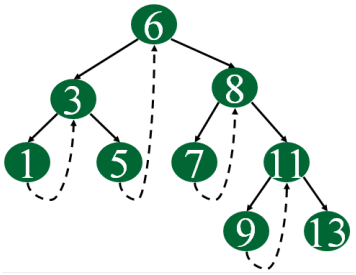
Convert a Binary Tree to Threaded binary tree

We have discussed Threaded Binary Tree. The idea of threaded binary trees is to make inorder traversal faster and do it without stack and without recursion. In a simple threaded binary tree, the NULL right pointers are used to store inorder successor. Where-ever a right pointer is NULL, it is used to store inorder successor.

Following diagram shows an example Single Threaded Binary Tree. The dotted lines represent threads.



Following is structure of single threaded binary tree.

```
struct Node
{
    int key;
    Node *left, *right;

    // Used to indicate whether the right pointer is a normal right
    // pointer or a pointer to inorder successor.
    bool isThreaded;
};
```

How to convert a Given Binary Tree to Threaded Binary Tree?

We basically need to set NULL right pointers to inorder successor. We first do an inorder traversal of the tree and store it in a queue (we can use a simple array also) so that the inorder successor becomes the next node. We again do an inorder traversal and whenever we find a node whose right is NULL, we take the front item from queuue and make it the right of current node. We also set isThreaded to true to indicate that the right pointer is a threaded link.

Following is the implementation of the above idea.

C++

```
/* C++ program to convert a Binary Tree to Threaded Tree */
#include <iostream>
#include <queue>
using namespace std;

/* Structure of a node in threaded binary tree */
struct Node
{
    int key;
    Node *left, *right;

    // Used to indicate whether the right pointer is a normal
    // right pointer or a pointer to inorder successor.
    bool isThreaded;
};

// Helper function to put the Nodes in inorder into queue
void populateQueue(Node *root, std::queue <Node *> *q)
{
    if (root == NULL) return;
    if (root->left)
        populateQueue(root->left, q);
```

```
    q->push(root);
    if (root->right)
        populateQueue(root->right, q);
}

// Function to traverse queue, and make tree threaded
void createThreadedUtil(Node *root, std::queue <Node *> *q)
{
    if (root == NULL) return;

    if (root->left)
        createThreadedUtil(root->left, q);
    q->pop();

    if (root->right)
        createThreadedUtil(root->right, q);

    // If right pointer is NULL, link it to the
    // inorder successor and set 'isThreaded' bit.
    else
    {
        root->right = q->front();
        root->isThreaded = true;
    }
}

// This function uses populateQueue() and
// createThreadedUtil() to convert a given binary tree
// to threaded tree.
void createThreaded(Node *root)
{
    // Create a queue to store inorder traversal
    std::queue <Node *> q;

    // Store inorder traversal in queue
    populateQueue(root, &q);

    // Link NULL right pointers to inorder successor
    createThreadedUtil(root, &q);
}

// A utility function to find leftmost node in a binary
// tree rooted with 'root'. This function is used in inOrder()
Node *leftMost(Node *root)
{
    while (root != NULL && root->left != NULL)
        root = root->left;
    return root;
}

// Function to do inorder traversal of a threadded binary tree
void inOrder(Node *root)
{
    if (root == NULL) return;

    // Find the leftmost node in Binary Tree
    Node *cur = leftMost(root);

    while (cur != NULL)
    {
        cout << cur->key << " ";

        // If this Node is a thread Node, then go to
        // inorder successor
        if (cur->isThreaded)
            cur = cur->right;

        else // Else go to the leftmost child in right subtree
            cur = leftMost(cur->right);
    }
}

// A utility function to create a new node
```

```
// A utility function to create a new node
Node *newNode(int key)
{
    Node *temp = new Node;
    temp->left = temp->right = NULL;
    temp->key = key;
    return temp;
}

// Driver program to test above functions
int main()
{
    /*       1
            / \
           2   3
          / \ / \
         4  5 6  7  */
    Node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);

    createThreaded(root);

    cout << "Inorder traversal of creeated threaded tree is\n";
    inOrder(root);
    return 0;
}
```

Java

```
// Java program to convert binary tree to threaded tree
import java.util.LinkedList;
import java.util.Queue;

/* Class containing left and right child of current
 node and key value*/
class Node
{
    int data;
    Node left, right;

    // Used to indicate whether the right pointer is a normal
    // right pointer or a pointer to inorder successor.
    boolean isThreaded;

    public Node(int item)
    {
        data = item;
        left = right = null;
    }
}

class BinaryTree
{
    Node root;

    // Helper function to put the Nodes in inorder into queue
    void populateQueue(Node node, Queue<Node> q)
    {
        if (node == null)
            return;
        if (node.left != null)
            populateQueue(node.left, q);
        q.add(node);
        if (node.right != null)
            populateQueue(node.right, q);
    }
```

```
}

// Function to traverse queue, and make tree threaded
void createThreadedUtil(Node node, Queue<Node> q)
{
    if (node == null)
        return;

    if (node.left != null)
        createThreadedUtil(node.left, q);
    q.remove();

    if (node.right != null)
        createThreadedUtil(node.right, q);

    // If right pointer is NULL, link it to the
    // inorder successor and set 'isThreaded' bit.
    else
    {
        node.right = q.peek();
        node.isThreaded = true;
    }
}

// This function uses populateQueue() and
// createThreadedUtil() to convert a given binary tree
// to threaded tree.
void createThreaded(Node node)
{
    // Create a queue to store inorder traversal
    Queue<Node> q = new LinkedList<Node>();

    // Store inorder traversal in queue
    populateQueue(node, q);

    // Link NULL right pointers to inorder successor
    createThreadedUtil(node, q);
}

// A utility function to find leftmost node in a binary
// tree rooted with 'root'. This function is used in inOrder()
Node leftMost(Node node)
{
    while (node != null && node.left != null)
        node = node.left;
    return node;
}

// Function to do inorder traversal of a threadded binary tree
void inOrder(Node node)
{
    if (node == null)
        return;

    // Find the leftmost node in Binary Tree
    Node cur = leftMost(node);

    while (cur != null)
    {
        System.out.print(" " + cur.data + " ");

        // If this Node is a thread Node, then go to
        // inorder successor
        if (cur.isThreaded == true)
            cur = cur.right;
        else // Else go to the leftmost child in right subtree
            cur = leftMost(cur.right);
    }
}

// Driver program to test for above functions
public static void main(String args[])
{
    BinaryTree tree = new BinaryTree();
```

```
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);

        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);
        tree.root.right.left = new Node(6);
        tree.root.right.right = new Node(7);

        tree.createThreaded(tree.root);
        System.out.println("Inorder traversal of created threaded tree");
        tree.inOrder(tree.root);
    }
}

// This code has been contributed by Mayank Jaiswal
```

Output:

```
Inorder traversal of creeated threaded tree is
4 2 5 1 6 3 7
```