《数据结构》非递归后序遍历二叉树算法的探讨与分析

无锡商业职业技术学院信息工程系　程冠琦

[摘　要]判断根结点何时出栈是非递归后序遍历二叉树算法中要解决的关键问题，大多数算法均采用在二叉树结点的存储结构中增加一个附加标志位的方法来实现，但同时也增大了存储空间的开销。本文对其进行了改进和完善，给出了一种设置同步标志栈的方法，解决了存储空间开销的问题。

[关键词]后序遍历二叉树　附加标志位　同步标志栈

1、概述

《数据结构》中二叉树的遍历方法是实现二叉树各种算法的基础，对于一棵二叉树来说，有先序遍历、中序遍历、后序遍历三种基本方法。在非递归实现二叉树的遍历算法中，需要用到栈这种数据结构。使用栈，其一是为了保存根结点信息，在访问完左子树后能够通过栈找到右子树进行访问；其二是二叉树的结构本身具有递归的特点，每一个结点在其右子树未被访问之前都要保存其信息，这符合栈的特点(后进先出)。

三种遍历非递归算法中，不同之处在于根结点出栈的时机问题。

先序遍历过程中，我们的操作顺序是：根结点进栈——访问根结点——访问左子树——找到右子树(根结点出栈)。

中序遍历过程中，我们的操作顺序是：根结点进栈——访问左子树——访问根结点——找到右子树(根结点出栈)。

后序遍历过程中，我们的操作顺序是：根结点进栈——访问左子树——访问右子树——访问根结点(根结点出栈)。

从三种遍历算法中的操作顺序中我们可以看到，在先序遍历和中序遍历算法中，根结点出栈均在右子树被访问之前，而后序遍历算法中，根结点出栈在右子树被访问之后。

2、非递归后序遍历算法的解决方案

2.1 附加标志位的使用

从以上分析中我们可以看出，非递归后序遍历二叉树算法中要解决的关键问题是判断根结点何时出栈，这个问题对于先序遍历和中序遍历非常好解决，但对于后序遍历来说，需要判断的是到底是从左子树返回还是从右子树返回，大多数算法采用的方法是在二叉树结点的存储结构中增加一个附加标志位，即作如下定义：

```
typedef struct bttree
{elem data;
struct bttree * lchild;
struct bttree * rchild;
int flag; /* 新增加的附加标志位，取值为 0,1,2 */
}bttree;
```

对于一个二叉树结点来说，有三种状态：1、其左、右子树均未被访问；2、左子树已经访问完，右子树还未访问；3、其左、右子树均已访问完毕。后序遍历中要求在第一种状态下根结点进栈，在第三种状态下根结点出栈。我们可以用 0,1,2 分别代表这三种状态，因此可以通过标志位的取值情况来判断何时进栈，何时出栈。

```
void posttravel(bttree *h)
{bttree *p=h, *s[20]; int top=0;
while(top>=0)
{if(p!=NULL)
{s[top]=p; top++;
p->flag=0; /* 其左、右子树均未被访问，设置标志为 0 */
p=p->lchild; }
else {
if(top>0)
{if(s[top-1]->flag==0) /* 标志为 0，表示从左子树返回 */
{s[top-1]->flag=1; /* 左子树已访问，右子树还未访问。设置标志值为 1 */
p=s[top-1]->rchild; }
else if(s[top-1]->flag==1) /* 标志为 1，表示从右子树返回 */
{s[top-1]->flag=2; /* 左右子树均已访问。设置标志值为 2 */
printf("%c--",s[top1-1]->data);
top--; } }
else
top--; }}}
```

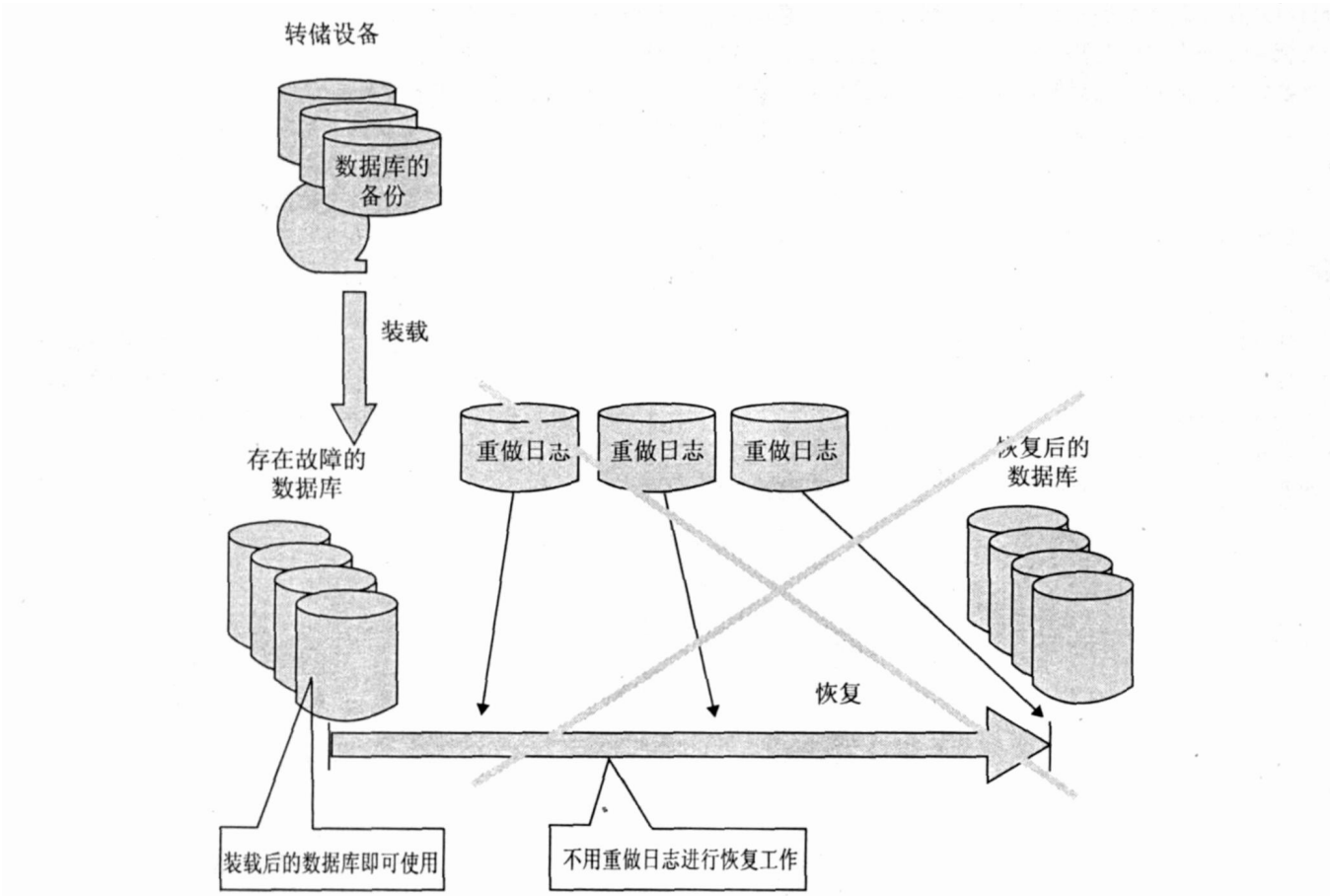
2.2 算法的改进——同步标志栈的使用

大多数算法在实现时均引入一个栈，本文要介绍的算法中多引入一个同步标志栈，该栈的作用是保存前一个栈每个结点的状态，即我们在上面讨论的三种状态，因此，该栈的类型应是整数类型，且每个元素只有 0、1、2 三种取值。并且，该栈和前一个栈要保持同步状态，即进、出栈要同步。这样我们就可以依据同步标志栈元素的状态确定下一步的操作。

C 语言源程序如下：

```
/* 后序非递归遍历二叉树 */
/* 算法思想：设置标志栈，用来判断是从左子树返回还是从右子树返回
标志为 0，表示左右子树均未访问；标志为 1，表示左子树已访问，右子树未访问；
标志为 2，表示左右子树均已经访问 */
void postorder2(bttree *h)
{
bttree *p=h; bttree *s[20];
int flag[20]; int top1=0; int top2=0;
while(top1>=0)
{if(p!=NULL)
{s[top1]=p;
top1++;
/* 标志域同时进栈 */
flag[top2]=0;
```


(下转第 210 页)



参考文献

[1]李海波·Oracle 数据库的安全及备份恢复[J]·电脑知识与技术,2004,(11):13-15.

[2]滕永昌等·Oracle 数据库系统管理[M]·清华大学出版社,2003,7.

[3]袁勤勇等·Oracle8.1.6 管理员指南[M]·北京:北京希望电子出版社,2000.

[4]Oracle9i 数据库的备份与恢复[M]·北京:电子工业出版社,2003.

[5]Oracle 数据库管理员手册[M]·北京:电子工业出版社,2002.

[6]http://asktom·oracle·com[EB]·

(上接第 207 页)

```
top2++;
p=p->lchild;
}
else
{if(top1>0)
{/*从左子树返回*/
if(flag[top2-1]==0)
{/*设置标志值*/
flag[top2-1]=1;
p=s[top1-1]->rchild;
}
/*从右子树返回*/
else if(flag[top2-1]==1)
{/*设置标志值*/
flag[top2-1]=2;
printf("%c--",s[top1-1]->data);
/*同时退栈*/
top1--; top2--;
}}
else
top1--;
}}}
```

3、算法分析

在使用标志位的算法中,对每一个二叉树结点都要多分配一个标志位的空间,对于无右子树的结点和孩子结点,该标志位基本无用;另外,这个标志位仅仅在后序遍历中有用,因此,从算法的空间复杂度上来说,该方法并不是一个好方法,存储空间浪费比较严重。笔者在使用 visual c++ 6.0 测试中发现,后序遍历一棵含有 20 个结点的二叉树,就要多出近 80 个字节的空间。

在使用同步标志栈的算法中,由于采用的同步标志栈属于局部变量,当后序遍历二叉树完毕后,该栈将自动收回,从空间复杂度的角度上来说,该算法是令人满意的。

参考文献

[1]许卓群,张乃孝,杨冬青·唐世渭《数据结构》,高等教育出版社,1988 年

[2]傅清祥,王晓东·算法与数据结构,电子工业出版社,1993

[3]严蔚敏,吴伟民·数据结构·清华大学出版社·1997