**6.6树和森林**

**树的区分：如果是一棵二叉树，则一定要区分出左**

**右子树，否则只能称为树，不能称为二叉树。这里所说的的树实际是指多叉树，当然二叉树也含在其中。**

**如：下列树是一棵二叉树**

**又如：下列树由于结点F是垂直下落，所以其不能算作左子树或右子树。只能称为树，不能称为二叉树。**

**6.6.1树的存储结构**

**常用的有三种方法：双亲表示法、孩子链表法和孩子兄弟链表法。**

**根据这三种方法提供的信息都可以画出对应树的结构。**

**（1）双亲表示法**

**每个结点包含一个数据域和一个指针域（存放双亲结点的序号）**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **序号** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **数据** | **A** | **B** | **C** | **D** | **E** | **F** | **G** |
| **双亲序号** | **-1（无）** | **1** | **1** | **2** | **2** | **2** | **3** |

**定义如下：**

**typedef struct**

**{**

**DataType data;**

**int parent;**

**}PTreeNode;**

**typedef struct**

**{**

**PTreeNode node[MAXSIZE];**

**int num;**

**}PTree;**

**（2）孩子链表法**

**双亲结点的指针链接着孩子的结点，孩子结点只需标出编号即可。**

|  |  |  |
| --- | --- | --- |
| **1** | **A** | 3  2 |
| **2** | **B** | 6  5  4 |
| **3** | **C** | 7 |
| **4** | **D** |  |
| **5** | **E** |  |
| **6** | **F** |  |
| **7** | **G** |  |

**定义如下：**

**typedef struct Cnode**

**{**

**int no;**

**struct Cnode \*next;**

**}ChNode;**

**typedef struct**

**{**

**DataType data;**

**ChNode \*child;**

**}ChTreeNode;**

**typedef struct**

**{**

**ChTreeNode nodes[MAXSIZE];**

**int num;**

**}PTree;**

**（3）孩子兄弟链表法**

**双亲结点的左子针指向最左的孩子结点，孩子结点的右指针指向右面的亲兄弟。**

**Root**

B

A

C

G

D

E

F

**6.6.2树、森林和二叉树的转换**

**一般的树或森林比二叉树更为普遍，但是前两者由于规律性差而难于处理。为了需要和方便，可以将一般树或森林转化为等价的二叉树再进行处理。当然也可以进行相反的转换。**

1. **树转换成二叉树**

**一棵树转换成等价的二叉树可以分三步完成：**

**（1）连线：相邻亲兄弟之间作连线；**

**（2）抹线：除保留双亲与最左孩子的连线外，抹**

**掉双亲与其他孩子之间的连线；**

**（3）旋转：作适当的顺时针旋转，旋转到有关树**

**结点能分出左或右子树即可，不要转过了头。**

**例：将一棵树转换成等价的二叉树**

**原树：**

**连线（红色）和抹线：**

**连线和抹线后：**

**旋转后的等价二叉树：**

**2.森林转换成二叉树**

**分为两步：**

**（1）将森林中的每一棵树分别转换成等价二叉树；**

**（2）合并：将第2棵树接入第1棵的右边成为它的右子树，第3棵接入第2棵的右边成为它的右子树，……**

**例如：森林中有三棵树如下**

**第1棵 第2棵**

**第3棵**

**说明：第1、3棵树转换成等价的二叉树需要连线、抹线和转角度，而第2棵树由于结构原因只需转角度即可。**

**将三棵树分别转换为等价二叉树**

**第1棵 第2棵 第3棵**

**将三棵二叉树合并成一棵二叉树**

**3.二叉树换转成树或森林**

**分为两步：**

**（1）断开右链：将二叉树的根结点的右链及右链的右链等全部断开，得到若干棵无右子树的二叉树**

**（2）二叉树还原成树或森林（与上节相反）**

**例如：将上例按相反的次序进行**

**先断开右链**

**断开右链后形成三棵二叉树**

**第1棵 第2棵 第3棵**

**三棵二叉树还原成树和森林**

**第1棵 第2棵**

**第3棵**

**6.6.3树和森林的遍历**

**在树或森林中，一个结点可能有两个以上的孩子结点，不便讨论中序遍历（因为“足迹”为“下”可能不止一个，也可能没有，如前面的第1、2棵树）；先序和后序遍历还是可以讨论的（因为“足迹”为“左”或“右”能唯一确定）。**

**树和森林的先序及后序遍历：**

**1. 先序遍历**

**（1）树的先序遍历（若树非空）：先遍历根结点，然后依次先序遍历各子树；**

**（2）森林的先序遍历（若森林非空）：先序遍历森林中第一棵树、第二棵树、第三棵树、……。**

**上节例三棵树先序遍历的结果是：**

**A，B，C，D，E，F，G，H，I**

**2. 后序遍历**

**(1)树的后序遍历（若树非空）：后序遍历各子树，最后遍历根结点；**

**(2)森林的后序遍历（若森林非空）：后序遍历森林中的第一棵树、第二棵树、第三棵树、……。**

**上节例三棵树后序遍历的结果是：**

**B，C，D，A，F，E，H，I ，G**

**注意各种遍历次序的关系是：**

**(1)树或森林与其等价的二叉树的先序遍历序列相同；**

**(2)树或森林的后序遍历序列与其等价的二叉树的中序遍历序列相同。**

**例如，由下列三棵树组成的森林和转换后的二叉树遍历序列比较**

**第1棵 第2棵**

**第3棵**

**森林的等价二叉树：**

**森林和等价二叉树遍历对照：**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **第1棵** | | | | **第2棵** | | **第3棵** | | |
| **森林的先序**  **二叉树先序** | **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** |
| **森林的后序**  **二叉树中序** | **B** | **C** | **D** | **A** | **F** | **E** | **H** | **I** | **G** |

**由此可知，当处理树或森林的存储时，其先序和后序遍历可分别借用对应二叉树的先序和中序处理。**

**6.7哈夫曼树（Huffman Tree）及其应用**

**6.7.1哈夫曼树的定义**

**●路径和路径长度：在一棵树中，从一个结点向下可以到达的孩子或子孙结点之间的通路，称为路径。通路中分支的数目称之为该路径长度。**

**显然结点的路径长度与其所在的层数（L）有关，即各结点的路径长度=L-1；**

|  |  |  |  |
| --- | --- | --- | --- |
| **结点** | **结点所在层数(L)** | **路径** | **路径长度(L-1)** |
| **A** | **1** | **A** | **0** |
| **B** | **2** | **A-B** | **1** |
| **C** | **2** | **A-C** | **1** |
| **D** | **3** | **A-C-D** | **2** |
| **E** | **3** | **A-C-E** | **2** |
| **F** | **4** | **A-C-D-F** | **3** |

**●结点的权和带权路径长度：若将树中的结点赋给一个有某种意义的数值，则该值被称为结点的权（W）。从树的根结点到某结点之间的路径长度与该结点权的乘积，称为该结点的带权路径长度。**

**●树的带权路径长度（WPL）：若二叉树有若干个叶子结点，它们的带权路径长度之和记为： 称为树的带权路径长度；**

**其中：n为叶结点的个数，和分别为第k个叶子结点的路径长度和权值。**

**例如：结点A，B，C，D的权值分别是4，5，6，7，可以构成不同的二叉树，所得到的WPL的值是不同的，最好能找到最小的。**

**第1棵：WPL=7\*1+6\*2+(4+5)\*3=46**

**第2棵：WPL=4\*1+7\*2+(5+6)\*3=51**

**第3棵：WPL=(4+5+6+7)\*2=44（最小，最优，哈夫曼树）**

**●带权路径长度（WPL）最小的二叉树称为最优二叉树。哈夫曼提出了构造最优二叉树的方法，所以又称其为哈夫曼树。**

**6.7.2哈夫曼树的构造方法**

**如何构造一棵哈夫曼树呢？根据哈夫曼树的定义，一棵哈夫曼树要使其WPL为最小，必须使权值越大（小）的叶子结点离根结点越近（远）。**

**哈夫曼提出了构造的方法，俗称哈夫曼算法：**

**设有n个叶子结点的权值分别为W1,W2,…,Wn，构造哈夫曼树的方法如下：**

**（1）将W1,W2,…,Wn看成是有n棵二叉树的森林（每棵二叉树仅有一个结点）；**

**（2）在森林中选取两个权值最小的结点（可能不止一种选取）分别作为左孩子和右孩子结点（左右孩子顺序可随意）添加一个根结点（其权值为左右孩子结点的权值之和），构造一棵新的二叉树，与未使用过的结点或子树组成新的森林。**

**（3）重复（2）直到森林中构成一棵二叉树为止，该二叉树就是哈夫曼树。**

**注意：每个叶子、子树和根结点只能使用一次。**

**例1：设有4个叶子结点A，B，C，D的权值分别是4，5，6，7，构建哈夫曼树**

**第1步：在结点4，5，6，7中选结点4和5**

**第2步：在结点9，6，7中选结点6和7**

**第3步：只有结点9和13**

**第4步：哈夫曼树成型**

**WPL=(4+5+6+7)\*2=44（最小）**

**哈夫曼树生成前初始情况：**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **次序Order** | **名称**  **Name** | **权值**  **Weight** | **双亲**  **Parents** | **左孩子**  **Lchild** | **右孩子**  **Rchild** |
| **1** | **A** | **4** |  | **无** | **无** |
| **2** | **B** | **5** |  | **无** | **无** |
| **3** | **C** | **6** |  | **无** | **无** |
| **4** | **D** | **7** |  | **无** | **无** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**哈夫曼树生成后情况：**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **次序Order** | **名称**  **Name** | **权值**  **Weight** | **双亲**  **Parents** | **左孩子**  **Lchild** | **右孩子**  **Rchild** |
| **1** | **A** | **4** | **5** | **无** | **无** |
| **2** | **B** | **5** | **5** | **无** | **无** |
| **3** | **C** | **6** | **6** | **无** | **无** |
| **4** | **D** | **7** | **6** | **无** | **无** |
| **5** |  | **9** | **7** | **1** | **2** |
| **6** |  | **13** | **7** | **3** | **4** |
| **7** |  | **22** | **无** | **5** | **6** |

**Huffman1.c**

**#define LEAFN 4**

**#define M 2\*LEAFN-1**

**#define MIN 9999**

**struct tree**

**{**

**int weight,parent,lch,rch;**

**}hftree[M+1]={{0},{4},{5},{6},{7}};**

**void creathuffmantree()//构建哈夫曼树**

**{**

**int i,j,p1,p2,s1,s2;**

**for(i=LEAFN+1;i<=M;i++)**

**{**

**p1=p2=0;s1=s2=MIN;**

**for(j=1;j<=i-1;j++)**

**if(hftree[j].parent==0)**

**if(hftree[j].weight<s1)**

**{**

**s2=s1;**

**s1=hftree[j].weight;**

**p2=p1;p1=j;**

**}**

**else if(hftree[j].weight<s2)**

**{s2=hftree[j].weight;p2=j;}**

**hftree[p1].parent=hftree[p2].parent=i;**

**hftree[i].lch=p1;hftree[i].rch=p2;**

**hftree[i].weight=hftree[p1].weight+**

**hftree[p2].weight;**

**}**

**}**

**void main()**

**{**

**int i;**

**printf("构建哈夫曼树之前：\n Order Weight Parent Lchild Rchild:\n");**

**for(i=1;i<=M;i++)printf("%4d%7d%7d%7d%7d\n",**

**i,hftree[i].weight,hftree[i].parent,**

**hftree[i].lch,hftree[i].rch);**

**creathuffmantree();**

**printf("\n构建哈夫曼树之后：\n Order Weight Parent Lchild Rchild:\n");**

**for(i=1;i<=M;i++)printf("%4d%7d%7d%7d%7d\n",**

**i,hftree[i].weight,hftree[i].parent,**

**hftree[i].lch,hftree[i].rch);**

**}**

**思考题：构造出的哈夫曼树是唯一的吗？**

**例2：4个叶子结点A，B，C，D的权值分别是4，5，7，10，构建哈夫曼树**

**第1步：在结点4，5，7，10中选4和5**

**第2步：在结点9，7，10中选9和7**

**第3步：只有结点10和16（结点D可以放在结点16的左边或右边）**

**第4步（1）：结点D放在结点16的左边**

**哈夫曼树成型之一：WPL=10\*1+7\*2+(4+5)\*3=51**

**第4步（2）：结点D放在结点16的右边**

**哈夫曼树成型之二：WPL=10\*1+7\*2+(4+5)\*3=51**

**结论：根据上例可见哈夫曼树的构造不唯一，但WPL的值一定最小。**

**例3：设有8个叶子结点A，B，C，D，E，F，G，H的权值分别是5，29，7，8，14，23，3，11，构建哈夫曼树**

**哈夫曼树生成前初始情况：**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **次序**  **Order** | **名称**  **Name** | **权值**  **Weight** | **双亲**  **Parent** | **左孩子**  **Lchild** | **右孩子**  **Rchild** |
| **1** | **A** | **5** |  | **无** | **无** |
| **2** | **B** | **29** |  | **无** | **无** |
| **3** | **C** | **7** |  | **无** | **无** |
| **4** | **D** | **8** |  | **无** | **无** |
| **5** | **E** | **14** |  | **无** | **无** |
| **6** | **F** | **23** |  | **无** | **无** |
| **7** | **G** | **3** |  | **无** | **无** |
| **8** | **H** | **11** |  | **无** | **无** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**哈夫曼树生成后情况：**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **次序**  **Order** | **名称**  **Name** | **权值**  **Weight** | **双亲**  **Parent** | **左孩子**  **Lchild** | **右孩子**  **Rchild** |
| **1** | **A** | **5** | **9** | **无** | **无** |
| **2** | **B** | **29** | **14** | **无** | **无** |
| **3** | **C** | **7** | **10** | **无** | **无** |
| **4** | **D** | **8** | **10** | **无** | **无** |
| **5** | **E** | **14** | **12** | **无** | **无** |
| **6** | **F** | **23** | **13** | **无** | **无** |
| **7** | **G** | **3** | **9** | **无** | **无** |
| **8** | **H** | **11** | **11** | **无** | **无** |
| **9** |  | **8** | **11** | **7** | **1** |
| **10** |  | **15** | **12** | **3** | **4** |
| **11** |  | **19** | **13** | **9** | **8** |
| **12** |  | **29** | **14** | **5** | **10** |
| **13** |  | **42** | **15** | **11** | **6** |
| **14** |  | **58** | **15** | **2** | **12** |
| **15** |  | **100** | **无** | **13** | **14** |

**WPL=(23+29)\*2+(11+14)\*3+(3+5+7+8)\*4=271**

**Huffman2.c**

**#define LEAFN 8**

**#define M 2\*LEAFN-1**

**#define MIN 9999**

**struct tree**

**{**

**int weight,parent,lch,rch;**

**}hftree[M+1]={{0},{5},{29},{7},{8},{14},{23},{3},{11}};**

**void creathuffmantree()//构建哈夫曼树**

**{**

**int i,j,p1,p2,s1,s2;**

**for(i=LEAFN+1;i<=M;i++)**

**{**

**p1=p2=0;s1=s2=MIN;**

**for(j=1;j<=i-1;j++)**

**if(hftree[j].parent==0)**

**if(hftree[j].weight<s1)**

**{**

**s2=s1;s1=hftree[j].weight;p2=p1;p1=j;**

**}**

**else if(hftree[j].weight<s2)**

**{s2=hftree[j].weight;p2=j;}**

**hftree[p1].parent=hftree[p2].parent=i;**

**hftree[i].lch=p1;hftree[i].rch=p2;**

**hftree[i].weight=hftree[p1].weight+**

**hftree[p2].weight;**

**}**

**}**

**void main（）**

**{**

**int i;**

**printf("****构建哈夫曼树之前：\n Order Weight**

**Parent Lchild Rchild:\n");**

**for(i=1;i<=M;i++)printf("%4d%7d%7d%7d%7d\n",**

**i,hftree[i].weight,hftree[i].parent,**

**hftree[i].lch,hftree[i].rch);**

**creathuffmantree();**

**printf("\n构建哈夫曼树之后：\n Order Weight**

**Parent Lchild Rchild:\n");**

**for(i=1;i<=M;i++)**

**printf("%4d%7d%7d%7d%7d\n",i,**

**hftree[i].weight,hftree[i].parent,**

**hftree[i].lch,hftree[i].rch);**

**}**

**●述评：**

**(1)给定一组权值确定的叶子结点所构造的哈夫曼树，形状可以不同（即不唯一），但它的带权路径长度WPL的值是相同的且一定最小。**

**(2) 哈夫曼树要使其WPL的值最小，必须使权值越大（小）的叶子结点离根结点越近（远）。**

**6.7.3 哈夫曼树的用途和哈夫曼编码**

**先看一个英文字母密码用二进制码代替的问题。**

**若有英文字母构成的密码是：“ACBCCACBCCDCBC”，其中A，B，C，D，分别用二进制码00，01，10，11，分别代替，则译出的二进制密码是“0010011010001001101011100110”，共28位。**

**当然希望代码短，消除冗余度，而且唯一。如果把A，B，C，D，的代码分别用0，00，1，01表示，会出现歧义，“01”是译成“D”还是译成“AC”呢？哈夫曼树可以解决这一问题。**

**所要解决的是任一字符的编码都不是另一字符编码的前缀，称这种编码为前缀编码，又称哈夫曼编码。**

**将密码中的不同字符作为二叉树的叶子结点，用字符出现的频度作为叶子结点的权，左右分支分别用0和1（或1和0）表示。**

**例：电文A，B，C，D出现的频度（权值）分别是2，3，8，1。**

**构造的哈夫曼树是：**

0

1

1

0

1

0

**（1）编码方案1（左、右子树分别编吗为0,1，即上图情形）：**

**字母编码：A(000)，B(01)，C(1)，D(001)**

**原电文：“00010111000101110011011”23位即可。**

**由于哈夫曼树不是唯一的，所以这种编码也不是唯一的。如将A和D交换，于是有**

**（2）编码方式2：**

0

1

1

0

1

0

**编码为：A(001)，B(01)，C(1)，D(000)**

**（3）编码方式3（左、右子树分别编为1，0）：**

1

0

0

1

0

1

**编码为：A(111)，B(10)，C(0)，D(110)**

**述评：实现频度越高的字符，编码越短（C），而出现频度越低的字符，编码越长（A，D），这是合理的。**