第四章、树

1、熟练掌握二叉树的基本性质。
2、熟练掌握二叉树的各种存储结构的实现，各存储结构的特点及适用范围。
3、熟练掌握二叉树各种遍历策略的递归算法。
4、熟练掌握基于遍历策略的二叉树操作及应用。
5、树(森林)与二叉树的关系(存储)。
6、了解最优树的特性，掌握建立最优树和哈夫曼编码的方法。

基本术语

结点的度：孩子的个数
树的度：树中结点最大的度
堂兄弟：双亲在同一层的结点
结点的深度：从根结点向下逐层累加
结点的高度：从叶结点向上逐层累加
两结点的路径：两结点间的结点序列
满二叉树：一颗深度为k，且有2^k-1个结点的二叉树
完全二叉树：深度为k，且有n个结点的二叉树，且每一个结点都与深度为k的满二叉树中编号从 1 至n的结点一一对应

二叉树的性质

1、在二叉树的第i层上至多有$2^{i-1}(i\geq 1)$个结点
2、深度为的二叉树至多有$2^{k-1}(k\geq 1)$个结点
3、叶子结点数为n_0，度为 2 的结点数n_2，$n_0=n_2+1$
4、具有n个结点的完全二叉树深度=$\lfloor\log_2 n\rfloor+1$
5、对有n个结点的完全二叉树按层序编号，则对任一结点$i(1\leq i\leq n)$有

待判断	判断条件	结果
双亲	$i=1$	无双亲（根）
	$i>1$	$PARENT(i)=\left\lfloor\frac{i}{2}\right\rfloor$
左孩子	$2i>n$	无左孩子（叶子结点）
	$2i\leq n$	$LCHILD(i)=2i$
右孩子	$2i+1>n$	无右孩子
	$2i+1\leq n$	$RCHILD(i)=2i+1$

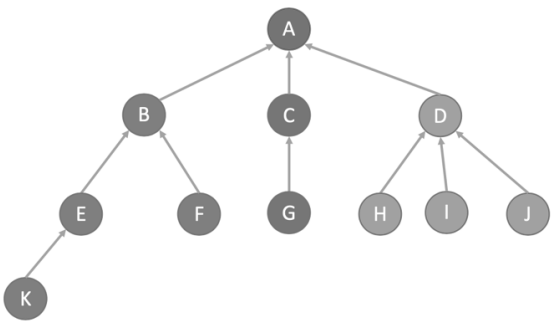
二叉树的各种存储结构的实现，各存储结构的特点及适用范围

<table>
<tr><th colspan="2">存储结构</th><th>定义</th><th>特点</th></tr>
<tr><td colspan="2">顺序存储结构</td><td>将完全二叉树上编号为i的结点元素存储在一维数组中下标为$i-1$的分量中</td><td>适用于完全二叉树</td></tr>
<tr><td rowspan="2">链式存储结构</td><td>二叉链表</td><td>结点结构：
<table><tr><td>lchild</td><td>data</td><td>rchild</td></tr></table></td><td>n个结点的二叉链表有$n+1$个空链域</td></tr>
<tr><td>三叉链表</td><td>结点结构：
<table><tr><td>lchild</td><td>data</td><td>parent</td><td>rchild</td></tr></table></td><td></td></tr>
</table>

```
//二叉树的顺序存储表示
#define MAX_TREE_SIZE 100             //二叉树最大结点数
typedef int SqBiTree[MAX_TREE_SIZE]; // 0 号单元存储根结点
SqBiTree bt;

//二叉树的二叉链表存储表示
typedef struct BiTNode
{
    TElemeType data;
    struct BiTNode *lchild, *rchild; //左右孩子指针
} BiTNode, *BiTree;
```

树(森林)与二叉树的关系(存储)
比如存储个



方法	解释	优缺点
	每个结点中保存指向双亲的指针，根结点的双亲保存为 0 或-1 `typedef struct PTNode`	

双亲表示法（顺序存储）

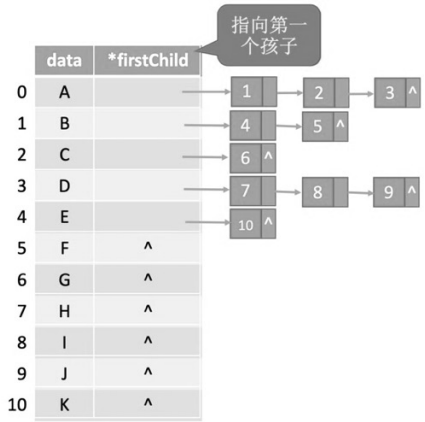
```
{ //结点结构
    TElemType data;
    int parent; //双亲位置域
} PTNode;
typedef struct
{ //树结构
    int nodes[MAX_TREE_SIZE];
    int r, n; //根的结点和结点数
} PTree;
```

	data	parent
0	A	-1
1	B	0
2	C	0
3	D	0
4	E	1
5	F	1
6	G	2
7	H	3
8	I	3
9	J	3
10	K	4
11		

求指定结点的孩子需要遍历整个结构

寻找双亲方便

孩子表示法（顺序+链式）



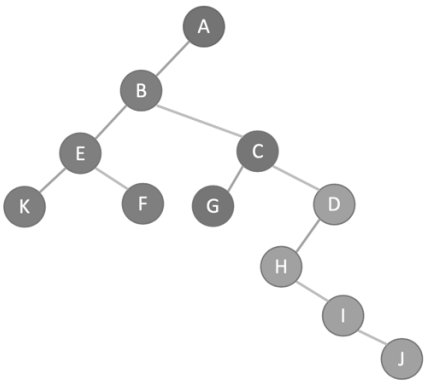

```
typedef struct CTNode
{ //结点结构
    int child;
    struct CTNode *next;
} * ChildPtr;
typedef struct
{ //树结构
```

便于对孩子进行操作

不适用于对双亲进行操作

<table>
<tr><td></td><td>

```
    int data;
    ChildPtr firstchild; //孩子链表头指针
} CTBox;
typedef struct
{
    CTBox nodes[MAX_TREE_SIZE];
    int n, r; //结点数和根的位置
} CTree;
```

</td><td></td></tr>
<tr><td>孩子兄弟表示法（链式）</td><td>右结点存储右兄弟，相当于是把这棵树搞成这样了
 


```
typedef struct CSNode
{
    int data;
    struct CSNode *firstchild, *nextsibling;
} * CSTree;
```

结点结构：

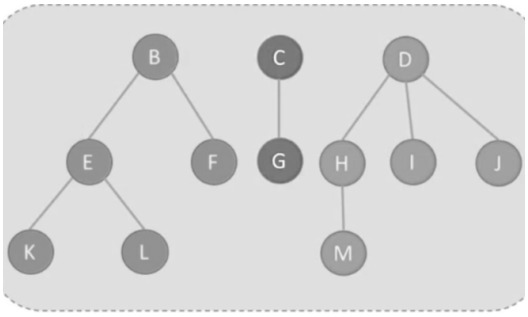
左孩子	data	右兄弟

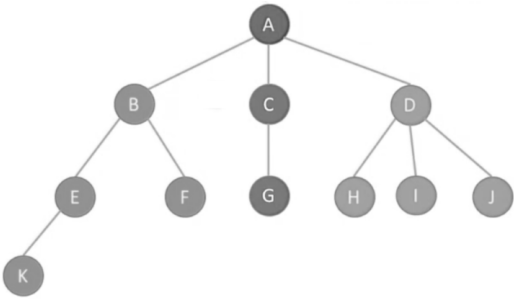
</td><td>也是树和二叉树的转化

易于找结点孩子</td></tr>
</table>

二叉树、树和森林的遍历

likethis





		描述	等同于
遍历的转化关系：森林→树→二叉树			
二叉树	先序遍历	根左右	/
	中序遍历	左根右	/
	后序遍历	左右根	/
	层序遍历	从上到下，从左到右	/
树	先根遍历	先访问根，再对每棵子树进行先根遍历。 如图，先根遍历序列为 A B C D A (B E F) (C G) (D H I J) A (B (E K) F) (C G) (D H I J)	（对应的）二叉树的先序遍历序列
	后根遍历	依次对每棵子树进行后根遍历，最后访问根. 如图，后根遍历序列为 B C D A (E F B) (G C) (H I J D) A ((K E) F B) (G C) (H I J D) A	（对应的）二叉树的中序遍历序列
	层序遍历	队列实现 *step1.*树非空，根结点入队 *step2.*队列非空，队头元素出队并访问，同时将该元素的孩子依次入队 *step3.*重复 *step2* 直到队列为空	/
森林	先序遍历	*step1.*森林非空，访问第一棵树的根结点 *step2.*先序遍历第一棵树中根结点的子树森林 *step3.*先序遍历剩下的树构成的森林 B C D (B E F) (C G) (D H I J) (B (E K L) F) (C G) (D (H M) I J)	等同于依次对各个树进行先根遍历 也等同于对应的二叉树的先序遍历
	中序遍历	*step1.*森林非空，中序遍历森林中第一棵树的根结点的子树森林 *step2.*访问第一棵树的根结点 *step3.*中序遍历剩余的树构成的森林 B C D (E F B) (G C) (H I J D) ((K L E) F B) (G C) ((M H) I J D)	等同于依次对各个子树进行后根遍历 也等同于对应的二叉树的中序遍历

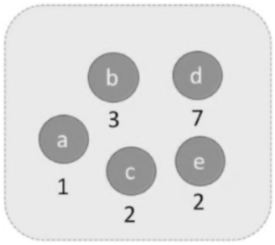
建立最优树（赫夫曼算法）

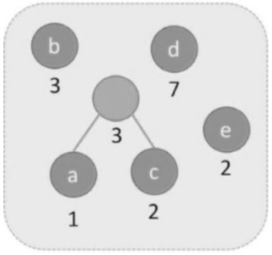
step1. 给定一个二叉树的集合，其中每棵二叉树中只有一个带权的根结点，其左右子树为空

step2. 访问二叉树集合，选两棵根结结点权值最小的树作为左右结点，构成新的二叉树，权值为两结点之和

step3. 删掉这两颗树，将新的二叉树加入二叉树集合中

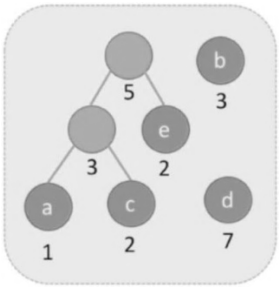
step4. 重复 *step2、3*

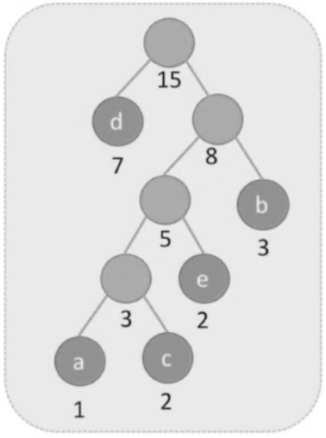




1　2

3　4





结点的带权路径长度：经过的边数与结点权值的乘积

树的带权路径长度WPL：所有叶结点的带权路径长度之和。这颗树的WPL就是1×7（d结点）$+3\times2$（b）$+2\times3$（e）$+2\times4$（c）$+1\times4$（a）

最优二叉树/赫夫曼树：由n个权值构造出的WPL最小的二叉树

最优树的特性

1、每个初始结点最终都为叶结点

2、权值最小的结点到根结点的路径长度越大

3、结点总数为$2*$叶子结点-1

4、不存在度为 1 的结点

5、不唯一，但WPL必然相同为最优

哈夫曼编码（用于数据压缩）的方法

方法	解释
前缀编码	任一个字符编码都不是另一个字符编码的前缀，解码无歧义
固定长度编码	每个字符用相等长度的二进制位表示
可变长度编码	允许对不同字符用不等长的二进制表示