|  |  |
| --- | --- |
| **Practical No: 11** | |
|  | |
| **Explanation/ Stepwise-Procedure/ Algorithm** | Write a menu-driven program to Create and display a threaded binary tree. |
| **Theory:** | Write the following questions  * What is a threaded Binary tree   A **Threaded Binary Tree (TBT)** is a type of binary tree in which "threads" replace certain NULL pointers to provide efficient in-order traversal of the tree without using recursion or a stack.   * Discuss different types of traversal of threaded binary tree   **In-order Traversal**: Efficiently uses threads for traversal without recursion or a stack.  **Pre-order Traversal**: Possible but less efficient than in-order, especially in a TBT.  **Post-order Traversal**: Complex in TBTs and often requires auxiliary tracking, so it’s less common.  In general, in-order traversal is the primary use case for threaded binary trees, as it provides the greatest efficiency due to direct pointers for navigating the tree structure.   * Write algorithms of different traversals of threaded binary tree   **In order Traversal**  void inorder(Node\* root) {  Node\* current = root;  // Step 1: Find the leftmost node in the tree  while (current->lThread == 0)  current = current->left;  // Step 2: Traverse the tree  while (current != NULL) {  // Visit the current node  printf("%d ", current->data);  // Step 3: If there is a right thread, follow it to the successor  if (current->rThread == 1)  current = current->right;  else {  // Step 4: Otherwise, go to the leftmost node in the right subtree  current = current->right;  while (current != NULL && current->lThread == 0)  current = current->left;  }  }  }  **Pre order Traversal**  void preorder(Node\* root) {  Node\* current = root;  // Step 1: Start traversal  while (current != NULL) {  // Visit the current node  printf("%d ", current->data);  // Step 2: If there is a left child, move to it  if (current->lThread == 0)  current = current->left;  else if (current->rThread == 0) {  // Step 3: If no left child, move to the right child if it exists  current = current->right;  } else {  // Step 4: If right child is also a thread, follow the thread  while (current != NULL && current->rThread == 1)  current = current->right;  // Move to the right child if it exists  if (current != NULL)  current = current->right;  }  }  }  Implementing **post-order traversal** in a **Threaded Binary Tree (TBT)** is indeed challenging because threaded binary trees are primarily designed to optimize in-order traversal, not post-order traversal. Threads in a TBT typically link nodes to their in-order successors, which doesn’t directly support the *left subtree → right subtree → root* order required in post-order traversal.  To perform post-order traversal in a TBT, we need to either:   * Modify the structure of the TBT to include parent pointers. * Use a stack or recursion specifically for post-order traversal, which partially negates the benefit of threading. * Discuss time complexity of TBT. * **Threaded binary trees** significantly improve in-order traversal time by reducing the need for extra space (stacks or recursion) and allowing O(1)O(1)O(1) moves to in-order successors or predecessors. * For insertion, search, and deletion, the time complexity is primarily dependent on the height of the tree. * **Balanced TBTs** offer efficient operations at O(logn)O(\log n)O(logn), whereas unbalanced TBTs may reach O(n) complexity.   4o |
| **Source Code/Algorithm/Flow Chart:** | #include <stdio.h>  #include <stdlib.h>  typedef struct Node {  int data;  struct Node \*left, \*right;  int lThread, rThread; // Flags for threads  } Node;  // Function to create a new node  Node\* createNode(int data) {  Node\* newNode = (Node\*)malloc(sizeof(Node));  newNode->data = data;  newNode->left = newNode->right = NULL;  newNode->lThread = newNode->rThread = 1; // Initially, threads are true (pointing to NULL)  return newNode;  }  // Insert a new node into the threaded binary tree  Node\* insert(Node\* root, int data) {  Node\* newNode = createNode(data);  if (root == NULL) {  root = newNode;  return root;  }  Node\* parent = NULL;  Node\* current = root;  // Find the appropriate position to insert the node  while (current != NULL) {  parent = current;  if (data < current->data) {  if (current->lThread == 0)  current = current->left;  else  break;  } else {  if (current->rThread == 0)  current = current->right;  else  break;  }  }  if (data < parent->data) {  newNode->left = parent->left;  newNode->right = parent;  parent->lThread = 0;  parent->left = newNode;  } else {  newNode->right = parent->right;  newNode->left = parent;  parent->rThread = 0;  parent->right = newNode;  }  return root;  }  // Inorder traversal of a threaded binary tree  void inorder(Node\* root) {  if (root == NULL) {  printf("Tree is empty.\n");  return;  }  Node\* current = root;  // Find the leftmost node  while (current->lThread == 0)  current = current->left;  // Traverse the threaded binary tree  while (current != NULL) {  printf("%d ", current->data);  // If this is a thread, follow it to the successor  if (current->rThread == 1)  current = current->right;  else {  // Otherwise, find the leftmost node in the right subtree  current = current->right;  while (current != NULL && current->lThread == 0)  current = current->left;  }  }  printf("\n");  }  // Display menu  void displayMenu() {  printf("\nThreaded Binary Tree Operations:\n");  printf("1. Insert\n");  printf("2. Inorder Traversal\n");  printf("3. Exit\n");  printf("Enter your choice: ");  }  int main() {  Node\* root = NULL;  int choice, data;  while (1) {  displayMenu();  scanf("%d", &choice);  switch (choice) {  case 1:  printf("Enter data to insert: ");  scanf("%d", &data);  root = insert(root, data);  break;  case 2:  printf("Inorder Traversal: ");  inorder(root);  break;  case 3:  exit(0);  default:  printf("Invalid choice. Please try again.\n");  }  }  return 0;  } |
| **Output Screenshots (if applicable)** |  |
| **Conclusion** | Thus, we have studied and implemented the Threaded Binary Tree and traversals on it. |
| **Post Lab Questions:** | * What are the advantages and disadvantages of TBT * What are applications of TBT |