Roll No. 21118
Date: 19-May-2021

PICT, PUNE

Subject: Data Structures and Algorithms

Assignment: 03

Problem Definition:

Create an ~~array~~ inordered threaded binary tree and perform inorder & preorder traversal. Analyze time & space complexity of the algorithm

Objectives:

- To understand threaded binary trees
- To understand & implement various operations on threaded binary tree - creation, traversal, deletion.
- To avoid overhead of stack data structure while performing preorder & inorder traversal

Outcomes:

- To implement threaded binary trees & its operations
- To use classes in c++ & pointers.

Hardware requirements:

Manufatures: Acer Inc.
System type: x-64 type, PC
Processor & RAM: Intel(R) Core i5-8265U @ 1.60 GHz & 8 GB of RAM

Software requirements:

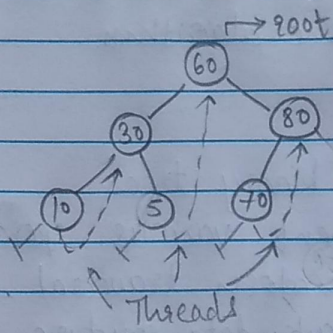
Operating System: MS Windows 10
IDE: Eclipse (2020 Edition)

Theory:

A threaded binary tree is used to make improvements in binary tree traversal by eliminating stack requirements. It is faster.

PICT, PUNE

A binary tree is made threaded by making all right child pointers point to the inorder successor of the node (if exists)



ADT:

```
class Node {
 private:
    int data;
    Node* lchild, *rchild;
    bool lTh, rTh;
};

class TBST {
 private:
    Node *root, *head;
 public:
    Node* getRoot();
    void InsertNode(int x);
    void ThInorder();
    void ThPreorder();
    bool Search();
    void delete();
}
```

Pseudocode:

```
insert(key) {   //inserting key in threaded BST
  if (root == NULL) {
      root = new Node(x);
      root->lchild = root->rchild = head;
  }
  else {
      Node *curr = root, *prev = NULL;
      while (curr != head) {
          prev = curr;
          if (x < curr->data) {
              if (curr->lTh == false)
                  curr = curr->lchild;
              else break;
          }
          else if (x > curr->data) {
              if (curr->rTh == false)
                  curr = curr->rchild;
              else break;
          }
          else if (x == curr->data)
              return;
      }
      curr = new Node(x);
      if (x < prev->data) {
          curr->rchild = prev;
          curr->lchild = prev->lchild;
          prev->lchild = curr;
          prev->lTh = false;
      }
```

```
        else if (x > prev->data) {
            curr->lchild = prev;
            curr->rchild = prev->rchild;
            prev->rchild = curr;
            prev->RTh = false;
        }
    }

    ThInOrder() {    //Inorder traversal
        if (root == NULL) {
            printf("EMPTY TREE")
            return;
        }
        Node* curr = root;
        while (curr->LTh == false)
            curr = curr->lchild;
        while (curr != head) {
            print(curr->data);
            if (curr->RTh == false) {
                curr = curr->rchild;
                while (curr->LTh == false)
                    curr = curr->lchild;
            }
            else curr = curr->rchild;
        }
    }

    ThPreorder() {    //Preorder traversal
        if (root == NULL) {
            printf("EMPTY TREE")
            return;
        }
```

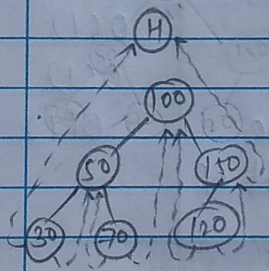
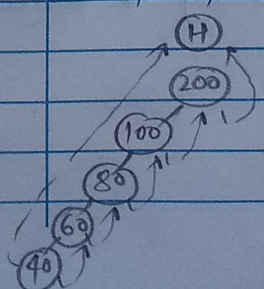
```
Node* curr = &ob;
while(curr != head) {
    printf(curr->data)
    if(curr->lTh == false)
        curr = curr->lchild;
    else {
        if(curr->rTh == false)
            curr = curr->rchild;
        else {
            while (curr != head && curr->rTh == true)
                curr = curr->rchild;
            if(curr == head)
                break;
            curr = curr->rchild;
        }
    }
}

deleteNode(int x) {
    Node* curr = NULL, *parent = NULL;
    if(!search(&ob, x, curr, parent)) {
        printf("NOT FOUND");
        return;
    }
    if(curr->lTh == false && curr->rTh == false) { // Intermediate node
        Node* temp = curr->rchild;
        parent = curr;
        while(temp->lTh == false) {
            parent = temp;
            temp = temp->lchild;
        }
```

```
        curr->data = temp->data;
        x = temp->data;
        curr = temp;
    }
    else
    if (curr->lTh == true && curr->rTh == true) { // Leaf Node
        if(curr == parent->lchild) {
            parent->lchild = curr->lchild;
            parent->lTh = true;
        }
        else if (curr == parent->rchild) {
            parent->rchild = curr->rchild;
            parent->rTh = true;
        }
        delete curr;
    }
    else if (curr->lTh == false && curr->rTh == true) {
        Node* temp = curr->lchild;
        if (parent->lchild == curr)
            parent->lchild = temp;
        else parent->rchild = temp;
        while (temp->rTh == false)
            temp = temp->rchild;
        temp->rchild = curr->rchild;
        delete curr;
    }
    else if (curr->lTh == true && curr->rTh == false) {
        Node* temp = curr->rchild;
        if (parent->lchild == curr)
            parent->lchild = temp;
```

PICT, PUNE

```
else parent->rchild = temp;
while (temp->lTh == false)
    temp = temp->lchild;
temp->lchild = curr->lchild;
delete curr;
}
}
```

Test Cases:

No	Testcase Description	Expected O/p	Actual O/p	Results
1>	Insert nodes 100, 150, 120, 50, 30, 70 	Threaded BST should be Created	Threaded BST is Created	Passed
	Inorder Traversal	30, 50, 70, 100, 120, 150	30, 50, 70, 100, 120, 150	Passed
	Preorder Traversal	100, 50, 30, 70 150, 120	100, 50, 30, 70, 150, 120	Passed
2>	Insert Nodes 200, 100, 80, 60, 40 	Threaded BST should be created	Threaded BST is Created	Passed

PICT, PUNE

Inorder Traversal	40,60,80,100,200	40,60,80,100,200	Passed
Preorder Traversal	200,100,80,60,40	200,100,80,60,40	Passed

Algorithm Analysis (Time & Space Complexity)

Method.	Time complexity	Space Complexity.
1> InsertNode()	O(h) where h is height of tree.	No extra space required hence O(1)
2> InorderTraversal()	O(n)	O(1)
3> PreorderTraversal()	O(n)	O(1)
4> DeleteNode()	O(h) where h is height of tree	O(1)

Applications of Threaded Binary Tree:

1> Threaded binary trees are used for various searching algorithms.
2> They are used for avoiding auxiliary stack for traversal of binary trees.
3> They are also useful for maintaining a sorted stream of data.

PICT, PUNE

Conclusion:

Through this assignment, we learned about threaded binary trees & their usage, along with their traversals.