EXPLORE A NEW WAY TO CONVERT A RECURSION ALGORITHM INTO A NON-RECURSION ALGORITHM

Yongping Gao *, Fenfen Guan
School of Information Technology East China Institue of Technology, FuZhou, JiangXi, China
* *Corresponding author, School of Information Technology East China Institue of Technology, FuZhou, JiangXi, China Email: ypgao_ypgao@163.com*

Abstract: This article discusses how to use queue to make non-recursion algorithm of binary link tree. As for a general binary tree, if we adopt sequence storage, firstly we should extend it first into a complete binary tree, secondly we store it to a temporary queue according to the sequence of up-down and left-right. On the basis of the properties of the complete binary tree and the queue, if we can confirm every element in the queue, then we can find the left and right children of the element in the queue. When we recur these steps, we can create a binary link bit tree. This algorithm enriches the method from recursion to non-recursion.

Keywords: sequence storage, bit link tree, queue, recursion, non-recursion

1. INTRODUCTION

There are two main problems in the designing program of recursion algorithm. On the one hand, not every language backs up recursion. On the other hand, it will take a longer time to finish a recursion program than a non-recursion one, and when there are too many layers of recursion, the stack overflow will appear in the running results (Xiaoyun Guo, 2004).

Gao, Y. and Guan, F., 2008, in IFIP International Federation for Information Processing, Volume 258; Computer and Computing Technologies in Agriculture, Vol. 1; Daoliang Li; (Boston: Springer), pp. 187–193.

If we convert a recursion algorithm into a related non-recursion one, the program can be implemented more efficiently. There are many different methods to transform recursion algorithm to non-recursion one. There are some familiar methods to solve different problems. For example, a recursion algorithm can be converted into a non-recursion one by stacks (Zhenyuan Zhu, 2003), depth-first search and N order Legendre polynomials (Zhong Li, 2001). As for the tower of Hanoi, we usually adopt Inorder-traversing binary tree to convert a recursion algorithm into a non-recursion one (Changbao, 2002).

In order to convert a recursion algorithm into a non-recursion one of a bit link tree, the article attempts to adopt another method—using a temporary queue and two points to finish the conversion.

2. PROBLEM FORMULATION

2.1 Introduction of the problem

If we apply the binary link tree to practical problems, we have to know how to produce a binary link tree. The algorithm adopted by many related books is to extend a binary tree first, and then input some information of node according to preorder traversal to create a binary link tree by recursion (Yuli Yuan, 2006). The detailed function is as following (Weimin Yan, 1997):

```
PBinTreeNode createRest_BTree()
/* To create a bit tree from root node by recursion */
{ PBinTreeNode  pbnode;
char ch;
  scanf("%c",&ch);
  if(ch=='@') pbnode=NULL;
  else
  { pbnode = (PBinTreeNode )malloc(sizeof(struct BinTreeNode));
    if(pbnode==NULL)
    {  printf("Out of space!\n");
       return pbnode;
    }
  pbnode->info=ch;
    pbnode->llink=createRest_BTree();  /*to create left bit tree*/
    pbnode->rlink=createRest_BTree();  /* to create right bit tree */
  }
  return pbnode;
}
```

As it is known to all, a recursion function is called by a stack data structure (Naixiao Zhang, 2002). When we teach such recursion algorithm, we find it difficult for many students to understand the process of creating a binary tree (Chunbao Li, 2002). To help students to understand the algorithm, we have to adopt a non-recursion algorithm to create a binary link tree. Meantime, we can break the rule—recursion of a function should be realized by a stack data structure (Fuying Wu, 2003), but we can change recursion algorithm into non recursion one by other data structures such as queue.

2.2 An analysis of the problem

As it is known by us, we store a complete binary tree according to the sequence of up-down and left-right, by this way; we use a series of memory in order to store the binary tree nodes. The fig. 1 and fig. 2 illustrate this.

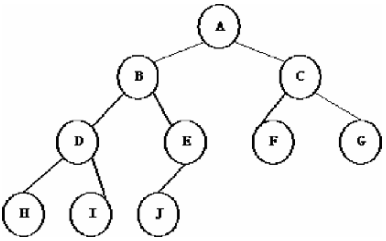

Fig. 1. A complete binary tree

A	B	C	D	E	F	G	H	I	J	
1	2	3	4	5	6	7	8	9	10	

Fig. 2. In order to store a complete binary tree

As for a general binary tree, if we adopt sequence storage, we should extend it first into a complete binary tree, and then store it according to the sequence of up-down and left-right as the following fig. 3.

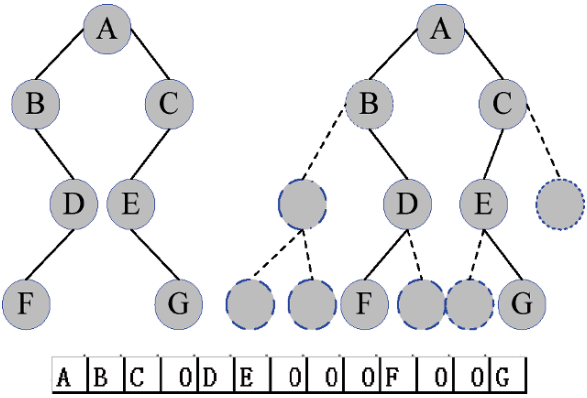

Fig. 3. In order to store a general binary tree

The binary tree stored by this way has the qualities of a complete binary tree and the detailed description is the following (Yongping Gao, 2005).

If we number a binary tree that has x nodes beginning with 1, then any node i (1<= i<=n) will (Zhili Tang, 2001):

1) if i=1, then i is root node; if i>1, then ⌊i/2⌋ is its parent node.

2) if 2i <=n, then 2i is its left children node; if 2i>n, then it has no left children.

3) if 2i+1 <= n, then 2i+1 is the right children node; if 2i+1 > n, then it has no right children.

From these qualities we can know that we can find the positions of one node's left and right children if we are given the node's position in an array. For example, node A is the first number in an array, if its left and right children exist, then the number of its left children is 2, and the number of its right children is 3 (Tao Zhu , 2005).

3. PROBLEM SOLUTION

Fig. 4. In order to store a binary tree after which is extend a complete binary tree

From the above analysis, we propose that we can create a binary link tree by the sequence storage of a binary tree.

We can regard the sequence storage of a binary tree as a queue, and add two pointers — one pointer f refers to the head pointer of the queue, the other pointer r refers to the tail one of the queue (Shuqun Gong, 2007), as what the fig. 4 illustrates.

1) Extend a binary tree into a complete binary tree.

2) Store the node address of the complete binary tree, rather than value of the node, in the unit of memory by the sequence of up-down and left-right. If the node is empty, then its address is null. When we store a node, we will add 1 to the value of tail pointer of the queue.

3) When the number element i comes out of the queue, we will add 1 to the value of head pointer and find out the left and right children of this element (if their children exist, the value of left children will be 2*i and the value of the right children will be 2*i+1). For instance, when the first node A comes out of the queue, the value of its left children is 2 (or the element B) and the value of its right children is 3 (or the element C).

4) Repeat the third step until there is no any element in the queue, then the head pointer equals to the tail pointer.

We can realize the function by the following codes:

```
typedef struct BiTNode  /*the definition of node type*/
{     elemtype data;
      struct BiTNode *lchild, *rchild;
 }BiTNode,*BiTree;
BiTree  CreateBiTree()
{  /*to create a binary tree by using a queue according to the sequence of the up-down and left-right*/
    char ch[20];
BiTree head, p, tm[20];/*head stands for storage of the head pointer of a binary tree, and tm stands for a queue*/
int i=0,n,f=1,r=1; /*f is the head pointer of the queue and r is the tail pointer*/
     gets(ch);   /*store the extended complete binary tree according to the sequence of up-down and left-right in the variable ch*/
n=strlen(ch);
   while(1)
   {  if(ch[i]==‘\0’ )  break;
```

```
    if(ch[i]=='*') p=NULL; /*if the value of ch[i] is empty node, then we will store null in
the queue*/
    else
     {p=(BiTree)malloc(sizeof(struct BiTNode));
      if(p==NULL)
{printf("not enough space!"); return NULL; }
        p->data=ch[i]; p->lchild=NULL; p->rchild=NULL;
     }
      tm[r]=p;  /*we put the first address of the node in the queue*/
       r++; /*we add 1 to the tail pointer*/
  if(i==0) head=p; /*we use head to store the address of head pointer*/
       i++;
   }
 while(f!=r) /*we recur the following steps if the queue is not empty*/
 { p=tm[f];
 if(p!=NULL)
 {  if(2*f<=n) /*we can judge whether the left children exist*/
   p->lchild=tm[2*f] ; /*then p->lchild refers to the left children*/
       if(2*f+1<=n) /*we can judge whether the right children exist*/
       p->rchild=tm[2*f+1]; /* p->rchild refers to the right children*/
 }
 f++; /*the head pointer of the queue will be added 1*/
 }
  return head;
 }
```

We can get the result after we run the program:

Input: ABC*DE***F**G

The output of preorder traversal will be (Wanlan Tian, 2003): A B D F C E G

The output of in order traversal will be (Zhong Li, 2003): B F D A E G C

Based on the results of the preorder and in order traversal, we can get only one binary tree (Chuanhong Chen, 2005), and the binary tree is in the fig. 4 and on the other hand, the algorithm we provided above is right (Yuansong Li, 2003).

4. CONCLUSION

In this paper, we have changed successfully a recursion function into non-recursion one by a queue. Thus it is very helpful for students to understand and master a binary link tree. This shows that in some occasions we can convert a recursion function into a non-recursion one by some others instead of by stacks, and this enriches the ways of converting a recursion algorithm into a non-recursion one.

REFERENCES

Changbao Shu, Zhenhai Liu, A Non-recursion Algorithm about Tower of Hanoi, Computer Development & Applications, Vol. 15, No. 11, 2002, pp. 33-34.

Chuanhong Chen, Wuying Shen, The Research and Application of Traversing Binary Tree, Journal of Xiaogan University, Vol. 25, No. 3, 2005, pp. 72-73.

Chunbao Li, Hui Zeng, Zhimin Zhang, Praxis of Program of Data Structure, The Press of Qinghua University, 2002.

Fuying Wu, Luosheng Tan, Mingwen Wang, Nonrecursive Algorithm of Inorder Traversing Sequential Storage Full Binary, Journal of Jiangxi Normal University (Natural Sciences Edition), Vol. 27, No. 4, 2003, pp. 372-375.

Naixiao Zhang, Algorithm and Data Structure, Higher Education Press, 2002.

Shuqun Gong, Yu Ren, Weiwei Chen, The Front and Rear Pointer Design of Circular Queue GONG, Modem Computer, No. 2, 2007, pp. 17-20.

Tao Zhu. Judging Fully Binary Tree on the Basis of Traversing Binary Tree [J]. Journal of Honghe University , Vol. 3, No. 6, 2005, pp. 47-48.

Wanlan Tian, The Discussion of Inorder-traversing and Postorder-traversing Binary-tree with Recursive Algorithm, Journal of Liangshan University, Vol. 5, No. 3, 2003, pp. 3-3.

Weimin Yan, Weimin Wu, Data Structure (C Language), The Press of Qinghua University, 1997.

Xiaoyun Guo, Non-recursive Simulation on Recursive Function, Journal of Xuzhou Normal University (Natural Science Edition), Vol. 22, No. 1, 2004, pp. 40-42.

Yongping Gao, Shumin Zhou, Use Stack to Make the Non-recursion Algorithm of Bit Link Tree, Computer Era, No. 11, 2005, pp. 24-25.

Yuansong Li, New Method of Traversing Binary Tree, Journal of Sichuan University of Science & Engineering (Natural Sicence Edition), Vol. 16, No. 4, 2003, pp. 45-46.

Yuli Yuan, Ling Hu, The Teaching Analysis of Inorder Traversing Binary Tree in the Data Structure, Journal of Neijiang Teachers College, Vol. 21, No. 4, 2006, pp. 109-111.

Zhenyuan Zhu, Cheng Zhu, Non-recursive Implementation of Recursive Algorithm, Mini-Micro Systems, Vol. 24, No. 3, 2003, pp. 567-570.

Zhili Tang, Methods for uniquely determining a tree or a binary tree based on its traversal sequences, Mini-Micro Systems, Vol. 22, No. 8, 2001, pp. 985-988.

Zhong Li, Recursive Algorithm Transform into Non-recursive Algorithm, Computer Science, Vol. 28, No. 8, 2001, pp. 96-98.

Zhong Li, Lin Meng, Dehui Yin, A discussion of postorder-traverse binary tree with no-recursive algorithm, Journal of Southwest University for Nationalities (Natrual Science Edition), Vol. 29, No. 5, 2003, pp. 537-538.