**ADVANCED DATA STRUCTURE**

**GROUP A**

**ASSIGNMENT 2**

**BATCH B1**

**YEAR: 2017-18**

**COLLEGE: VIIT**

**Date Of Completion : 30/03/2018**

**Title:**

Threaded Binary Tree

**Problem Statement:**

Convert given binary tree into threaded binary tree. Analyze time and space complexity of

the algorithm.

**Objective:**

To convert binary tree to threaded binary tree and compare efficiency of both.

Software And Hardware Requirement:

1. 64-bit Open source Linux or its derivative.

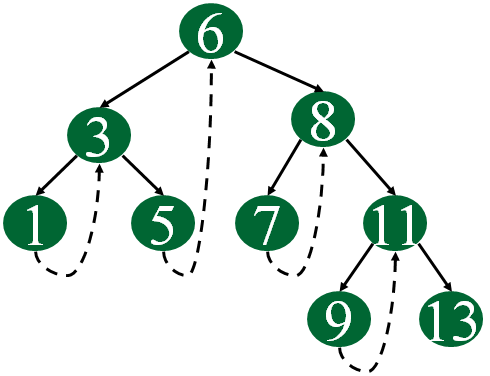
2. Open Source C++ Programming tool like G++/GCC.

**Theory:**

Inorder traversal of a Binary tree is either be done using recursion or with the use of a

auxiliary stack. The idea of threaded binary trees is to make inorder traversal faster and

do it without stack and without recursion.



A binary tree is made threaded by making all

right child pointers that would normally be NULL point to the inorder successor of the node

(if it exists). There are two types of threaded binary trees. Single Threaded: Where a

NULL right pointers is made to point to the inorder successor (if successor exists) Double

Threaded: Where both left and right NULL pointers are made to point to inorder predecessor

and inorder successor respectively. The predecessor threads are useful for reverse inorder

traversal and postorder traversal. The threads are also useful for fast accessing ancestors

of a node. Following diagram shows an example Single Threaded Binary Tree. The dotted

lines represent threads.

representation of a Threaded Node Following is C representation of a single threaded node.

struct Node

{

int data;

Node \*left, \*right;

bool rightThread;

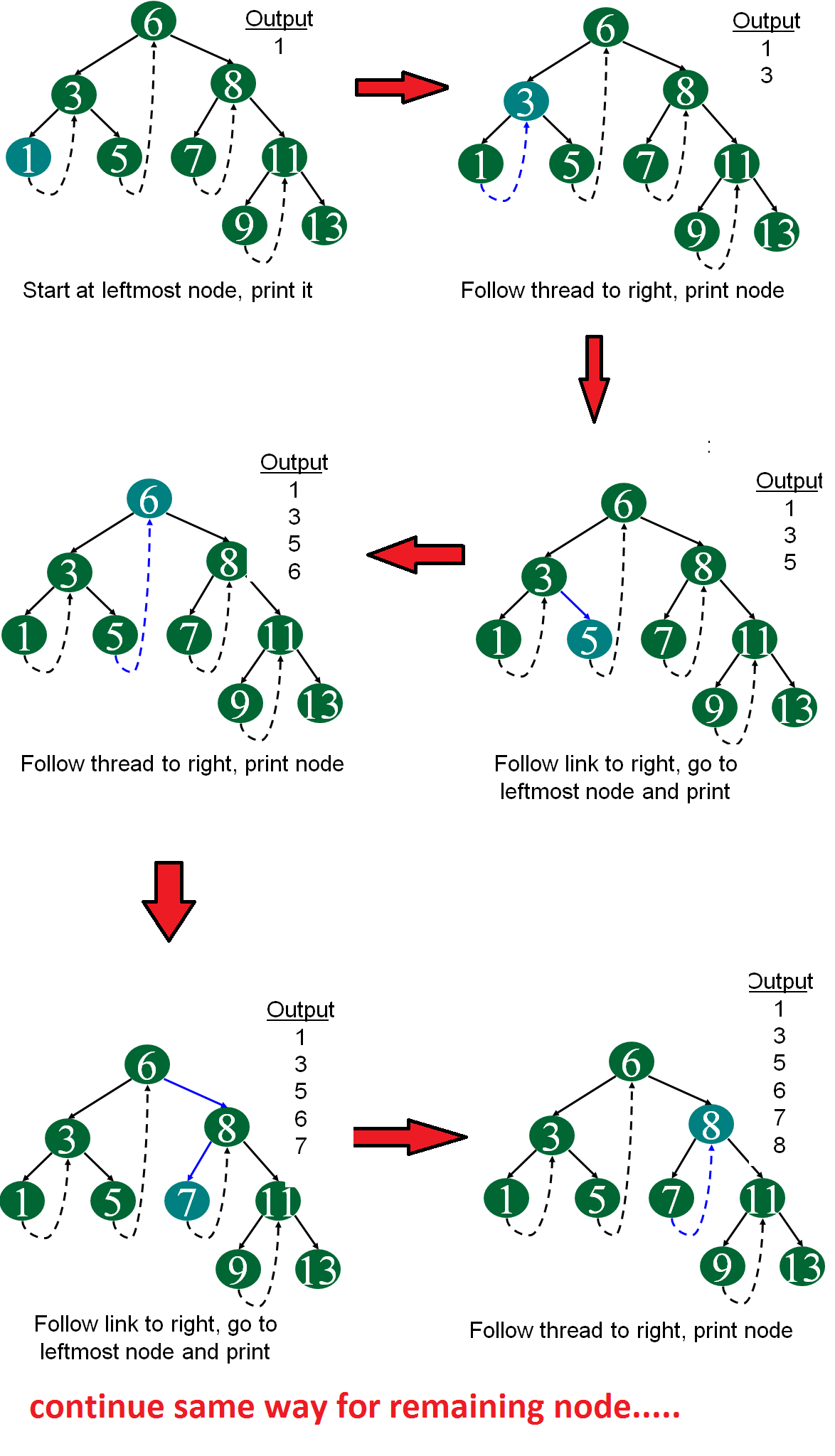
}

Run on IDE Since right pointer is used for two purposes, the boolean variable rightThread

is used to indicate whether right pointer points to right child or inorder successor. Similarly,

we can add leftThread for a double threaded binary tree.

Following diagram demonstrates inorder order traversal using threads



**Algorithm:**

1. Initialize curr as root

32. While curr is not NULL

If curr does not have left child

a) Print curr’s data

b) Go to the right, i.e., curr = curr->right

Else

a) Make curr as right child of the rightmost

node in curr’s left subtree

b) Go to this left child, i.e., curr = curr->left

**Code :**

/\*

\* BSTree.cpp

\*

\* Created on: 12 Jan 2017

\*

Author: batch2

\*/

#include<iostream>

using namespace std;

typedef struct node

{

int data;

struct node \*left;

struct node \*right;

}node;

class Tree

{

private:

int val;

char ch;

node \*head,\*New,\*temp;

public:

node \*Create\_Node();

node \*Create\_BST(node \*root);

void Inorder(node \*root);

void Insert\_Node(node \*root);

void Mirror(node \*root);

void Search(node \*root);

void MinEle(node \*root);

void MaxEle(node \*root);

int LongPath(node \*root);

void Mirror\_NC(node \*root);

4};

node \*Tree::Create\_Node()

{

cout<<"\nEnter Node-Data : ";

cin>>val;

New=new node;

New->data=val;

New->left=NULL;

New->right=NULL;

return New;

}

node \*Tree::Create\_BST(node \*root)

{

char ch1;

int cnt=0;

do

{

New=Create\_Node();

cnt++;

if(root==NULL)

{

root=New;

cout<<"\nRoot Node Created....!!!";

}

else

{

temp=root;

while(1)

{

if(temp->data < New->data)

{

if(temp->right==NULL)

{

temp->right=New;

cout<<New->data<<" Added to right of "<<temp->data;

break;

}

else

temp=temp->right;

}

else if(temp->data > New->data)

{

if(temp->left==NULL)

{

5temp->left=New;

cout<<New->data<<" Added to left of "<<temp->data;

break;

}

else

temp=temp->left;

}

else

{

cout<<"\nSame Data Not Allowed...!!!\n";

break;

}

}

}

cout<<"\nDo you want to Add More(y/n) : ";

cin>>ch1;

}while(ch1==’y’|| ch1==’Y’);

cout<<"\nTotal Nodes In Tree : "<<cnt;;

return root;

}

void Tree::Insert\_Node(node \*root)

{

New=Create\_Node();

if(root==NULL)

{

root=New;

cout<<"\nRoot Node Created....!!!";

}

else

{

temp=root;

while(1)

{

if(temp->data < New->data)

{

if(temp->right==NULL)

{

temp->right=New;

cout<<New->data<<" Added to right of "<<temp->data;

break;

}

else

temp=temp->right;

}

else if(temp->data > New->data)

6{

if(temp->left==NULL)

{

temp->left=New;

cout<<New->data<<" Added to left of "<<temp->data;

break;

}

else

temp=temp->left;

}

else

{

cout<<"\nSame Data Not Allowed...!!!\n";

break;

}

}

}

}

void Tree::Inorder(node \*root)

{

if(root==NULL)

return ;

if(root!=NULL)

{

Inorder(root->left);

cout<<root->data<<" ";

Inorder(root->right);

}

}

/\*void Tree::Mirror(node \*root)

{

if(root==NULL)

return ;

else

{

Mirror(root->left);

Mirror(root->right);

// Swapping The left n right //

temp=root->left;

root->left=root->right;

root->right=temp;

}

}\*/

void Tree::Mirror\_NC(node \*root)

{

if(root==NULL)

7return ;

else

{

// Swapping The left n right //

temp=root->left;

root->left=root->right;

root->right=temp;

}

node \*temp1;

temp1=root->left;

while(temp1!=NULL)

{

temp=temp1->left;

temp1->left=temp1->right;

temp1->right=temp;

temp1=temp1->left;

}

node \*temp2;

temp2=root->right;

while(temp2!=NULL)

{

temp=temp2->left;

temp2->left=temp2->right;

temp2->right=temp;

temp2=temp2->right;

}

}

void Tree::Search(node \*root)

{

int key;

int flag=0;

cout<<"\nEnter Data To Find : ";

cin>>key;

if(root->data==key)

{

cout<<"\nData Found...!!!";

flag=1;

return ;

}

else

{

temp=root;

while(temp!=NULL)

8{

if(temp->data==key)

{

cout<<"\nData Found...!!!";

return ;

}

if(temp->data < key)

temp=temp->right;

else if(temp->data > key)

temp=temp->left;

};

}

if(flag==0)

{

cout<<"\nData Not Found...!!!";

return ;

}

}

void Tree::MinEle(node \*root)

{

temp=root;

while(temp->left!=NULL)

temp=temp->left;

cout<<"\nMinimum Element : "<<temp->data<<endl;

temp=root;

while(temp->right!=NULL)

temp=temp->right;

cout<<"\nMaximum Element : "<<temp->data<<endl;

}

/\*void Tree::MinEle(node \*root)

{

temp=root;

if(temp->left!=NULL)

MinEle(temp->left);

cout<<"\nMinimum Element : "<<temp->data<<endl;

}

void Tree::MaxEle(node \*root)

{

temp=root;

if(temp->right!=NULL)

MaxEle(temp->right);

cout<<"\nMaximum Element : "<<temp->data<<endl;

}\*/

int Tree::LongPath(node \*root)

{

int cntL,cntR;

9if(root->left==NULL && root->right==NULL )

return 0;

else

{

cntL=LongPath(root->left);

cntR=LongPath(root->right);

if(cntR > cntL)

return (cntR);

else

return (cntL);

}

}

int main()

{

int ch;

Tree b;

node \*root=NULL;

do

{

cout<<"\n\n\t MENU \n";

cout<<"\n1.Create BST\n2.Insert Node\n3.Search Node\n4.Mirror Image\n5.MinMax Element\n6

cout<<"\n\tEnter your choice : ";

cin>>ch;

switch(ch)

{

case 0:

return 0;

case 1:

root=b.Create\_BST(root);

cout<<"\nInorder Traversal : ";

b.Inorder(root);

break;

case 2:

b.Insert\_Node(root);

cout<<"\nInorder Traversal : ";

b.Inorder(root);

break;

case 3:

b.Search(root);

break;

case 4:

//b.Mirror(root);

//cout<<"\nMirror BST Traversal : ";

b.Mirror\_NC(root);

cout<<"\nMirror BST Traversal(NC) : ";

b.Inorder(root);

break;

10case 5:

b.MinEle(root);

//b.MaxEle(root);

break;

case 6:

int height=b.LongPath(root);

cout<<"\nLongest Path : "<<height<<endl;

break;

}

}while(1);

return 0;

}

**Output :**

MENU

1.Create BST

2.Insert Node

3.Search Node

4.Mirror Image

5.MinMax Element

6.Longest Path

0.Exit

Enter your choice : 1

Enter Node-Data : 150

Root Node Created....!!!

Do you want to Add More(y/n) : y

Enter Node-Data : 110

110 Added to left of 150

Do you want to Add More(y/n) : y

Enter Node-Data : 50

50 Added to left of 110

Do you want to Add More(y/n) : y

Enter Node-Data : 40

40 Added to left of 50

Do you want to Add More(y/n) : y

11Enter Node-Data : 50

Same Data Not Allowed...!!!

Do you want to Add More(y/n) : y

Enter Node-Data : 122

122 Added to right of 110

Do you want to Add More(y/n) : n

Total Nodes In Tree : 6

Inorder Traversal : 40 50 110 122 150

MENU

1.Create BST

2.Insert Node

3.Search Node

4.Mirror Image

5.MinMax Element

6.Longest Path

0.Exit

Enter your choice : 2

Enter Node-Data : 150

Same Data Not Allowed...!!!

Inorder Traversal : 40 50 110 122 150

MENU

1.Create BST

2.Insert Node

3.Search Node

4.Mirror Image

5.MinMax Element

6.Longest Path

0.Exit

Enter your choice : 2

Enter Node-Data : 450

450 Added to right of 150

Inorder Traversal : 40 50 110 122 150 450

12MENU

1.Create BST

2.Insert Node

3.Search Node

4.Mirror Image

5.MinMax Element

6.Longest Path

0.Exit

Enter your choice : 3

Enter Data To Find : 50

Data Found...!!!

MENU

1.Create BST

2.Insert Node

3.Search Node

4.Mirror Image

5.MinMax Element

6.Longest Path

0.Exit

Enter your choice : 4

Mirror BST Traversal(NC) : 450 150 122 110 50 40

MENU

1.Create BST

2.Insert Node

3.Search Node

4.Mirror Image

5.MinMax Element

6.Longest Path

0.Exit

Enter your choice : 5

Minimum Element : 450

Maximum Element : 40

139

**Conclusion :**

After completing this assignment we were able to convert a BST to single threaded tree and

traverse it without using stack or recursion.