TREE TRAVERSALS USING PROLOG

ARTIFICIAL INTELLIGENCE

Submitted By -

Jatin Rohilla

14HCS4109

ACKNOWLEDGEMENT

"The compilation of any project depends upon the co-operation, coordination and combined efforts of several resources of knowledge, inspiration and energy".

Words fall short acknowledging immense support lent to us by everyone and we give full credit to them.

Our sincere thanks goes to **Mr. Ankit Rajpal** for giving us the opportunity to discover more knowledge. We are thankful to him for his constant support, guidance and cooperation throughout to accomplish this project. It has been of the greatest help in bringing out our task in the present state.

Jatin Rohilla

CERTIFICATE

This is to certify that the project entitled, *"Tree Traversals Using Prolog"* has been done by **Jatin Rohilla** of Bachelor of Computer Science (Hons.) during semester VI in the academic year 2016-2017 from Deen Dayal Upadhyaya College, University of Delhi under the supervision of **Mr. Ankit Rajpal**.

Mr. Ankit Rajpal

INTRODUCTION

Trees are one of the most important data structures and help us in a wide range of problems by providing us an efficient way of storing, searching and manipulating data. For effective usage, we must be able to traverse the tree.

Also, solely by the traversal type we can solve a number of common problems, including the Infix and Postfix notations of an expressions.

PROBLEM STATEMENT –

Given a tree, do the In-order, pre-order and post-order traversal of the tree using GNU prolog.

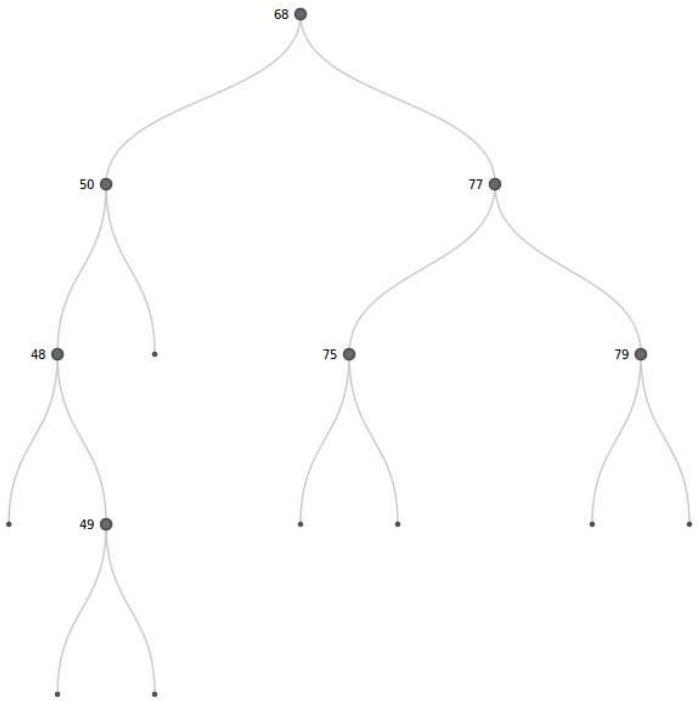
The order of nodes in each traversal differs slightly.

Pre-order - Root Node, Left Subtree, Right Subtree

In-order - Left Subtree, Root Node, Right Subtree

Post-order - Left Subtree, Right Subtree, Root Node

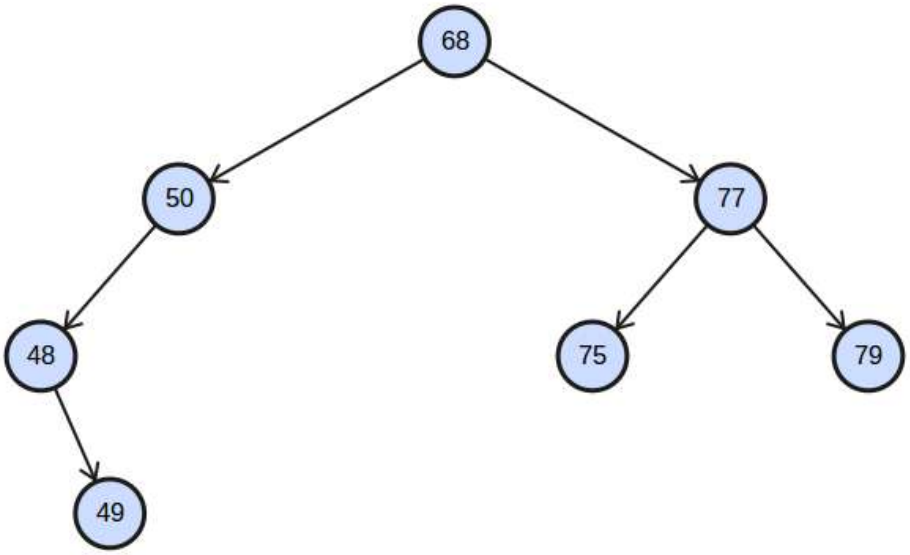
INITIAL STATE



The initial state is the tree notation of prolog. For the above tree, the tree notation in prolog will be –

```
tr(
     tr(
          tr(
               nil,
               48,
               tr(nil,  49,nil)
             ),
          50,
          Nil
       ),
     68,
     tr(
          tr(nil,75,nil),
          77,
          tr(nil,79,nil)
       )
  )
```

GOAL STATE



The Goal State for the problem is lists containing respective orders of the tree.
Expected Traversals for the given tree is -

The In-order traversal is : `[48,49,50,68,75,77,79]`

The Pre-order traversal is : `[68,50,48,49,77,75,79]`

The Post-order traversal is : `[49,48,50,75,79,77,68]`

PROLOG CODE

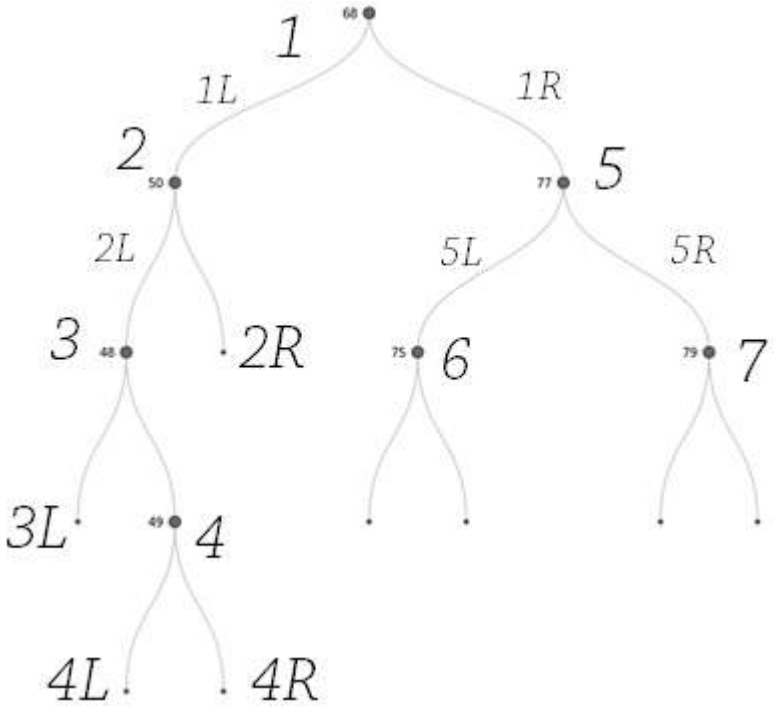
```
go:-  nl,
      write('*** TREE TRAVERSALS - IN-Order,
      PRE-Order, POST-Order ***'), nl, nl,
      write('Enter the Tree : '), read(T),
      nl, nl, write('For the Tree   : '), write(T),nl,nl,
      inorder(T,R1),
      write('The Inorder traversal is    : '),
      write(R1),nl,
      preorder(T,R2),
      write('The Pre-order traversal is  : '),
      write(R2),nl,
      postorder(T,R3),
      write('The Post-order traversal is : '),
      write(R3),nl.

inorder(nil,[]):-!.
inorder(tr(nil,X,nil),[X]):-!.
inorder(tr(LEFT,R,RIGHT),Res):- inorder(LEFT,LT),
                                inorder(RIGHT,RT),
                                append(LT,[R],Temp),
                                append(Temp,RT,Res),!.
preorder(nil,[]):-!.
preorder(tr(nil,X,nil),[X]):-!.
preorder(tr(LEFT,R,RIGHT),Res):- preorder(LEFT,LT),
                                 preorder(RIGHT,RT),
                                 append([R],LT,Temp),
                                 append(Temp,RT,Res),!.
postorder(nil,[]):-!.
postorder(tr(nil,X,nil),[X]):-!.
postorder(tr(LEFT,R,RIGHT),Res):- postorder(LEFT,LT),
                                  postorder(RIGHT,RT),
                                  append(LT,RT,Temp),
                                  append(Temp,[R],Res),!.
```

OUTPUT

```
| ?- consult('tree.pl').
compiling /home/jatin/Documents/AI/project/tree.pl
for byte code...
/home/jatin/Documents/AI/project/tree.pl compiled,
 26 lines read - 4936 bytes written, 4 ms

yes

| ?- go.

*** TREE TRAVERSALS - IN-Order, PRE-Order, POST-Order ***

Enter the Tree : tr(tr(tr(nil,48,tr(nil,49,nil)),50,nil),68,
tr(tr(nil,75,nil),77,tr(nil,79,nil))).


For the Tree   : tr(tr(tr(nil,48,tr(nil,49,nil)),50,nil),68,
tr(tr(nil,75,nil),77,tr(nil,79,nil)))

The Inorder traversal is    : [48,49,50,68,75,77,79]
The Pre-order traversal is  : [68,50,48,49,77,75,79]
The Post-order traversal is : [49,48,50,75,79,77,68]

yes
| ?-
```

DRY RUN

In-Order Traversal



/* Want to process Node 1, move to left Subtree */

inorder(tr(tr(tr(nil,48,tr(nil,49,nil)),50,nil),68, tr(tr(nil,75,nil),77,tr(nil,79,nil))) ,_378)

/* NOW processing Left Subtree of Node 1 **********/

/* Want to process Node 2, move to left Subtree */

inorder(tr(tr(nil,48,tr(nil,49,nil)),50,nil),_402)

/* NOW processing Left Subtree of Node 2 */

/* Want to process Node 3, move to left Subtree */

inorder(tr(nil,48,tr(nil,49,nil)), _426)

/* Moving to Left Subtree of Node 3 */ /* Processing Node 3L */

inorder(nil,_450) , inorder(nil,[])

/* Moving to Right Subtree of Node 3 */ /* Processing Node 4 */
inorder(tr(nil,49,nil),_475) , inorder(tr(nil,49,nil),[49])

/* Merging Branches of Node 3 and Node 3L */
append([],[48],_505) , append([],[48],[48])
/* Merging Branches of Node(3,3L) and Node 4 */
append([48],[49],_531) append([48],[49],[48,49])
/* Resultant Merged Node((3,3L),4) reported back as 2L */
inorder(tr(nil,48,tr(nil,49,nil)),[48,49])

/* Processing Node 2R */
inorder(nil,_559) , inorder(nil,[])
/* Merging Branches of Node 2L and Node 2 */
append([48,49],[50],_587) , append([48,49],[50],[48,49,50])
/* Merging Branches of Node(2L,2) and Node 2R */
append([48,49,50],[],_617) , append([48,49,50],[],[48,49,50])

/* Resultant Merged Node((2,2L),2R) reported back as 1L */
inorder(tr(tr(nil,48,tr(nil,49,nil)),50,nil),[48,49,50])
/* Processing of Left-Subtree of Node 1 is complete. Result is 1L = [48,49,50] **********/

/* NOW processing Right Subtree of Node 1 **********/
/* Want to process Node 5, move to left Subtree */
inorder(tr(tr(nil,75,nil),77,tr(nil,79,nil)),_649) /* */

/* NOW processing Left Subtree of Node 5 */
/* Processing Node 6 */
inorder(tr(nil,75,nil),_673) , inorder(tr(nil,75,nil),[75])

/* NOW processing Left Subtree of Node 5 */
/* Processing Node 7 */
inorder(tr(nil,79,nil),_700) , inorder(tr(nil,79,nil),[79])

/* Merging Branches of Node 6 and Node 5 */

append([75],[77],_730) , append([75],[77],[75,77])

/* Merging Branches of Node(6,5) and Node 7 */

append([75,77],[79],_758)

append([75,77],[79],[75,77,79])

/* Resultant Merged Node((6,5),7) reported back as 1R */

inorder(tr(tr(nil,75,nil),77,tr(nil,79,nil)),[75,77,79])

/* Processing of Right Subtree of Node 1 complete 1R =[75,77,79] **********/

/* Merging Branches of Node 1L and Node 1 */

append([48,49,50],[68],_791)

append([48,49,50],[68],[48,49,50,68])

/* Merging Branches of Node(1L,1) and Node 1R */

append([48,49,50,68],[75,77,79],_823)

append([48,49,50,68],[75,77,79],[48,49,50,68,75,77,79])

/* Resultant Merged Node((1L,1),1R)reported back as

the Final Result = [48,49,50,68,75,77,79] **********/

inorder(tr(tr(tr(nil,48,tr(nil,49,nil)),50,nil),68,

tr(tr(nil,75,nil),77,tr(nil,79,nil))),[48,49,50,68,75,77,79])

The Final Result of In-Order Traversal is - [48,49,50,68,75,77,79]

Similarly, other two traversals can also be DRY-RUN.

BIBLIOGRAPHY –

1. William F. Clocksin, Christopher S. Mellish, Programming in Prolog.
2. http://stackoverflow.com/