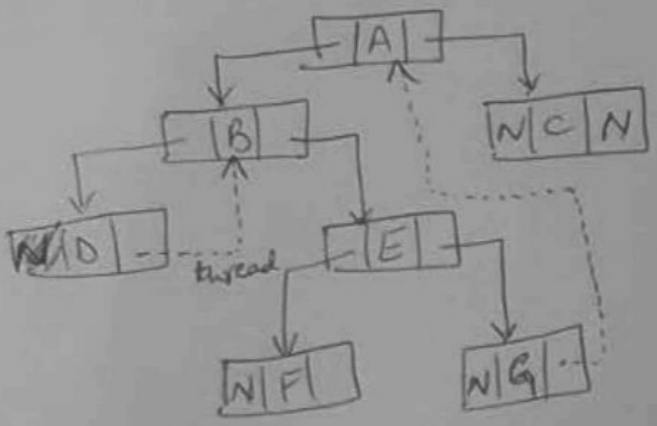
Threaded BST Use of null pointers of terminal nodes that contain the addresses of their successors and predecessors.

Right-in threaded BST (pointers to successor in inorder traversal)

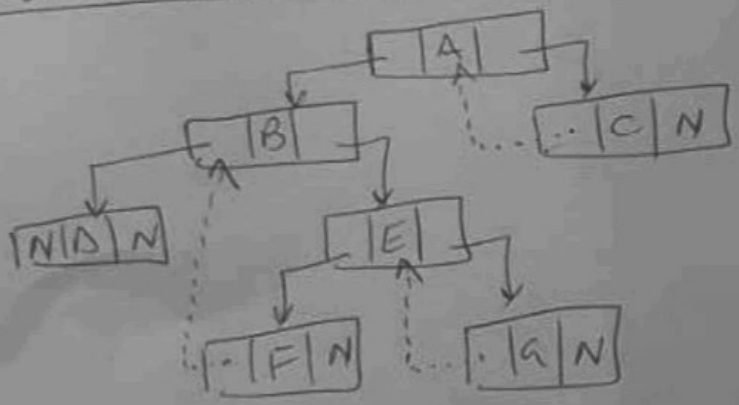


Inorder traversal

D B F E G A C

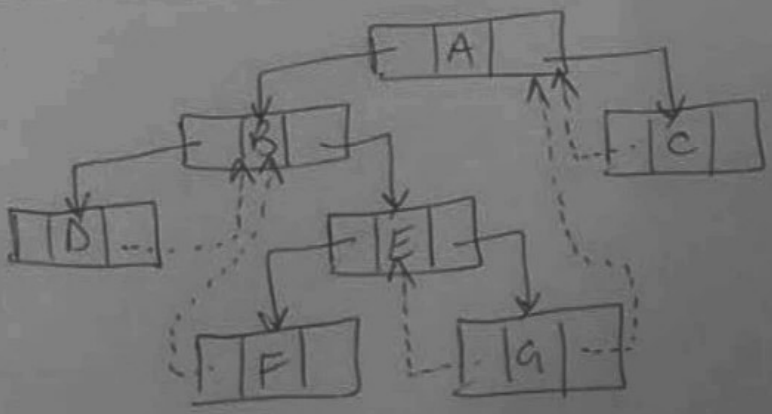
Successor of D = B
" " N = A

Left-in threaded BST



Predecessor of F = B
" " G = E
" " C = A

Fully in threaded BST



Successors and predecessors both.

Similarly we can have TBST for preorder and postorder traversal. Left pointers will point to the predecessors and right pointers will point to the successors.

Two boolean variables

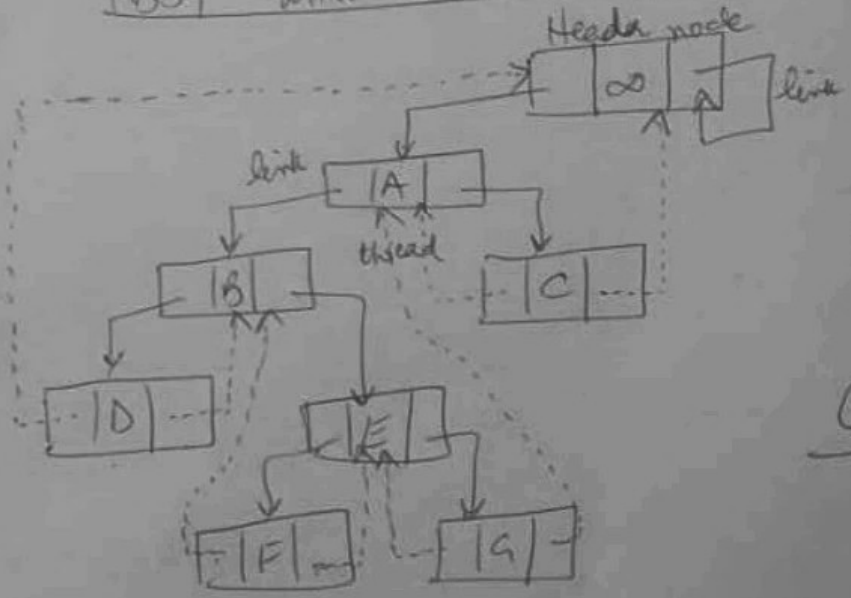
left = link (pointer leftptr points to left child of a node)

left = thread (pointer leftptr points to inorder predecessor of a node)

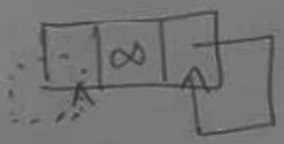
right = link
right = thread

Left pointer of first node and right pointer of last node are always null. Solution is:

IBST with a header node



Empty tree

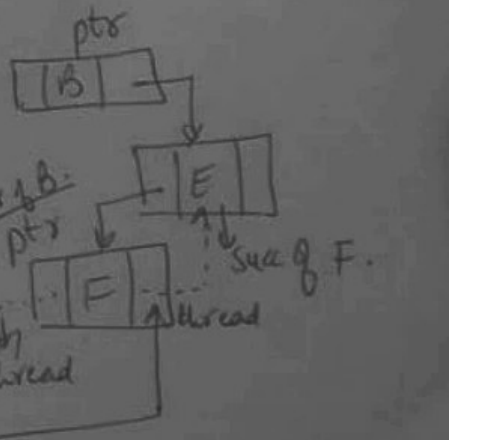


Condition.

head.lchild = head

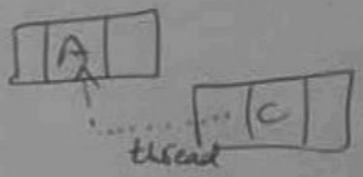
Finding inorder successor of a node in a TBST

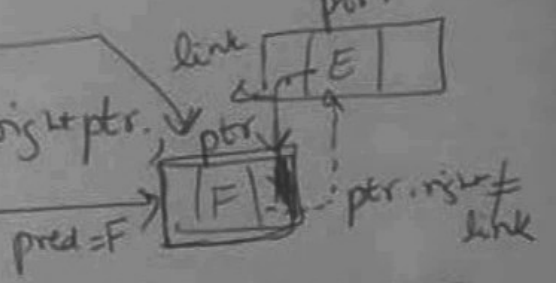
```
node insucc (node ptr)
{
    node succ;
    if (ptr.right = thread)
        succ = ptr.rightptr;
    else
    {
        ptr = ptr.rightptr;
        while (ptr.left = link) ptr = ptr.leftptr;
        succ = ptr;
    }
    return succ;
}
```



Finding inorder predecessor of a node in in-thread BST

```
node inpred(node ptr)
{
    node pred;
    if(ptr.left == thread)
        pred = ptr.leftptr;
    else
    {
        ptr = ptr.leftptr;
        while(ptr.right == link) ptr = ptr.rightptr;
        pred = ptr;
    }
    return pred;
}
```





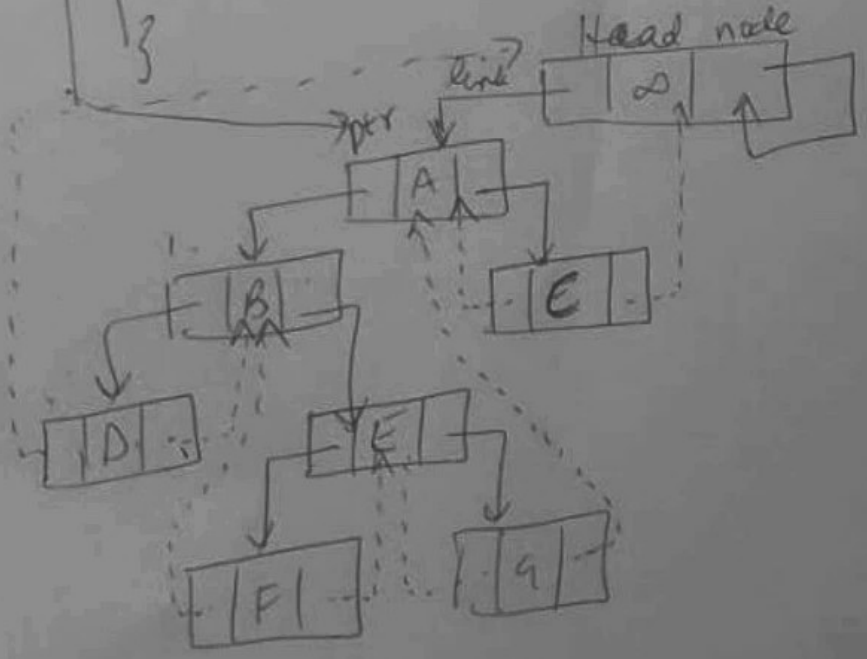
F is predecessor of E.

Inorder traversal of in-threaded BST

```
inorder()
{
    node ptr;
    ptr = head.leftptr;        Tree start node (root)
    while(ptr.left == link) ptr = ptr.leftptr;
    output ptr.info (leftmost node)
    while(1)
    {
        ptr = insuc(ptr);
        if (ptr == head) (last node reached b/c rightptr of
                          last node points to head)
            break;
        output ptr.info;
    }
}
```

Preorder traversal of threaded BST

```
preorder()
{
    node ptr;
    ptr = head->leftptr;  -> first node of the tree (root)
    while( ptr != head)   (last node)
    {
        output ptr.info;
        if (ptr.left == link) ptr = ptr.leftptr;
        else if (ptr.right == link) ptr = ptr.rightptr;
        else
        {
            while (ptr != head && ptr.right == thread)
                ptr = ptr.rightptr;
            if (ptr != head) ptr = ptr.rightptr;
        }
    }
}
```



Preorder traversal

A B D E F G C

Similarly postorder traversal can be done

Insertion and Deletion in TBST

Same as in BST but we will have to adjust the threads after these operations.

Insertion

Case 1 When the tree is empty.

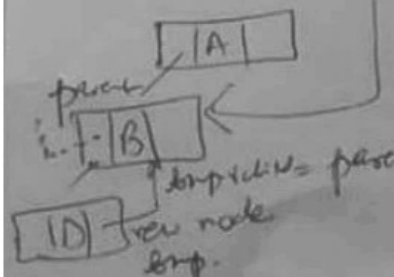
```
head.left = link;
head.leftptr = temp;
temp.leftptr = head;
temp.rightptr = head;
```

New node created

```
temp.leftptr = thread;
temp.rightptr = thread;
```

Case 2 When new node is inserted as the left child of its parent.

```
parent.left = link;
parent.leftptr = temp;
temp.leftptr = parent.leftptr (parent predecessor is now its predecessor)
temp.rightptr = parent; (thread)
```



Case 3 When new node is inserted as the right child of its parent.

```
parent.right = link;
parent.rightptr = temp;
temp.rightptr = parent;
temp.leftptr = parent.leftptr;
```

Deletion

Case 1 Node to be deleted is leaf node.

If left leaf node →

```
parent.left = thread;
parent.leftptr = loc.leftptr;
```

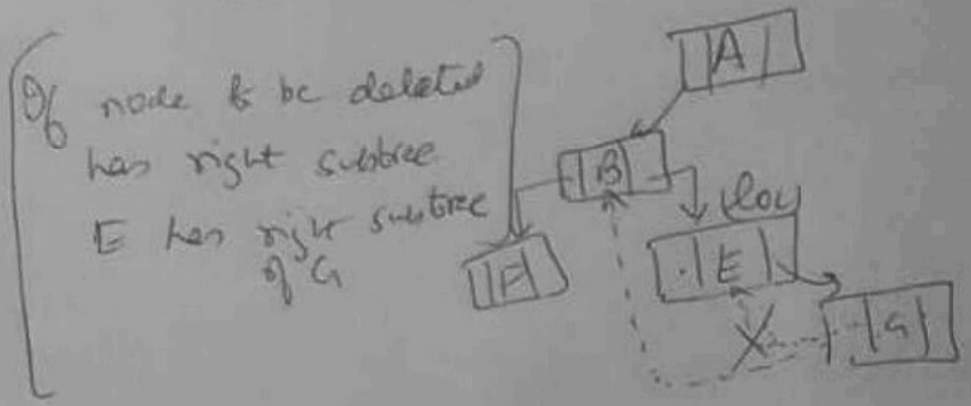
If right leaf node →

```
parent.right = thread;
parent.rightptr = loc.rightptr;
```

Case 2: Node to be deleted has one child.

Delete the node as in BST.
After deletion, find the inorder successor and predecessor of that node.

```
s = insuc(loc);
p = inpred(loc);
if(loc->right = link) s->leftptr = p;
```



Inorder traversal
F B E G A
pred succ
P = B
S = G
s->leftptr = P

```
if (loc->left = link) p->rightptr = s;
```

Case 3: Node to be deleted has 2 children. replaced.
Same as in BST such that it will be deleted with its inorder successor which will be found with insucc() function. That inorder successor will first be deleted with case a() or case(b) [if it is a terminal node (case a) OR a node with one child. (case b)]