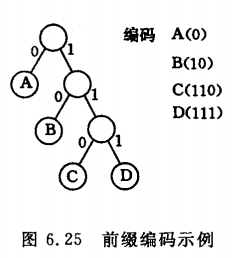
第一步：初始情况，n棵二叉树的集合{T1,T2…Tn}，权重分别为{w1,,w2…wn}，集合中的每棵二叉树左右子树都为空。

第二步：将集合中权重最小的两棵子树挑选出来作为左右子树构造一颗新的二叉树，新二叉树的根节点的权重等于这两棵左右子树权重之和。

第三步：在集合中删除这两棵子树，并将新的子树放入集合中。

第四步：重复第二步和第三步直至集合中只含有一棵树为止，这棵树就是赫夫曼树。

（6）赫夫曼编码

问题：如果以右图所示的方式对字符编码，现在需要编码后长度电报的内容最短。

方案：每个字符在电报中出现的次数wi，编码长度为li ,则电报的总长度为 。对应一颗二叉树，wi是叶子的权重，li是根节点到叶子的路径长度，电报的长度则是二叉树的带权路径长度。当对应二叉树是赫夫曼树时报文编码后长度最短，这种方式得到的前缀编码称为**赫夫曼编码**。

前缀编码：任何一个字符的编码都不是另一个编码的前缀。

**5.8 树的计数**

（1）二叉树的相似：T1和T2相似指的是，两者都为空树或二者都不为空树，且他们的左右子树分别相似。（两棵二叉树的形状相同）。

（2）二叉树的等价：T1和T2等价指的是，两者不但相似，而且对应节点上数据也相等。

（3）二叉树的计数问题就是具有n个节点、互不相似的的二叉树的数目：