Journal of Computer and Communications, 2016, 4, 42-47
Published Online May 2016 in SciRes. http://www.scirp.org/journal/jcc
http://dx.doi.org/10.4236/jcc.2016.47006

Binary Tree's Recursion Traversal Algorithm and Its Improvement

Hua Li

Department of Information Engineering, Hangzhou Polytechnic College, Hangzhou, China
Email: zjxulhm@163.com

Received 15 March 2016; accepted 22 May 2016; published 25 May 2016

Abstract

Binary tree is a very important data structure in computer science. Some major properties are discussed. Both recursive and non-recursive traversal methods of binary tree are discussed in detail. Some improvements in programming are proposed.

Keywords

Binary Tree, Traversal, Stack

1. Introduction

Binary tree is a very important data structure in which each node has at most two children, which are referred to as the left child and the right child. In computing, binary trees are seldom used solely for their structure. Much more typical is to define a labeling function on the nodes, which associates some value to each node. Binary trees labelled this way are used to implement binary search trees and binary heaps, and are used for efficient searching and sorting. The designation of non-root nodes as left or right child presents matters in some of these applications, even when there is only one child, and it is particularly significant in binary search trees [1]. In mathematics, what is termed binary tree can vary significantly from author to author. Some use the definition commonly used in computer science, but others define it as every non-leaf having exactly two children and don't necessarily order (as left/right) the children either [2]. The basic structure of the binary tree can be summarized as follows [3].

- Degree of node: The number of children of a node is denoted as the degree of the node.
- Height of the tree: A tree's maximum number of level is denoted as tree height (or depth).
- The i level of binary tree ($i \geq 1$) has up to $2^{(i-1)}$ nodes.
- Binary tree of depth k has at most $2^k - 1$ nodes ($k \geq 1$).
- To any binary tree, if the number of leaf node is a and the number of nodes of degree 2 is b, then $a = b + 1$.

How to cite this paper: Li, H. (2016) Binary Tree's Recursion Traversal Algorithm and Its Improvement. *Journal of Computer and Communications*, **4**, 42-47. http://dx.doi.org/10.4236/jcc.2016.47006

- The depth of complete binary tree which have n nodes is (log (2^n)) + 1.
- For complete binary tree with n nodes and nodes hierarchically from top to bottom, from left to right are encoded, then to any node a ($1 \le a \le n$) has if a = 1, then node a is the root of binary tree and have no parent, if a > 1,then its parent is a/2 (Rounded down). If 2a > n, then node i have no child node, else its left child node is 2a. If 2a + 1 > n, then a have no right child node, else its right child node is 2a + 1.
- If the binary tree which degree is n has 2^n − 1 nodes, then it is called a full binary tree. Full binary tree is also called complete binary tree.
- For complete binary tree, the number of node which is 1 degree is only possible to 1 or 0.
- For any tree, the total number of nodes = the sum of each node number + 1.

2. Why Use a Binary Tree Traversal and Its Practical Application

In binary tree, we often need to find the binary tree node that has some certain characteristics, or need to find all the nodes and process them. For example, based on digital image disorder binary tree traversal—A digital image scrambling method based on binary tree traversal, and discussed the periodic scrambling method and inverse transform. The method is simple and easy to operate, and suitable for images of any sizes. And it has good scrambling effect and great scrambling cycle. Under certain attacks, scrambled image can recover the original image. To some extent, it can meet the digital image encryption and hidden robustness requirements, and using binary tree traversal to expand the convex outer surface of the polyhedron (It can help some production construction). All of these require a binary tree traversal. However, the binary tree is a nonlinear structure, and each node may have two trees. So, we need to find rules that can make all nodes of a binary tree are arranged on a linear queue. So binary tree traversal of each node is in accordance with a path to access binary tree, and each note can be visited only once. Thus, the binary tree node is accessed sequentially formed by a linear sequence, whose result is that each node on the binary tree can be accessed more easily [4].

3. Binary Tree's Recursive Traversal Algorithm and Description [5]

Since the tree traversal rule is recursive, recursive traversal of a binary tree is very popular and convenient. Thus, according to the child-first traversal of a binary tree rules, there are three recursive traversal orders:

1) Preorder: access root node, traverse the left subtree, traverse the right subtree

2) Inorder: traverse the left subtree, access root node, traverse the right subtree

3) Postorder: traverse the left subtree, traverse the right subtree, access root node

It can be summed up as some rules. First, preorder traversal of the first root node is the root node, while postorder traversal of the last root node is the root node.

Second, the last root node of preorder traversal is the rightmost child node of the right subtree, the last node of inorder traversal is the most right node of the root node right subtree. Third, leftmost root node inorder traversal first node to the root of the left subtree, postorder traversal is the first node as a left subtree the left child node.

From the above rules, we can draw the following inferences. The whole tree sort can be derived through the preorder traversal and the postorder traversal. Inorder traversal and postorder traversal can determine a binary tree. Preorder traversal and postorder traversal cannot determine a binary tree by themselves.

Let's write a first binary tree's Preorder traversal, Inorder traversal and Postorder traversal

```
Public class BinaryTree<T> implements BinaryTTree<T>{
Public BinaryNode<T>root;
Public BinaryTree(){this.root=null;}
Public Boolean isEmpty(){return this.root==null;}
}
Public void preOrder(){ // Preorder traversal
        PreOrder(root);// Call the recursive method to preorder traversal
   }
Public void preorder(BinaryNode<T>    p){
if(p!=null)
{
    System.out.print(p.data.toString()+"  ");//access to root node
```

```
    preOrder(p.left);// According to preorder traversal traverse left subtree，then recursive call
    preorder(p.right);// According to preorder traversal traverse right subtree，then recursive call
}
    Public void inOrder(){//inorder traversal
            inOrder(root);
}
    Public void inOrder(BinaryNode<T> p)
    {
        If(p!=null)
        {
            inOrder(p.left);
            System.out.print(p.data.toString()+”  ”);
            inOrder(p.right);
        }
    }
  Public void postOrder(){//postorder traversal
            postOrder(root);
}
  Public void postOrder(BinaryNode<T> p)
    {
        If(p!=null)
        {
            postOrder(p.left);
            postOrder(p.right);
            System.out.print(p.data.toString()+”  ”);
        }
    }
```

The above algorithm is based on the definition of the root node p to determine the entire recursive method, The root node p will be refined in each recursive And then find a child node p, and child node is priority. If there is child node, then continue to search until there has no child node, we can output the nodes which have searched before in order.

4. Another Algorithm of Binary Tree Traversal Algorithm—Non-Recursive Calls Algorithm

The binary trees Preorder, Inorder and Postorder all belong to recursive algorithm. When the chain store of binary tree structure is given, the programming language with the recursive function can easily achieve the above algorithm. But recursive algorithm must have parameters. And it should distinguish multiple processing ways through different practical parameters. The method described above is regarding the node p as an argument. When p is a pointer which points to a different node, it means different trees. Accordingly, using different ways to call p is a matter of different traversing.

So the binary tree’s non-traversal algorithm needs to build a stack to store traversal. Its algorithm is described as follows: Setting an empty stack; Node p from the binary tree root node, when p is not empty or not empty stack, do the following cycle, and finish the binary tree until the stack is empty.

- If p is not empty, showing just arrived p junction, put p junction stack, enter p left subtree.
- If p is empty, but the stack is not empty, and finish the route, we need to return to find another path. The node returns just after the last point, as long as the stack find a node of the p-point he could enter the right subtree.

Thus, we can launch a non-recursive algorithm Binary Tree

1) Non-recursive of Preorder traversal’s implement

In the following algorithm, binary tree stored by binary linked list, Create an array stack [Pointer] in order to achieve Stack, top in stack is used to indicate the current location of the stack.

```
void inOrder（BiTree bt）
{/* Non-recursive preorder binary tree */
   BiTree stack[Point],t;
   int top;
   if (bt==NULL) return;
   top=0;
   t=bt;
   while(!(t==NULL&&top==0))
       { while(t!=NULL)
              { Visite(t.data);       /* Data field access node */
                if (top<MAXNODE-1)    /* The current push pointer p */
                 { stack[top]=t;
                    top++;
                  }
                else { printf(“Stack Overflow”);
                          return;
                        }
                 t=t.leftchild;                /* Pointer to the left child of p */
                 }
              if (top<=0) return;          /* stack empty, then over*/
              else{ top--;
                       t=stack[top];                    /* Pop the top element from the stack */
                       Visite(t.data);       /* access node data field */
t=t.rightchild;                      /* Pointer to the right child node p */
                    }
         }
}
```

2) Non-recursive of inorder traversal's implement

The non-recursive of inorder traversal's come true, simply preorder traversal non-recursive algorithm in the Visite (t.data) moved to between t = stack [top] and t = t.rightchild.

3) Non-recursive of Postorder traversal's implement

In the following algorithm, array stack [Pointer] is used to achieve the overall structure of the stack. When the pointer variable p points to the current node to be processed, and node t is used to indicate the position of the current stack initial value of −1, sign integer variable amount can confirm that whether the node p has other children.

```
void Tree(Tree    bt)
/* Non-recursive of Postorder traversal bt*/
{ stacktype stack[Point];
   Tree p;
   int t,sign;
   if (bt==NULL) return;
   top=-1                    /* The default value of -1 means no stack location element within the stack */
   p=bt;
   while (!(p==NULL && t==-1))
     { if (p!=NULL)                  /* The first node into the stack */
         { t++;
           stack[t].link=t;       /*into stack*/
           stack[t].flag=1;
           p=p.lchild;               /* Get the node left child node*/
         }
         else { p=stack[t].link;
```

```
                sign=stack[t].flag;
                t--;
                if (sign==1)                    /*if exist right child node*/
                  {top++;
                   stack[t].link=p;
                   stack[t].flag=2;             /* Marking the second time out of the stack */
                   p=p.rightchild;
                  }
                else { Visite(p.data);          /* If not, the direct access to the node data field values */
                       p=NULL;
                      }
               }
        }
}
```

The nature of the entire non-recursive algorithm is through the establishment of a stack to store each node to traverse down the node element, and it can sequentially output according to the characteristics of the stack. In a recursive algorithm by means of a recursive loop we will find each of the nodes is in the whole recursive loop, but recursion can be removed at any time to view through putting the element r into the stack one by one, Although this way is more cumbersome than the recursive algorithm, it reduces more computing time and system resources. And it is easier to see the whole nature which traverse through the program algorithm.

5. Improvement of Non-Recursive Algorithm

According to the above non-recursive algorithm, it can be seen that before preorder traversal and time complexity is O (n), then preorder compared to O (n^2), algorithm is relatively complicated. So what can be improved so that the complexity and the time when the same preorder it? Let's first look postorder traversal non-recursive algorithm description, in process of postorder traversal non-recursive algorithm, to ensure the left and right child nodes are traversed, and left nodes must be traversed before the right in order to traverse the root node, we usually use the same array and stack on a non-recursive algorithm as described above, but they are more cumbersome. The postorder traversal of binary Tree determines its complexity of non-recursive algorithm design. If we blindly consider the issue from the "left and right root nodes" perspective, the created algorithm is undoubtedly very complex. If we change the angle of thoughts, the binary operation after preorder access order reversed, that is the "root points to the right-to-left node" that we are familiar with and very simple "preorder traversal". The difference is that the preorder traversal of a binary tree is the first access node and then left and right node access, and the "preorder traversal" here is the first visit and then visit the left node and right node. Here we don't discuss the left node and right node of order, but this idea gave us space to think and provides another way of thinking for the design of the non-recursive algorithm of binary tree traversal. So we can use preorder traversal to output result, then use the result to reverse output. Then the result becomes the result of postorder traversal. We just need to set up a non-recursive traversal more than a usual stack to store the previous preorder traversal non-recursive node obtained from before and then output it. Then there is the improved postorder traversal non-recursive algorithm program that I used to write with c language.

```
Status PostOrderTraverse(BiTree T,Status(* visit)(TElemType e))
{
 /* According to preorder traversal traversal binary thinking first */
 InitStack(C1);InitStack(C2);/* Initialize the stack */
 If(T)Push(C1,T);
 While (！ StackEmpty(C1))
 {
 Pop(C1,p);
 Push(C2,p);
 If(p->lchild) Push(C1,p->lchild);// The left node stack
```

```
If(p->rchild) Push(C1,p->lchild);// The right node stack
}
/* Output traversal sequence */
While(! StackEmpty(C2))
{
  Pop(C2,p);
  Visit(p->data);//output
  }
  Return OK;
```

6. Conclusion

Compared to the non-recursive algorithm, binary tree's recursive traversal algorithm is more simple and clear. Through simple recursive call, we can write the binary tree traversal algorithm very quickly. However, recursive algorithm relies heavily on pointer node, which means the entire recursive algorithm will not continue to go on if the pointer is lost. Besides, a recursive algorithm is computationally intensive so it needs to call itself constantly to narrow the scope of the call, which leads to a low-efficient program. Thus, we think out the non-recursive calls. Although the non-recursive binary tree traversal is cumbersome procedure, it can better to see the whole process of traversal, such as how to use stack to storage node and then output it one by one. And the efficiency of program is high and the calculate amount is small. In this way, the non-recursive algorithm has been optimized and its complexity has been reduced. And the program operates faster than before. Then, whether to select non-recursive or recursive algorithm depends on their own program.

Acknowledgements

The work is supported by the project of Zhejiang province education department of China, Grant No.Y201326675.

References

[1] Makinson, D. (2009) Sets, Logic and Maths for Computing. Springer Science & Business Media, London, 199.

[2] Hazewinkel, M., Ed. (2001) "Binary Tree", Encyclopedia of Mathematics, Springer, London; Also in Print as Hazewinkel, M. (1997). Encyclopaedia of Mathematics. Supplement I. Springer Science & Business Media, London, 124. http://dx.doi.org/10.1007/978-94-015-1288-6

[3] Ye, H.Y. (2015) Data Structure (Java Version). 2nd Edition, Electronic Industry Publishing House, Beijing.

[4] Xu, F.S., Li, L.C. and Ma, X.R. (2006) Universal Binary Tree Traversal Non-Recursive Algorithm. *Fujian Computer*, No. 6, 41, 121.

[5] Zhang, X.Q. (2012) Improvement of Binary Non-Recursive Algorithm. *Journal of Jiamusi University* (*Natural Science Edition*), **31**, 926-928.