Assignment 6

Name: Bhavana Bafna
Class: SE 9
Batch: E 9
Roll no.: 23107

1) Title : Threaded Binary Tree

2) Aim : To implement threaded binary tree

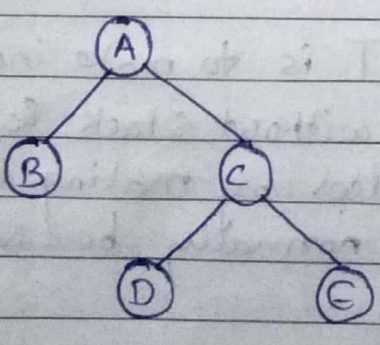
3) Problem Statement : Implement Inorder TBT. Traverse the implemented tree in Preorder & inorder traversal.

4) Theory:

i) Limitations of problem with normal binary tree :

Too many null pointer representation of binary tree.

n = no. of nodes
no. of non - null links = n - 1
Total links = 2n
Null links = 2n - (n-1) = n+1



0	1	2	3	4	5	6
A	B	C	—	—	D	E

Scanned with CamScanner

- Traversing is the most frequently used operation on tree.
- In normal binary tree a temp data structure(stack) is required to implement non-recursive traversal.
- We overcome these problems in TBT by replacing the null pointers with useful pointers called "threads".

2) TBT concept, definition with example:

Concept: As we discussed the problems faced in normal binary tree above, we replace the null pointers with threads.

Thread: It's a pointer to other nodes in tree for replacing the Null link.

By doing this we reutilize Null pointers.
This will result in:

1) No wastage of memory for null pointer.
2) Non recursive traversal without stack.

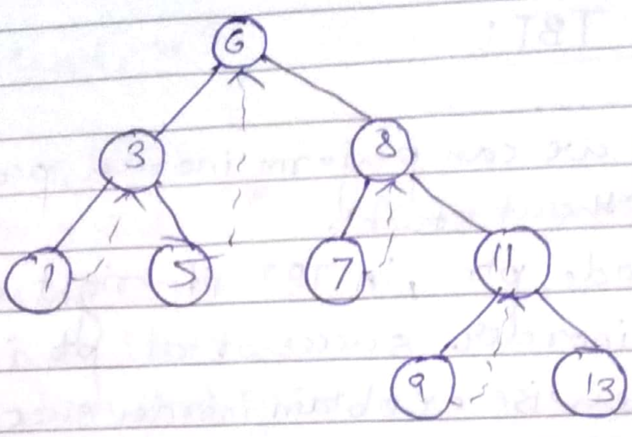
Definition:

The idea of TBT is to make inorder traversal footer & do it without stack & recursion. A BT is made threaded by making all right child pointer that would normally be NULL point to inord successor of node.

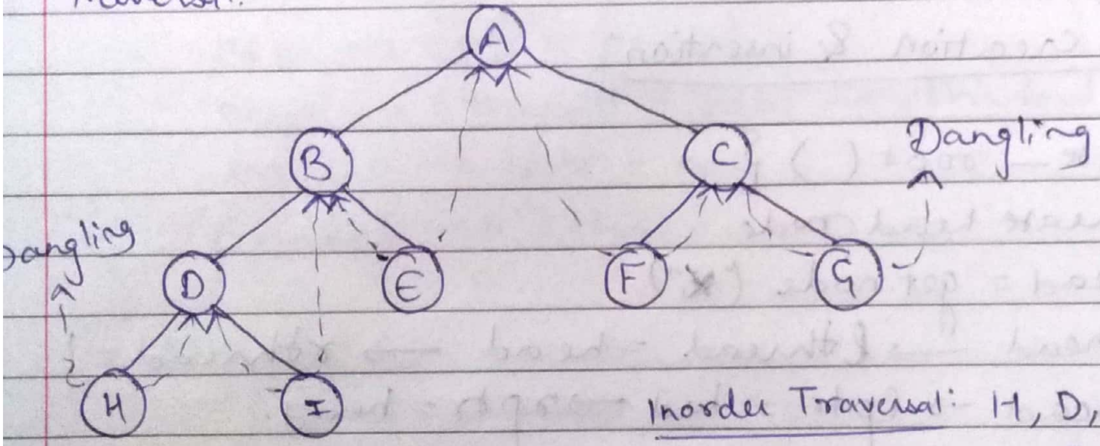
Types:

1) Single Threaded: Where a NULL right pointer is made to point to inorder successor (if exists).

Scanned with CamScanner



2) Double Threaded: Where both left & right NULL pointer are made to point to inorder predecessor & inorder successor respectively. The predecessor threads are useful for reverse inorder traversal & postorder traversal.



Inorder Traversal: H, D, I, B, E, A, F, C, G.

Rules for Construction:

1. If ptr → left_child is null
replace it with a pointer to the node that would be visited before ptr in an inorder traversal.
(i.e. inorder precedecessor).

2. If ptr → right-child in null
replace it with a pointer to node that would be visited after ptr in an inorder traversal.
(i.e., inorder successor)

Scanned with CamScanner

4) Advantages of TBT:

1) By threading we can peform inorder, preorder, postorder without stack.

2) For any node, ptr., in TBT if right child is present the inorder successor of ptr is ptr's parent. Otherwise we obtain inorder successor of ptr by following a path of left child links from right child of ptr until we reach a node with left subtree.

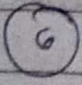
⊛ Algorithm:

1) Tree creation & insertion:

```
Create_root() {
// create head node
  head = getnode("*")
   head → lthread = head → rthread = 1
   head → lptr = hed → rptr = head.

// create root
  root = getnode()
  root → lthread = rthread = 1
  root → rptr = lptr = head
  head → lptr = root
  head → lthrd = 0
}
```

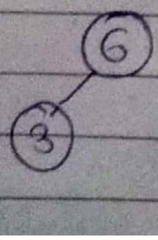
```
Insert (Head, x)

1) if Head = = Null
      Then print ("Create Rootfirst")
      (create_root (head))
      return head.
2) Parent = head → lptr.
3) Repeat through step 4 till insertion takes place.
4) Write ('Root is', parent → data)
   // take choice from user to insert.
   if choice is 1          / insert as left child.
   {     new = get node (x)
         new → lptr = parent → lptr
         new → rptr = parent
         new → lthread = new → rthread = 1
         parent → lptr = new
         parent → lthread = 0
   }
   else
         parent = parent → rptre

5) Return head.
```

1) Take input from user



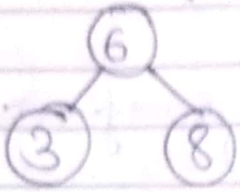
// Root created.

2) If user gives input as left child.

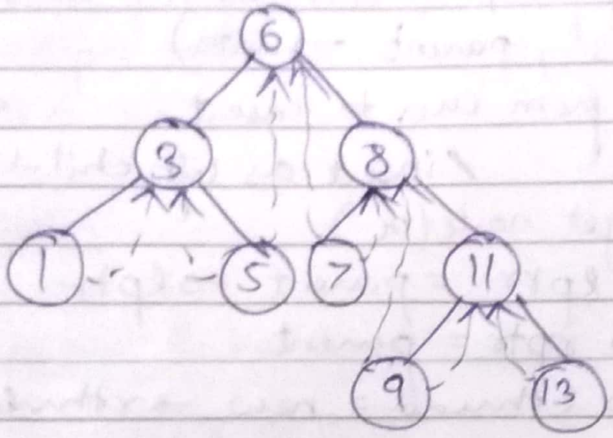


Scanned with CamScanner

3) If user gives input for right child of 6:

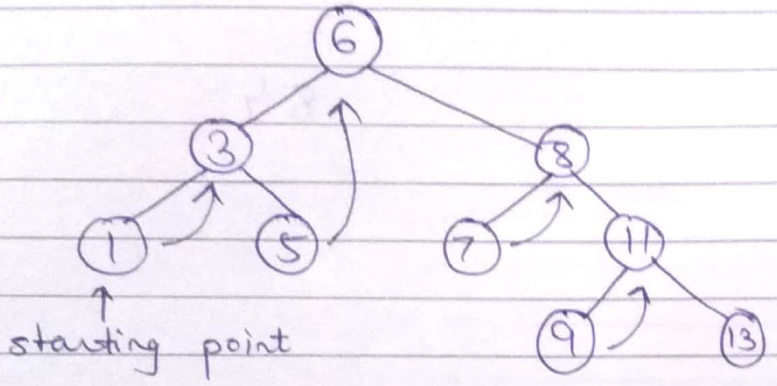


Similarly, we can insert nodes in tree:



2) <u>Inorder Traversal:</u>

```
Procedure inorder(head)
{
  current=head → lptr
  if (current → lptr = head)
     Then write ("Empty tree")
  Return
  Repeat while (current → lthread ≠ 0)
       //go to leftmost child of left subtree
       current = current → lptr
  repeat while (current != head)
  { display (current → data)
       if current → rthread=1
          current = current → rptr
   else {
    current = current → rptr
    repeat while (current → lthread = 0)
    current = current → lptr }
  } // end of inorder.
```



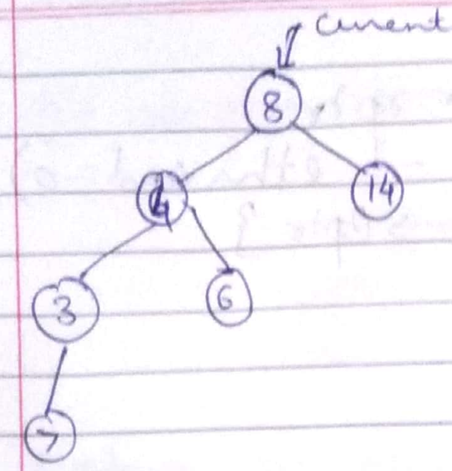
Output = 13 5 6 7 8 9 11 13

3) Preorder Traversal:

```
Preorder (head)
{ current = head → lptr
  if (current = head)
       print ("Empty tree")
       return
  repeat while current != head.
 { display (current → data)
   if (current → lthread = 0)     // if lchild present traverse
        current = current → lptr
  else
     { // if no left subtree, go to right subtree
       repeat while (current → rthread = 1)
          current = current → rptr
          current = current → rpt   // if left tree present
     }                                 process it first
} // end
```

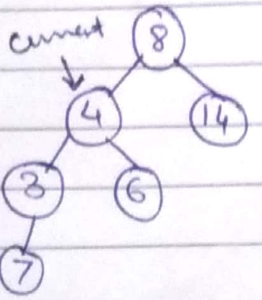
Output

1.



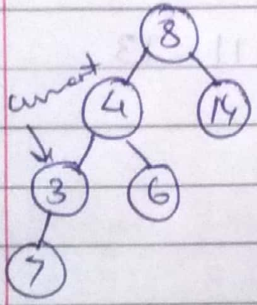
8

2.



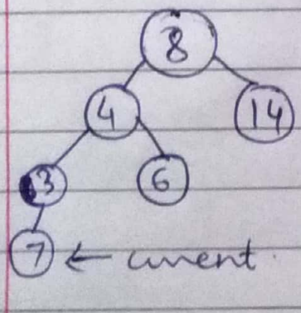
8 4

3.



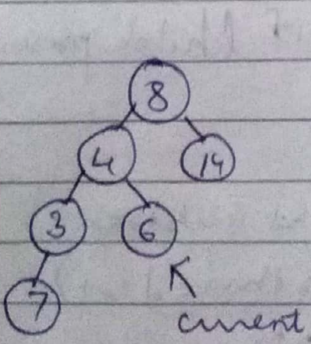
8 4 3

4.



8 4 3 7

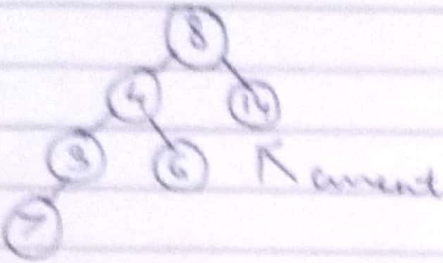
5.



8 4 3 7 6

Scanned with CamScanner

6.



output

8 4 3 7 6 14

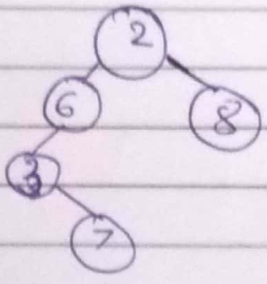
8) Test cases and Validations:

1) Valid input data with respect to tree you are constructing

2) Test case:

Input:

2	
6	(2 L)
8	(2R)
3	(6 L)
7	(3R)



Inorder = 3 7 6 2 8
Preorder = 2 6 3 7 8

9) Conclusion:

Understood working & creation of Threaded Binary tree.

Analysis:

Time complexity = O(n)
Space complexity = O(1)

Scanned with CamScanner