**树课堂纪要**

written by王保明

# 树基本概念

|  |
| --- |
|  |
| 非线性结构，一个直接前驱，但可能有多个直接后继（1:n） |
| 树的定义具有递归性，即树中还有树 |
| 根 叶子 森林  有序树 无序树  双亲 孩子 兄弟 堂兄弟 祖先 子孙 |
| 结点 结点的度 结点的层次 终端结点 分支结点 |
| 树的度 **所有结点度中的最大值（Max{各结点的度}**  树的深度**指所有结点中最大的层数（Max{各结点的层次}**  (或高度) |
| 关于子树不相交的说明 |

# 树的表示法

|  |
| --- |
| 图形表示法 |
| 广义表表示法 |
| 左孩子－右兄弟表示法 |
| 双亲孩子表示法 |

# 树的逻辑结构

|  |
| --- |
| 一对多（1:n），有多个直接后继（如家谱树、目录树等等），但只有一个根结点，且子树之间互不相交。 |
| 广义表表示法 |
| 左孩子－右兄弟表示法 |
|  |

树的存储

|  |
| --- |
| **顺序存储、链式存储** |
|  |

# 二叉树

## 1、基本概念

|  |
| --- |
| 二叉树的结构最简单，规律性最强 |
| 可以证明，所有树都能转为唯一对应的二叉树，不失一般性 |
| 定义：是n（n≥0）个结点的有限集合，由一个根结点以及两棵互不相交的、分别称为左子树和右子树的二叉树组成 |
| 二叉树性质 |
| 性质1: 在二叉树的第i层上至多有2i-1个结点（i>0） |
| 性质2: 深度为k的二叉树至多有2k-1个结点（k>0） |
| 性质3: 对于任何一棵二叉树，若2度的结点数有n2个，则叶子数（n0）必定为n2＋1 （即n0=n2+1） |
| 满二叉树：一棵深度为*k* 且有2*k* -1个结点的二叉树。  （特点：每层都“充满”了结点） |
| 完全二叉树：深度为*k* 的*，*有*n*个结点的二叉树，当且仅当其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。 |
|  |
| *性质4:* 具有n个结点的完全二叉树的深度必为⎣log2n⎦＋1  *性质5:* 对完全二叉树，若从上至下、从左至右编号，则编号为*i* 的结点，其左孩子编号必为*2i*，其右孩子编号必为*2i＋1*；其双亲的编号必为*i/2*（*i＝1* 时为根,除外） |
| ***二叉树的存储结构*** |
| 一、顺序存储结构  按二叉树的结点“自上而下、从左至右”编号，用一组连续的存储单元存储。  答：一律转为完全二叉树！  讨论：不是完全二叉树怎么办？  方法很简单，将各层空缺处统统补上“虚结点”，其内容为空 |
| **二、链式存储结构**  二叉树结点数据类型定义：  typedef struct node \*tree\_pointer;  typedef struct node {  int data;  tree\_pointer left\_child, right\_child;  } node; |
| **树的三叉链表表示** |

## 2、遍历二叉树

|  |
| --- |
| 二叉树递归的概念，以每个叉为单位 |
| 树的性质确认 |
|  |
| 树的遍历引申 |
|  |
|  |
|  |
|  |

### 二叉树的非递归遍历-中序遍历非递归算法

|  |
| --- |
| **中序 遍历**的几种情况  分析1：什么时候访问根、什么时候访问左子树、什么访问右子树  当左子树为空或者左子树已经访问完毕以后，再访问根  访问完毕根以后，再访问右子树。  分析2：为什么是栈，而不是其他（如队列）。  先走到的后访问、后走到的先访问，显然是栈结构  分析3：结点所有路径情况  步骤1：结点的所有路径情况  如果结点有左子树，该结点入栈；  如果结点没有左子树，访问该结点；  分析3：路径所有情况  如果结点有右子树，重复步骤1；  如果结点没有右子树（结点访问完毕），回退，让栈顶元素出栈，访问栈顶元素，并访问右子树，重复步骤1  如果栈为空，表示遍历结束。  注意：入栈的结点表示，本身没有被访问过，同时右子树也没有被访问过。  分析4：有一个一直往左走入栈的操作 |
|  |
|  |
|  |

## 3、二叉树编程实践

|  |
| --- |
| typedef struct node{  int data;  struct node \*lchild,\*rchild；  } NODE;  NODE \*root; |
| DLR(NODE \*root )  {  if (root) //非空二叉树  {  printf(“%d”,root->data); //访问D  DLR(root->lchild); //递归遍历左子树  DLR(root->rchild); //递归遍历右子树  }  } |
| 中序遍历算法  LDR(NODE \*root)  { if(root !=NULL)  { LDR(root->lchild);  printf(“%d”,root->data);  LDR(root->rchild);  } } |
| 后序遍历算法  LRD (NODE \*root)  {if(root !=NULL)  {LRD(root->lchild);  LRD(root->rchild);  printf(“%d”,root->data);  } } |
| 练习  **例：编写递归算法，计算二叉树中叶子结点的数目** |
| DLR\_CountLeafNum(NODE \*root)//采用中序遍历的递归算法  {  if ( root) //非空二叉树条件，还可写成if(root !=NULL )  { if(!root->lchild&&!root->rchild) //是叶子结点则统计并打印  { sum++; printf("%d\n",root->data); }  DLR\_CountLeafNum(root->lchild); //递归遍历左子树，直到叶子处；  DLR\_CountLeafNum(root->rchild);}//递归遍历右子树，直到叶子处；  } return(0);  } |
|  |

前序遍历

|  |
| --- |
| 思路：利用前序遍历来建树（结点值陆续从键盘输入，用DLR为宜）  Bintree createBTpre( )  { Bintree T; char ch;  scanf(“%c”,&ch);  if(ch==’#’) T=NULL;  else  { T=( Bintree )malloc(sizeof(BinTNode));  T->data=ch;  T->lchild=createBTpre();  T->rchild=createBTpre();  }  return T;  } |
| 后序遍历销毁一个数 |
| 结论：通过中序遍历和先序遍历可以确定一个树  通过中序遍历和后续遍历可以确定一个数。  通过先序遍历和后序遍历确定不了一个数。 |
|  |

## 4、二叉线索树

|  |
| --- |
| 概念  普通二叉树只能找到结点的左右孩子信息，而该结点的直接前驱和直接后继只能在遍历过程中获得。  若可将遍历后对应的有关前驱和后继预存起来，则从第一个结点开始就能很快“顺藤摸瓜”而遍历整个树了。 |
|  |
|  |
|  |
|  |
|  |
| **线索化过程就是在遍历过程（假设是中序遍历）中修改空指针的过程：**  **将空的lchild改为结点的直接前驱；**  **将空的rchild改为结点的直接后继。** |
|  |
| **二叉树线索化算法** |
| void InTreading(BiThrTree p)  //中序遍历进行中序线索化  {  if (p)  {  InThreading(p->lchild); /\*左子树线索化\*/  if (!p->lchild) /\*前驱线索\*/  { p->ltag=1; p->lchild=pre; }  if (!pre->rchild) /\*后继线索\*/  { pre->rtag=1; pre->rchild=p; }  pre=p;  InThreading(p->rchild); /\*右子树线索化\*/    }  } |
| 二叉树线索化遍历算法 |
| 程序注解 (非递归，且不用栈)：  P=T->lchild; //从头结点进入到根结点；  while( p!=T)  { while(p->LTag==link)p=p->lchild; //先找到中序遍历起点  if(!visit(p->data)) return ERROR; //若起点值为空则出错告警  while(p->RTag==Thread ……){ p=p->rchild; Visit(p->data);}  //若有后继标志，则直接提取p->rchild中线索并访问后继结点；  p=p->rchild; //当前结点右域不空或已经找好了后继，则一律从结点的右子树开始重复{ }的全部过程。  }  Return OK; |

## 5、霍夫曼树（哈夫曼树）

|  |
| --- |
| 对于文本”BADCADFEED”的传输而言，因为重复出现的只有  ”ABCDEF”这6个字符，因此可以用下面的方式编码： |
|  |
| 接收方可以根据每3个bit进行一次字符解码的方式还原文本信息。  这样的编码方式需要30个bit位才能表示10个字符  那么当传输一篇500个字符的情报时，需要15000个bit位  在战争年代，这种编码方式对于情报的发送和接受是很低效且容易出错的。  如何提高收发效率？ |
| 要提高效率，必然要从编码方式的改进入手，要避免每个字符都占用相同的bit位 |
|  |
| 准则：任一字符的编码都不是另一个字符编码的前缀！ |
| 霍夫曼树  1.给定n个数值{ v1, v2, …, vn}  2.根据这n个数值构造二叉树集合F  F = { T1, T2, …, Tn}  Ti的数据域为vi，左右子树为空  3.在F中选取两棵根结点的值最小的树作为左右子树构造一棵新的二叉树，这棵二叉树的根结点中的值为左右子树根结点中的值之和  4.在F中删除这两棵子树，并将构造的新二叉树加入F中  5.重复3和4，直到F中只剩下一个树为止。这棵树即霍夫曼树 |
| 假设经过统计ABCDEF在需要传输的报文中出现的概率如下 |
|  |
| 霍夫曼树是一种特殊的二叉树  霍夫曼树应用于信息编码和数据压缩领域  霍夫曼树是现代压缩算法的基础 |
| C:\Users\ASUS\Documents\Tencent Files\1210267211\Image\Group2\AC\KO\ACKO{]@WKCX09P0W}74MWZM.jpg |