**PUNE INSTITUTE OF COMPUTER TECHNOLOGY, PUNE - 411043 Department of Computer Engineering** 

**S.No.-27, Pune Satara Road, Dhankawadi, Pune-411043**

**Data Structures and Algorithms Laboratory**

**Batch-IV (H4)**

**Name: Rushikesh Vidye**

**Roll No.: 21471**

**Class: SE4**

**Assignment No. 4**

**Title:** Create an inordered threaded binary tree and perform inorder and preorder traversals. Analyse time and space complexity of the algorithm.

**Software Requirement:**

a) OS : Microsoft Windows 10.

b) Browser: Google Chrome.

c) VS Code.

**Hardware Requirement:**

a) Processor: Intel Core i5-8265U.

b) Ram: 8 GB DDR4 2800Mhz.

**Theory:**

Threaded binary tree is a simple binary tree but they have a speciality that null pointers of leaf node of the binary tree is set to inorder predecessor or inorder successor.The main idea behind setting such a structure is to make the inorder and preorder traversal of the tree faster without using any additional data structure(e.g auxiliary stack) or memory to do the traversal.

Types of Threaded Binary Tree

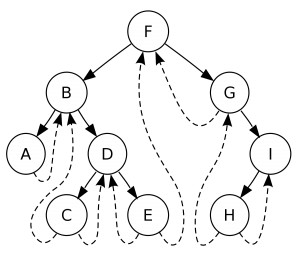
There are two types of threaded binary tree:

1. Single Threaded Binary Tree

2. Double Threaded Binary Tree

Single Threaded Binary Tree: Here only the right NULL pointer is made to point to an inorder successor.

Double Threaded Binary Tree: Here both the right as well as the left NULL pointers are made to point inorder successor and inorder predecessor respectively.



**Insertion in Threaded Binary Search Tree:**

Insertion in Binary threaded tree is similar to insertion in binary tree but we will have to adjust the threads after insertion of each element.

Example:

class Node

{

Node \*left, \*right;

int info;

bool lthread;

bool rthread;

}

There can be three cases during insertion:

Case 1: Insertion in empty tree

Both left and right pointers of tmp will be set to NULL and a new node becomes the root. root = tmp;

tmp -> left = NULL;

tmp -> right = NULL;

Case 2: When new node inserted as the left child

After inserting the node at its proper place we have to make its left and right threads points to inorder predecessor and successor respectively. The node which was in order successor. So the left and right threads of the new node will be

tmp -> left = par ->left;

tmp -> right = par;

Before insertion, the left pointer of parent was a thread, but after insertion it will be a link pointing to the new node.

par -> lthread = false;

par -> left = temp;

Case 3: When new node is inserted as the right child

The parent of tmp is its inorder predecessor. The node which was inorder successor of the parent is now the inorder successor of this node tmp. So the left and right threads of the new node will be

tmp -> left = par;

tmp -> right = par -> right;

Before insertion, the right pointer of parent was a thread, but after insertion it will be a link pointing to the new node.

par -> rthread = false;

par -> right = tmp;

**Algorithm :**

**Main() :**

1. Start
2. Declare variables and objects.
3. Accept  element from user to crate tree

Accept choice from user whether they want to continue to add elements(y/n)

* + - 1. If ch=y
    1. Element accepted
    2. Otherwise
       1. Go to step 3

1. Display inorder traversal by calling method inorder()
2. Display preorder traversal by calling method preorder()
3. Stop

**Preorder()**

1. Start
2. While temp is not equal to head repeat steps 3 to 7
3. if flag is set to 0
4. Print data of temp
5. if lth of temp and flag are set to 0

Set temp to left of temp

1. otherwise if rth of temp is zero
   * 1. Set temp to right of temp
     2. Initialize flag to 0
2. else Set temp to right of temp and set flag to 1
3. Stop

**Inorder()**

1. Start
2. While temp is not equal to head repeat steps 2 to 3
3. Set temp to inordersucc(temp)
4. If temp is not equal to head

Print data of temp

1. Stop

**Time Complexity:**

|  |  |
| --- | --- |
| Sr. no. | Method Complexity |
| 1 | inorder() O(n) |
| 2 | preorder() O(n) |
| 3 | inorderSuccessor() O(n) |

**Test Cases:**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr.no | Input | Output | Result |
| 1 | Enter number data: 6,3,8,1,5,7,11,9,13 | Root created | pass |
| 2 | Inorder traversal: 1,3,5,6,7,8,9,11,13 | 1,3,5,6,7,8,9,11,13 | pass |
| 3 | Preorder traversal:  6,3,1,5,8,7,11,9,14 | 6,3,1,5,8,7,11,9,14 | pass |

**Conclusion:**

Developed an in ordered threaded binary tree and performed inorder and preorder traversals.

**Code:**

|  |
| --- |
| #pragma once  #include<iostream>  #include<stdio.h>  #include<conio.h>  using namespace std;  class Node  {      public:      int data;      Node\* left, \* right;      bool leftTh, rightTh;      friend class ThreadedTree;      Node(int ele){          this->data = ele;          this->left = this->right = NULL;          this->leftTh = this->rightTh = true;}      ~Node(){}};  class ThreadedTree{  public:      Node\* root;      ThreadedTree();      Node\* AddNode(int ele);      Node\* inorderSuccessor(Node\* ptr);      void Inorder(Node\* root);      void PreOrder(Node\* root);};  ThreadedTree::ThreadedTree(){}  Node\* ThreadedTree::AddNode(int ele){      Node\* newnode, \*move,\*par;      newnode = new Node(ele);      move = root;      par = NULL;      while(move!=NULL){          if(ele == move->data){              printf("Duplicate");              return root;}          par = move;          if(ele < move->data){              if (move -> leftTh == false)                      move = move-> left;              else                  break;}          else{              if (move->rightTh == false)                  move = move-> right;              else                  break;}}      if(move == NULL){          root = newnode;          newnode->left = newnode->right = NULL;}      else if(ele < move->data){          newnode->left = move->left;          newnode->right = move;          move->leftTh = false;          move->left = newnode;}  else{          newnode->left = move;          newnode->right = move->right;          move->rightTh = false;          move->right = newnode;}      return root;}  Node\* ThreadedTree::inorderSuccessor(Node\* ptr){      if(ptr->rightTh)          return ptr->right;      ptr = ptr->right;      while(!ptr->leftTh)          ptr = ptr->left;      return ptr;}  void ThreadedTree::Inorder(Node\* root){      Node \*move = root;      while(!move->leftTh)          move = move->left;      while (move != NULL){          printf(" %d\n",move->data);          move = inorderSuccessor(move);}}  void ThreadedTree::PreOrder(Node\* root){      Node \*move = root;      printf("%d\t",move->data);      if(!move->leftTh)          PreOrder(move->left);      if(!move->rightTh)          PreOrder(move->right);}  #include"header.h"  int main(void) {      ThreadedTree tree;      tree.root = tree.AddNode(6);      tree.root = tree.AddNode(3);      tree.root = tree.AddNode(8);      tree.root = tree.AddNode(1);      tree.root = tree.AddNode(5);      tree.root = tree.AddNode(7);      tree.root = tree.AddNode(11);      tree.root = tree.AddNode(9);      tree.root = tree.AddNode(13);      printf("Ioder Traversal\n");      tree.Inorder(tree.root);      printf("preorder traversal\n");      tree.PreOrder(tree.root);} |

**Output:**

|  |
| --- |
| Ioder Traversal  1 3 5 6 7 8 9 11 13  preorder traversal  6 3 1 5 8 7 11 9 13 |