TEST Engineering & Management

January - February 2020
ISSN: 0193 - 4120 Page No. 1950 - 1956

Recursive and Non Recursive Algorithms to Traverse Non Linear Data Structures

Dr. B. Madhuravani[1], K. Sai Prasad[2], Prof. N. Chandra Sekhar Reddy[3]
[1,2,3]*Department of Computer Science and Engineering, MLR Institute of Technology, Hyderabad*

Article Info
Volume 82
Page Number: 1950 - 1956
Publication Issue:
January-February 2020

Article History
Article Received: *14 March 2019*
Revised: *27 May 2019*
Accepted: *16 October 2019*
Publication: *11 January 2020*

Abstract

This paper portrays a study on various traversal procedures to print the components of non linear data structures like Tree and Graph. As linear data structures can carry out the traversal procedure with the aid of any repetitive statement which is not permitted on non linear data structures. For this reason, we approached different techniques to perform it using either Stack or Queue Data structures. Several methods have been suggested to visit elements of Non linear data structures. This paper offers, various traversal techniques and implemented on C platform.

Keywords: *non linear data structures, algorithm, recursive, non-recursive.*

INTRODUCTION

Storage of data in a computer system plays a crucial role to perform operations efficiently on a group of data. The procedure to store the data in a system is called a Data Structures. The various data structures are grouped into two categories. a) Linear b) Non Linear. In Linear data structures the elements are stored sequentially where in non linear elements are not in sequential order. Numerous applications are there with non linear data structures. The non linear data structures include Tree and Graph which are used in several real time applications like transportation, networking, gaming, decision making, search engines etc..Further more, trees are used in Operating systems, designing compilers, processing text etc[1].. In view of this, it is very important to store and retrieve data in non linear data structures. The common techniques to traverse a Tree are inorder, preorder and postorder, whereas techniques for Graph are Breadth First Search and Depth First Search[1, 2, 3, 4, 5, 6].The traversal sequence for the Binary Tree depicted in Fig 1 and Graph in Fig 2 is

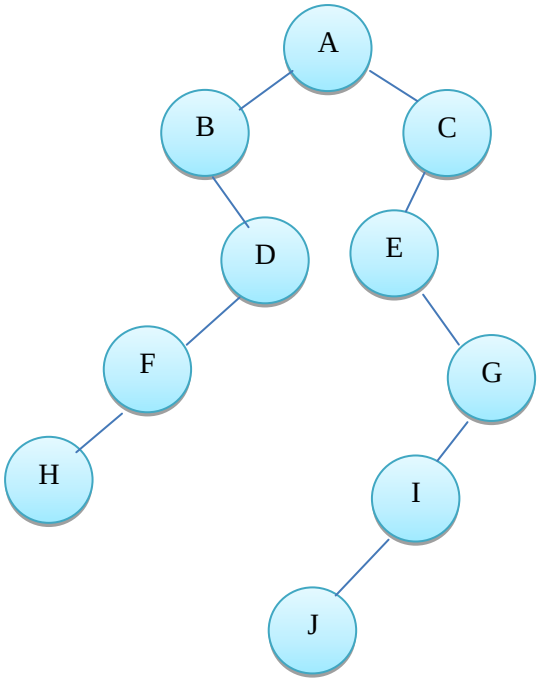


Fig 1: Binary Tree

January - February 2020
ISSN: 0193 - 4120 Page No. 1950 - 1956

The Fig 1 traversal sequence is..

Inorder : B=>H=>F=>D=>A=>E=>J=>I=>G=>C

Preorder:A=>B=>D=>F=>H=>C=>E=>G=>I=>J

Postorder:H=>F=>D=>B=>J=>I=>G=>E=>C=>A

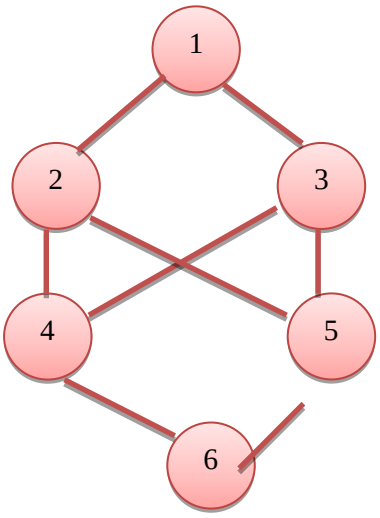


Fig 2. Graph

The Fig 2 traversal sequence is..

BFS : 1-2-3-4-5-6

DFS : 1-2-4-3-5-6

RECURSIVE ALGORITHMS FOR TREE

A node in a tree is defined as taking 3 parts- a) data – stores a value b) *left – stores the address of left child c)*right – stores the address of right child. The traversal can be done in 3 ways which are further taken as 6 ways.

i) inorder ii) preorder iii) postorderiv) converse_inorder v) converse_preorder vi) converse_postorder

INORDER TRAVERSAL:

The procedure is (Fig 3)-

Step 1: Left sub-tree traversal in Inorder

Step 2: Print Root node

Step 3: Right sub-tree traversal in Inorder

```
Algorithm inorder(root)
{if root is not NULL
        {callinorder for root left subtree
                print root
                callinorder for rootrightsubtree}
```

PREORDER TRAVERSAL:

The procedure is (Fig 3)-

Step 1: Print Root node

Step 2: Left sub-tree traversal in Inorder

Step 3: Right sub-tree traversal in Inorder

```
Algorithm preorder(root)
{
        if root is not NULL
        {
                print root
                callinorder for root left subtree
                callinorder for rootrightsubtree
        }
}
```

POSTORDER TRAVERSAL:

The procedure is (Fig 3)-

Step 1: Left sub-tree traversal in Inorder

Step 2: Right sub-tree traversal in Inorder

Step 3: Print Root node

```
Algorithm postorder(root)
{
        if root is not NULL
```

January - February 2020
ISSN: 0193 - 4120 Page No. 1950 - 1956

```
        {
                callinorder for root left subtree
                callinorder for rootrightsubtree
                print root
        }
}
```

CONVERSE INORDER TRAVERSAL:

The procedure is (Fig 3)-

Step 1: Right sub-tree traversal in Inorder

Step 2: Print Root node

Step 3: Left sub-tree traversal in Inorder

```
Algorithm inorder(root)
{
        if root is not NULL
        {
                callinorder for rootrightsubtree
                print root
                callinorder for root left subtree
        }
}
```

CONVERSE PREORDER TRAVERSAL:

The procedure is (Fig 3)-

Step 1: Print Root node

Step 2: Right sub-tree traversal in Inorder

Step 3: Left sub-tree traversal in Inorder

```
Algorithm preorder(root)
{
        if root is not NULL
        {
                print root
                callinorder for rootrightsubtree
                callinorder for root left subtree
        }
}
```

CONVERSE POSTORDER TRAVERSAL:

The procedure is (Fig 3)-

Step 1: Right sub-tree traversal in Inorder

Step 2: Left sub-tree traversal in Inorder

Step 3: Print Root node

```
Algorithm postorder(root)
{
        if root is not NULL
        {
                callinorder for root rightsubtree
                callinorder for root left subtree
                print root
        }
}
```

January - February 2020
ISSN: 0193 - 4120 Page No. 1950 - 1956

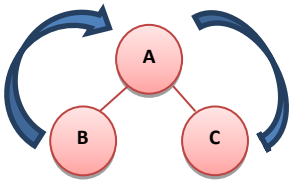
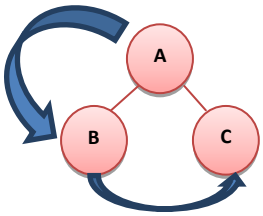
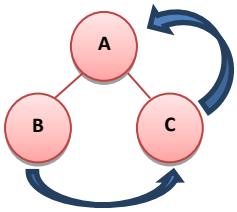
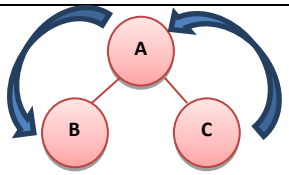
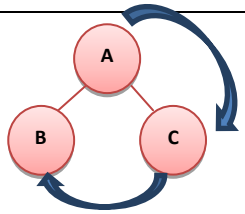
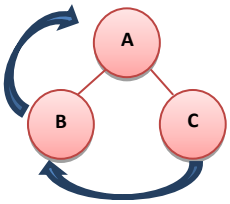
INORDER	PREORDER	POSTORDER
 B – A - C	 A- B - C	 B – C - A
CONVERSE_INORDER	**CONVERSE_PREORDER**	**CONVERSE_POSTORDER**
 C – A - B	 A – C - B	 C – B - A

Fig 3: Recursive Traversal Techniques

NON RECURSIVE ALGORITHMS FOR TREE

A special kind of a tree, a Binary Tree can be traversed using Inorder, Preorder and Postorder in non recursive manner. To traverse a tree in non recursion, we use STACK data structure, where the most recent element will be processed.

INORDER TRAVERSAL using NON RECURSION

The procedure is:

Step 1: Start with Root node

Step 2: Initialize current = Root

Step 3: Push current to stack and move to its left child.

Step 4: Repeat Step 3 until there is no left child.

Step 5: Initialize current with Stack top element

Step 6: Print current element

Step 7: Move to current right child

Step 8: Repeat steps 3-7 until Stack is empty.

```
Algorithminorder(struct node *cont)
{
STACK[0]=NULL;
while (cont != NULL || !isEmpty())
   {
while (cont !=  NULL)
      {
STACK[++top]=cont;
cont = cont->LEFT;
      }
```

January - February 2020
ISSN: 0193 - 4120 Page No. 1950 - 1956

```
cont=STACK[top--];
printcont;
cont = cont->RIGHT;
   }
}
```

POSTORDER TRAVERSAL using NON RECURSION

The procedure involves two stacks

Step 1: Push Root node to stack1

Step 2: Initialize current with stack1 popped element

Step 3: Push popped element to Stack2

Step 4: Until there is a left child to current push it onto stack

Step 5: Until there is a right child to current push it onto stack

Step 6: Repeat steps 2-5 until stack1 is empty.

```
Algorithmpostorder(struct node *cont)
{
s[0]=NULL;STACK1[++top]=cont;
while (!isEmpty1())
   {
cont = STACK1[top--];
STACK2[++t]=cont;

if(curr->LEFT != NULL)
STACK1[++top] = curr->LEFT;
if(curr->RIGHT!= NULL)
STACK1[++top] = curr->RIGHT;
   }
while(!isEmpty2())
   {
cont=STACK2[t--];
printcont;
   }
}
```

PREORDER TRAVERSAL using NON RECURSION

The procedure is

Step 1: Initialize current to Root

Step 2: Print current

Step 3: Until there is a right child to current push it onto stack

Step 4: Until there is a left child to current push it onto stack

Step 5: Store popped element in current

Step 6: Repeat steps 2-5 until current is NULL

```
Algorithmpreorder(struct node *cont)
{
STACK[0]=NULL;
while (cont != NULL)
   {
printcont;
if(cont->RIGHT!= NULL)
STACK[++top] = curr->RIGHT;
if(curr->LEFT != NULL)
STACK[++top] = curr->LEFT;
```

January - February 2020
ISSN: 0193 - 4120 Page No. 1950 - 1956

```
curr = STACK[top--];
    }
}
```

The analysis about Recursive and Non recursive Tree traversal technique is given in Table 1:

S.No	Recursive Traversal	Non Recursive Traversal
1	Easy to implement	Complex to implement
2	Takes less SLOC	Results in more SLOC
3	Takes less space	Takes more space
4	Time complexity: O(n)	Time complexity: O(nlogn)

Table 1: Recursive vs Non Recursive Traversals

GRAPH TRAVERSAL TECHNIQUES

A non linear data structure, Graph can be traversed in two ways: a) BFS- Breadth First Search and b) DFS – Depth First search

BFS- BREADTH FIRST SEARCH

The BFS implementation make use of QUEUE data structure. The algorithm is given below:

```
AlgorithmBFS(int v)
{
        insert_queue(v);
        while(!isEmpty_queue())
        {
                v = delete_queue( );
                print v;
                VISIT[v] = 1;
                for(x=1; x<=n; x++)
                {
                        if(GRAPH[v][x] == 1
                        &&VISIT[x] == 0)
                        {
                                insert_queue(x);
                        VISIT[x]=1;
                                }
                        }
                }
        }
```

DFS- DEPTH FIRST SEARCH

The DFS implementation make use of STACK data structure. The algorithm is given below:

```
AlgorithmDFS(int v)
{

intx,flag=0;
        push(v);
        VISIT[v] = 1;
        printf("%d  ",v);
        while(!isEmpty_stack())
        {
        flag=0;
                y=STACK[top];
                for(x=1; x<=n; x++)
                {
                        if(a[y][x] == 1
                        &&VISIT[x] == 0)
                        {
                                push(x);
                                printx;
                        VISIT[x]=1;
                        flag=1;
                        break;
                        }
                }
                if(flag==0) top--;
        }

}
```

January - February 2020
ISSN: 0193 - 4120 Page No. 1950 - 1956

5. CONCLUSION

This paper, explains various techniques and implementations to visit and print the elements of non linear data structures. The implementations for non recursive algorithms make use of either STACK or QUEUE data structure. This paper presented the procedure in a simpler fashion with less SLOC. In future, we implement all the techniques without taking either STACK or QUEUE.

References:

[1] Vinu V Das, "A new Non-Recursive Algorithm for Reconstructing a Binary Tree from its Traversals", IEEE Comm., pp. 261-263, 2010

[2] Vinu V Das, "Principles of Data Structures Using C and C++", New Age International Publishers, Reading, Mass., 2005.

[3] M. Weiss, Data Structures & Problem Solving Using Java, 2"d ed., Addison Wesley, 2002

[4] D. E. Knuth, The Art of Computer Programming, Vol. 3 (2nd ed.): Sorting and Searching, Addison Wesley, 1998.

[5] J. Driscoll, and Y. Lien, A Selective Traversal Algorithm for Binary Search Trees, Communications of the ACM, Number 6, Vol. 21, 1978, pp. 445-447.

[6] R. Sedgewick, Algorithms in Java, 3d edition, Addison Wesley, 2003

[7] J. Driscoll, and Y. Lien, A Selective TraversalAlgorithm for Binary Search Trees, Communicationsof the ACM, Number 6, Vol. 21, 1978, pp. 445-447.

[8] Data Structures Using C, ReemaThareja, Second Edition 2011.

[9] N. Chandra Sekhar Reddy, Dr. Purna Chandra Rao, Dr G Govardhan, An Intrusion Detection System for Secure Distributed Local Action Detection and Retransmission of Packets, International Journal of Soft Computing, 12(1), 45-49, 2017.

[10] Shirisha N, Sowmya G, DivyaJyothi G, Navya K, Design and implementation of an android application for management of events, International Journal of Engineering and Technology(UAE) 2018.