PES UNIVERSITY

Department of Computer Science and Engineering

PES UNIVERSITY

UE19CS202: Data Structures and its Applications (4-0-0-4-4)

Trees
Threaded Binary Search Tree

Dr. Shylaja S S
Ms. Kusuma K V

Threaded binary search tree and its implementation

```
//Iterative Inorder Traversal Revisited
iterativeInorder(root)
 s = emptyStack
p = root
do {
                while(p != null)
                {       /* Travel down left branches as far as possible saving pointers to
nodes passed in the stack*/
                        push(s, p)
                        p = p->left
                } //At this point, the left subtree is empty
                p = pop(s)
                print p ->info          //visit the node
                p = p ->right           //traverse right subtree
} while(!isEmpty(s) or p != null)
```

If we examine the iterativeInorder function to find out the reason for the need of stack, we see that the stack is popped when *p* equals NULL. This happens in two cases:
case (i): The **while** loop is exited after having been executed one or more times. This implies that the program has travelled down left branches until it reached a NULL pointer, stacking a pointer to each node as it was passed. Thus, the top element of the stack is the value of *p* before it became NULL. If an auxiliary pointer *q* is kept one step behind *p*, the value of *q* can be used directly and need not be popped.

case (ii): The **while** loop is skipped entirely. This occurs after reaching a node with an empty right subtree, executing the statement *p = p -> right*, and returning to repeat the body of the **do while** loop. At this point, top points to the node whose left subtree was just traversed. If we had not stacked the pointer to each node as it was passed then we would have lost our way in this case. So, a node with an empty right subtree, instead of containing a NULL pointer in its right field, suppose it contained a pointer to the node that would be on top of the stack at that point in the algorithm (that is, a pointer to its inorder successor), then there would no longer be a need for the stack, since the last node visited during traversal of a left subtree points directly to its inorder successor. Such a pointer is called a **thread** and must be differentiable from a tree pointer that is used to link a node to its left or right subtree.

A binary tree in which the right pointer of a node, points to the inorder successor if in case it is not pointing to the child is called **Right In-Threaded Binary Tree**. Figure 1 shows a binary search tree and Figure 2 shows the corresponding right in-threaded binary search tree. The threads are drawn in dotted lines to differentiate them from tree pointers. Note that the rightmost node in the tree still has a NULL right pointer, since it has no inorder successor.

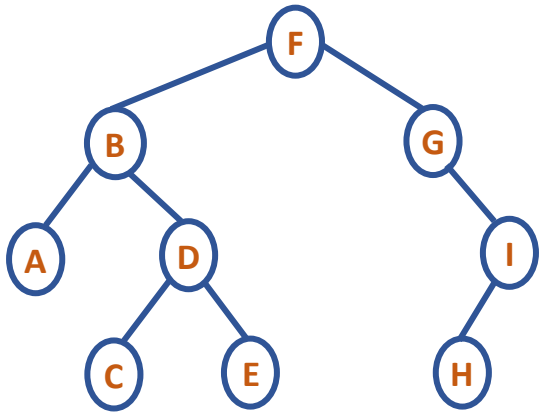

Figure 1: Binary Search Tree

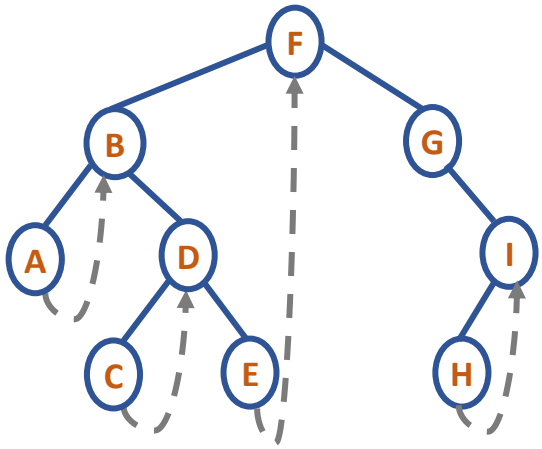

Figure 2: Right In-Threaded Binary Search Tree

Nodes with Right Pointer NULL	A	C	E	H	I
Inorder Successor	B	D	F	I	-

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

A binary tree in which the left pointer of a node, points to the inorder predecessor if in case it is not pointing to the child is called **Left In-Threaded Binary Tree**. A binary tree in which the right and left pointer of a node, points to the inorder successor and inorder predecessor respectively if in case it is not pointing to the right and left child respectively is called **In-Threaded Binary Tree**. Figure 3 and Figure 4 shows the left in-threaded and in-threaded binary search tree respectively, corresponding to the binary search tree in Figure 1.

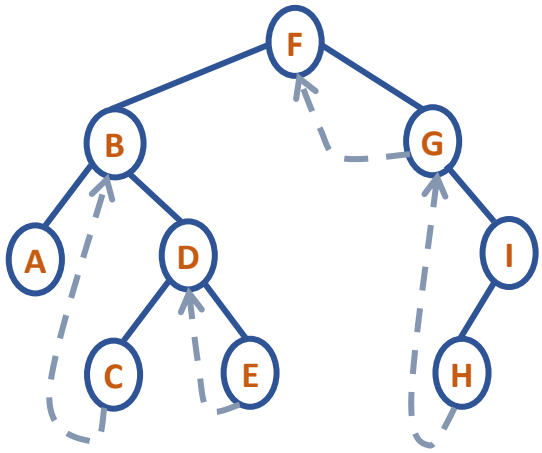

Figure 3: Left In-Threaded Binary Search Tree

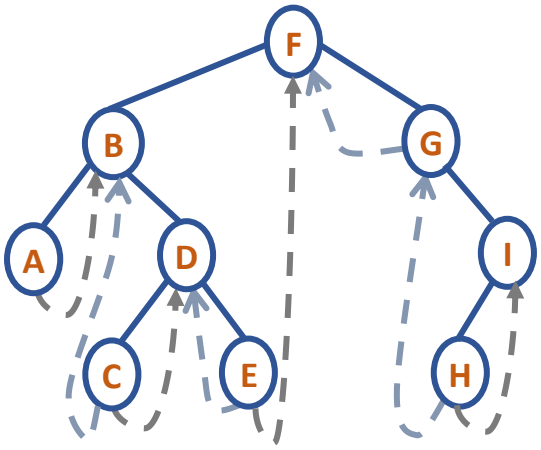

Figure 4: In-Threaded Binary Search Tree

Nodes with Left Pointer NULL	A	C	E	G	H
Inorder Predecessor	-	B	D	F	G

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

To implement a right in-threaded binary search tree under the dynamic node implementation of a binary tree, an extra logical field, rthread is included within each node to indicate whether or not its right pointer is a thread. Note that, for consistency, the rthread field of the rightmost node of a tree (that is, the last node in the tree's inorder traversal) is also set to TRUE, although its right field remains NULL. Such a node is declared as follows:

```
typedef struct node
{
        int info;
        bool rthread;           //TRUE if right is NULL or a non-NULL thread
        struct node *left;      //Pointer to left child
        struct node *right;     //Pointer to right child
}NODE;
```

//C program to demonstrate construction and inorder traversal of right in-threaded binary search tree

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>

typedef struct node
{
        int info;
        bool rthread;
        struct node *left;
        struct node *right;
}NODE;

typedef struct tree
{
        NODE *root;
}TREE;

void init(TREE *pt)
{
        pt->root=NULL;
}
```

```
NODE* createNode(int e)
{
        NODE* temp=malloc(sizeof(NODE));
        temp->info=e;
        temp->left=NULL;
        temp->right=NULL;
        temp->rthread=1;
        return temp;
}

void setLeft(NODE* q,int e)             //set node to left of queue
{
        NODE* temp=createNode(e);
        q->left=temp;
        temp->right=q;
}

void setRight(NODE* q,int e)
{
        NODE* temp=createNode(e);
        temp->right=q->right;                   //thread
        q->right=temp;
        q->rthread=0;
}

void inOrder(TREE *t)
{
        NODE *p=t->root;
        NODE *q;

        do{
                q=NULL;
                while(p!=NULL)
                {
                        q=p;
                        p=p->left;
                }
                if(q!=NULL)
                {
                        printf("%d ",q->info);
                        p=q->right;
```

```
                        while(q->rthread && p!=NULL)
                        {
                                printf("%d ",p->info);
                                q=p;
                                p=p->right;
                        }
                }
        }while(q!=NULL);
}

void create(TREE *pt)
{
        NODE *p,*q;
        int e,wish;

        printf("Enter root info\n");
        scanf("%d",&e);

        pt->root=createNode(e);

        do{
                printf("Enter info\n");
                scanf("%d",&e);

                p=pt->root;
                q=NULL;

                while(p!=NULL)
                {
                        q=p;

                        if(e<p->info)
                                p=p->left;
                        else{
                                if(p->rthread)
                                        p=NULL;
                                else
                                        p=p->right;
                        }
                }
                if(e < q->info)
                        setLeft(q,e);
```

```
                else
                        setRight(q,e);

                printf("Do you wish to add another element\n");
                scanf("%d",&wish);
        }while(wish);
}

int main()
{
        TREE t;
        init(&t);
        create(&t);
        inOrder(&t);

        return 0;
}
```