文章编号:1671 - 6361(2009)03 - 0013 - 04

二叉树创建算法的研究

邓晨曦, 胡 灿

(湖南环境生物职业技术学院 信息技术系,湖南 衡阳 421005)

摘 要:通过对二叉树四种遍历方法的分析,给出四种建立二叉树的通用算法,以及对这些算法效率的初步分析.解决了二叉树在实际应用中创建的问题,同时,给出的算法对一些高级树型数据结构的研究也具有一定参考价值.参4.

关键词:二叉树;递归;遍历;二叉链表
中图分类号:TP31 文献标识码:A

在计算机科学领域,数据结构以及算法分析是软件开发和程序设计的理论基础.而在实际应用中,二叉树又是一种重要的非线性结构,而且应用很广泛.

在对二叉树的操作中,遍历是一种重要的操作.由于二叉树本身的非线性结构特点,在对二叉树的某个结点进行处理等操作时,前提条件是二叉树必须存在,也就是说计算机内存中应该准确的存储该二叉树.因此,建立二叉树是应用的前提条件,由于篇幅关系,在此介绍的几种算法都是基于二叉链表[1]的存储结构.

1 二叉树遍历及对应关系

二叉树的遍历是指按照一定的规律,搜索并访问各个结点,使得每个结点均被访问一次的过程,其实质是把非线性结构的数据线性化.根据二叉树的特点,常常分为四种遍历形式:先序遍历、中序遍历、后序遍历和层次遍历.

先序遍历(Preorder Traversal):若二叉树为空,则空操作;否则,①访问根结点;②先序遍左子树;③先序遍历右子树.

中序遍历(Inorder Traversal):若二叉树为空,则空操作;否则,①先序遍左子树;②访问根结点;③先序遍历右子树.

后序遍历(Postorder Traversal):若二叉树为空,则空操作;否则,①先序遍左子树;②先序遍历右子树;③访问根结点.

层序遍历(Level Traversal):若二叉树为空,则空操作;否则,按照树的结构,从根开始自上而下,自左而右进,从而实现对每个结点的访问.

依据以上四遍历方式,我们不难证明[1]:先序遍历和中序遍历,中序遍历和后序遍历,层序遍历和中序遍历可以唯一确定一棵二叉树,以下将介绍利用递归、栈或队列、排序树和前驱这四种不同思想创建二叉树的算法.

2 基于递归思想创建二叉树

2.1 已知先序遍历和中序遍历

由先序遍历可知,先序遍历序列的第一个元素为二叉树的根,假设为root;再由中序遍历可得,中序序列中的root元素可以把二叉树划分为左子树和右子树(root左边的子序列为左子树的中序序列,root右边的子序列为右子树的中序序列),再根据左子树中序序列的长度可确定左子树先序序列,同样也可以确定右子树先序序列,最后采用递归方式,从而确定左、右子树.

伪代码如下:

```
HString preod, inod;
//其中preod为先序序列,inod为中序序列,为全局变量
```

收稿日期:2008 - 12 - 25
作者简介:邓晨曦(1980 -),男,湖南衡阳人,硕士生,软件设计师,研究方向:算法分析与软件设计.

```
void PreAndInOrderCreateBiTree1(int p_s,int p_e,int i_s,int i_e,BiTree &T){
// p_s、p_e 为先序序列的起始下标和结束下标,
//i_s、i_e 为中序序列的起始下标和结束下标
//指针 T 为采用二叉链表存储结构的二叉树根
建立根结点 T,并赋值结点的数据域;
查找先序序列的第 p_s 个元素在中序序列的位置,为 j;
//j 就为中序序列对应的根结点的标号
若左子树为空,则 T 的左指针位空;
PreAndInOrderCreateBiTree(p_s+1,p_s+j-i_s,i_s,j-1,T->lchild);
//先序序列左子树的起始标号为 p_s+1,终止标号为 p_s+j-i_s
//中序序列右子树的起始标号为 i_s,终止标号为 j-1
若右子树为空,则 T 的右指针位空;
PreAndInOrderCreateBiTree(p_s+j-i_s+1,p_e,j+1,i_e,T->rchild);
//先序序列左子树的起始标号为 p_s+j-i_s+1,终止标号为 p_e
//中序序列右子树的起始标号为 j+1,终止标号为 i_e
}
```

此算法在实际应用中速度比较快,但是每次递归调用都要查找,为影响效率之关键所在.

2.2 已知后序遍历和中序遍历

利用后序遍历和中序遍历序列建立二叉树的算法和上述算法一致,只是有两点不同:查找中序序列的根结点时,根结点元素为后序序列的终止标号;递归调用时参数不同.

2.3 已知层序遍历和中序遍历

由层序遍历可知,层序遍历序列的第一个元素为二叉树的根,假设为 root;同样,由 root 可把中序序列划分为左右子树,这是需要确定左右子树的对应的层序序列;由于左右子树的层序序列在整个层序序列中已经确定,也就是说左右子树的层序序列的顺序和整棵二叉树的层序序列的顺序没有变化,例如:层序遍历序列中子树的一个结点为 A[i],序列中该结点的上一个 A[i-1],下一个为 A[i+1],在二叉树层序序列中,结点 A[i]的标号一定在 A[i+1]之前,在 A[i-1]之后.因此,可以通过对层序序列的搜索找到作左右子树的层序序列,最后递归调用之即可[3,4].

伪代码如下:

```
HString inod;//中序序列 inod 为全局变量
void LevelAndInOrderCreateBiTree1(HString levelod,int i_s,int i_e,BiTree &T){
// 二叉树的层序序列为 levelod,i_s、i_e 为中序序列的起始下标和结束下标
//lod_l 保存左子树的层序序列,lod_r 保存右子树的层序序列
//j 为中序序列根结点元素位置
建立根结点 T,并赋值结点的数据域;
查找层序序列的第 1 个元素在中序序列的位置,为 j;
// j 为中序序列根结点元素位置
查找左右子树对应的层序序列,分别存放在 lod_l 和 lod_r 中;
根结点在中序的位置 j 为中序的起始位置,则 T 无左子树;
LevelAndInOrderCreateBiTree(lod_l,inod,i_s,j-1,T->lchild);
//左子树的层序为 lod_l,中序起始位置为 i_s,终止位置为 j-1
根结点在中序的位置 j 为中序的结束位置,则 T 无右子树;
LevelAndInOrderCreateBiTree(lod_r,inod,j+1,i_e,T->rchild);
//右子树的层序为 lod_r,中序起始位置为 j+1,终止位置为 i_e
}
```

此算法效率较低,主要每次递归调用都必须执行一个一重循环和一个二重循环,大大降低算法的效率;从空间上来讲,该算法每次递归调用都要开辟长度-1的空间存储左右子树的层序序列,因此空间消耗较大,特别是对于遍历序列较长的二叉树.

3 利用栈或队列创建二叉树

3.1 已知先序遍历和中序遍历

已知先序序列的话,那么从左至右扫描该序列,理论上我们可以利用栈先进后出的特点建立二叉树:先建立根结点,再建立左子树,最后在建立右子树.可是,问题是如何区别某个结点有没有左右子树?我们可以利用中序序列的特点,依次为每个结点找出左子树和右子树的在中序序列中起始下标和终止下标,通过对下标的判断,从而设置该结点的左右孩子域.这里还有一个小问题,怎样区别左右孩子域为空或者还在建立中?我们把每个新建立的结点的左右孩子域指针自身,这样就可解决这个问题.

该算法关键是要注意进栈和出栈条件的判断,并且在对栈顶(top)进行操作的时候要注意 top 和 top - 1 的关系,确定每个结点的左右子树的范围.

3.2 已知后序遍历和中序遍历

对于后序序列,我们知道最后一个元素为二叉树的根,上述算法也是适用的. 只是在扫描后序序列时要从后至前,这样的话就变成了先访问根,再访问右子树,最后访问左子树了,因此,在栈操作中,需要先判断栈顶结点的右孩子域,再判断左孩子域,其它方面在伪代码中基本一致.

3.3 已知层序遍历和中序遍历

我们知道,层序遍历的实质是利用了队列,每次把一个结点的左孩子和右孩子进队列(如果左右孩子存在),然后该结点出队列,再把队列中的出队列的这个结点下一个结点的左孩子和右孩子进队列,依次类推,就能够得到一个层序遍历序列. 因此,如果利用队列,是可以根据曾序序列逆向建立二叉树的,前提条件是通过中序序列知道该结点是否有左右子树. 该算效率较高,影响效率的因素还是查找层序序列某结点在中序序列的位置.

4 基于排序树思想创建二叉树

通过对二叉排序树的分析,中序遍历二叉排序树将得到一个有序序列. 反之,如果我们给每一中序序列的元素加上一个合适的权值,是可以构造一棵二叉树,关键是加上什么权值. 由于中序序列的编号恰好是一个有序的数值序列,因此,中序序列的每个元素的权值可以设为它对应的下标,这样利用构造二叉排序树的方法可以创建一颗二叉树.

4.1 已知先序遍历和中序遍历

在具体实现时,由于给定一个下标,马上就可以访问该下标对应的元素,如果给出一个元素,必须通过查找才能够得到下标. 若在构造二叉树时每插入一个结点都要查找对应的权值(中序下标),那么算法的效率将会大打折扣,因此我们在设计数据结构的时候需要加上一个 weight 域用来存储结点的权值.

伪代码如下:

```
void PreAndInOrderCreateBiTree3 ( HString preod,HString inod,BiTree &T){
//其中 preod 为先序序列,inod 为中序序列,T 为二叉树根结点指针
//建立树根
为指针 p 开辟空间,数据域为先序序列第 0 个元素,左右指针为空;
p 的权值域为 i;T = p;//二叉树根结点
for(i = 1;i < 序列长度;i + + ){//从先序第 1 个位置开始
扫描中序字符,得到先序 i 在中序的下标 j;
p = T;//从根结点开始搜索
为指针 np 开辟空间,数据域为先序序列第 i 个元素,左右指针为空;
np 的权值域为 j;
while(true){//建立二叉树
如果 np - > data 在中序的位置小于 p - > weight,则 np 在 p 的左树
如果 p 的左树为空,则 np 作为 p 的左树插入
否则 p = p - > lchild;//p 的左树为不空,则继续从左树查找
否则 np 在 p 的右树
如果 p 的右树为空,则 np 作为 p 的右树插入
p = p - > rchild;//p 的右树为不空,则继续从右树查找
}}}
```

该算法使用了两重循环,时间主要还是消耗在查找上,因为查找最坏情况下是需要搜索整个中序序列的,而建立二叉树的插入查找过程较快. 总体来说,本算法效率是比较高的.

4.2 已知后序遍历和中序遍历

依然采用上述算法,只是在扫描后序序列时从后最后一个序列元素开始向前扫描.

4.3 已知层序遍历和中序遍历

同样算法,伪代码基本一致,故不详述.

5 基于前驱技术创建二叉树

一个二叉树中序遍历序列中,如果知道根结点的下标,那么,该结点中序序列左边的结点为此结点的左子树,右边的结点为右子树. 如果把左子树结点的前驱用正整数表示,右子树结点的前驱用负整数表示. 如 A[0],A 表示结点,假设下标为 7,0 表示该结点的前驱结点为在中序序列中下标为 0 的结点,那么 B[7]表示表示 A 结点的左孩子为 B 结点,同样 C[-7]表示 A 结点的右孩子为 C 结点,依次类推. 由于先序遍历序列是先访问根结点的,而层序遍历的实质也是优先访问根结点,同样后序遍历序列的逆序也先访问根结点(对于树或者子树),因此,采用递推的方法是可以得到每个结点在中序遍历中的前驱. 最后根据中序遍历序列建立一个连续的二叉链表,依次设定数据域和左右孩子指针域即可.

5.1 已知层序遍历和中序遍历

为了区别 -0 和 +0,我们在实现时把前驱都

做一个映射:前驱为负就减去 1,前驱为正就加上 1,这样就可以避免 +0 和 -0 带来的问题.

伪代码如下:

void LevelAndInOrderCreateBiTree3 (HString levelod,HString inod,BiTree &T) {

//其中 level 为层序序列,inod 为中序序列,T 为二叉树根结点指针

//为区别一个结点前驱是否已确定完毕,用 flag 数组表示,同时也为查找方便

分别为 flag 和 parent 开辟空间,空间长度为遍历序列长度;

初始化 parent,flag[i];

//flag[i] =0,表示下标为 i 的中序结点未建立前驱,1 表示已建立

//parent[i] =0,根结点的前驱结点设为 0,其它结点初始前驱也为 0

for(i =0;i < 遍历序列长度;i + +) {//求前驱,保存在 parent 中

for(j =0;j < 遍历序列长度;j + +)

查找层序第 i 个结点在中序序列的位置 j,同时设 flag[j] =1;

向前查找,如果下标为 k 的结点的前驱和树(子树)根结点相等,设为 j +1;

向后查找,如果下标为 k 的结点的前驱和树(子树)根结点相等,设为 -j -1;

}

为 tree 开辟遍历序列长度 +1 个空间;

for(i =0;i < 遍历序列长度;i + +) {//初始化 tree

数据域依次为中序序列元素;

tree[i + 1]. lchild = tree[i + 1]. rchild = NULL;//初始化左右孩子指针域

}

for(i =1;i < = 遍历序列长度;i + +) {//转换为二叉链表

if(前驱为正) 该结点为双亲结点的左孩子;

else if(/前驱为负) 该结点为双亲结点的右孩子;

else 前驱为 0,则是根结点;

} }

此算法效率较高,影响效率的主要部分是查找结点在中序序列中的下标,但是查找是必需的,若有更高效的查找算法,该算法性能可近一步提高.

5.2 已知后序遍历和中序遍历

依然采用上述算法,只是在扫描后序序列时从后最后一个序列元素开始向前扫描.

5.3 已知先序遍历和中序遍历

同样的算法,伪代码基本相同,不详述.

6 结束语

遍历在二叉树的应用中占有重要的地位,以上给出的几种不同创建二叉树算法都是利用了二叉树遍历序列的性质,从而设计出来的. 另外,对于高级数据结构,如红黑树、AA - 树、AVL 树等的研究,以上算法都有一定实用价值.

参考文献:

[1] 严蔚敏,吴伟民. 数据结构(C 语言版)[M]. 北京:清华大学出版社,1997.

[2] 盛 魁. 二叉树的遍历探究与应用[M]. 北京:电脑知识与技术,2008.

[3] Coremen T H. 算法导论[M]. 北京:机械工业出版社,2006.

[4] 田小梅,龚 静. 遗传算法改进措施[J]. 湖南环境生物职业技术学院学报,2008,14(1):24 - 27.

Study on the Algorithm of Establishing Binary Tree

DENG Chen - xi, HU Can

(Department of Information Technology, Hunan Environment - Biological Polytechnic, Hengyang 421005, China)

Abstract: This paper analyzed four kinds of traversal of binary tree, provided four general algorithms to establish binary tree and preliminary analysis on these algorithms efficiency. The problem for established binary tree in practical applications has been solved; meanwhile, these algorithms have reference value for the research of some advanced tree type data structure. 4refs.

Key words: binary tree; recursion; traversal; binary list