Practice Questions : Binary Tree

Data Structures and Algorithms

6 March, 2025

Questions

1. **Creating a Node**
    - Define a structure/class to represent a node in a binary tree.
2. **Creating a Tree**
    - Implement a function to create a binary tree.
3. **Hardcoding Elements to Leaves of the Node**
    - Write code to manually assign values to leaf nodes of a binary tree.
4. **Tree Traversals**
    - Write functions to perform the following traversals:
        - In-order traversal
        - Pre-order traversal
        - Post-order traversal
5. **Finding Minimum and Maximum Elements in a Tree**
    - Implement a function to find the minimum element in a binary search tree.
    - Implement a function to find the maximum element in a binary search tree.
6. **Counting Elements in the Tree**
    - Write a function to count the number of nodes in a binary tree.
    - Analyze the time complexity of your implementation.
7. **Finding the Height of the Tree**
    - Implement a function to determine the height of a binary tree.
    - What is the base case for a recursive implementation?

Solution Code in C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>

typedef struct BTNode {
    int nData;
    struct BTNode *pParent;
    struct BTNode *pLeft;
    struct BTNode *pRight;
} BTNode;

void PreOrderTraversal(BTNode *pRoot){
    if (pRoot!=NULL) {
        printf ("%d ", pRoot->nData);
        PreOrderTraversal (pRoot->pLeft);
        PreOrderTraversal (pRoot->pRight);
    }
}

void InOrderTraversal(BTNode *pRoot){
    if (pRoot!=NULL) {
        InOrderTraversal (pRoot->pLeft);
        printf ("%d ", pRoot->nData);
        InOrderTraversal (pRoot->pRight);
    }
}

void PostOrderTraversal(BTNode *pRoot){
    if (pRoot!=NULL) {
        PostOrderTraversal (pRoot->pLeft);
        PostOrderTraversal (pRoot->pRight);
        printf ("%d ", pRoot->nData);
    }
}

int count (BTNode *pRoot) {
    if (pRoot==NULL){
        return 0;}
    else{
        return 1 + count(pRoot->pLeft) + count(pRoot->pRight);
    }
}

int height (BTNode *pRoot) {
    if (pRoot==NULL){
        return 0;}
    else{
        return 1 + max(height(pRoot->pLeft), height (pRoot->pRight));}
}

int max(int a, int b) {
    return a>=b ? a : b;
}
```

```
int min(int a, int b) {
    return a<=b ? a : b;
}

int TreeMaxBT(BTNode *pRoot) {
    // Base case: empty tree
    if (pRoot == NULL) {
        return INT_MIN; // Use INT_MIN from limits.h
    }

    // If leaf node, return its value
    if (pRoot->pLeft == NULL && pRoot->pRight == NULL) {
        return pRoot->nData;
    }

    // Find max value in this subtree
    int res = pRoot->nData;
    int leftMax = (pRoot->pLeft != NULL) ? TreeMaxBT(pRoot->pLeft) : INT_MIN;
    int rightMax = (pRoot->pRight != NULL) ? TreeMaxBT(pRoot->pRight) : INT_MIN;

    return max(res, max(leftMax, rightMax));
}

int TreeMinBT(BTNode *pRoot) {
    // Base case: empty tree
    if (pRoot == NULL) {
        return INT_MAX; // Use INT_MAX from limits.h
    }

    // If leaf node, return its value
    if (pRoot->pLeft == NULL && pRoot->pRight == NULL) {
        return pRoot->nData;
    }

    // Find min value in this subtree
    int res = pRoot->nData;
    int leftMin = (pRoot->pLeft != NULL) ? TreeMinBT(pRoot->pLeft) : INT_MAX;
    int rightMin = (pRoot->pRight != NULL) ? TreeMinBT(pRoot->pRight) : INT_MAX;

    return min(res, min(leftMin, rightMin));
}

int main(){
    //MAKING A RANDOM BINARY TREE
    {BTNode *BT = malloc(sizeof(BTNode));
    BT->nData=15;
    BTNode *BTL1 = malloc(sizeof(BTNode));
    BTL1->nData=13;
    BTNode *BTL2 = malloc(sizeof(BTNode));
    BTL2->nData=8;
    BTNode *BTL3 = malloc(sizeof(BTNode));
    BTL3->nData=9;
    BTNode *BTL4 = malloc(sizeof(BTNode));
    BTL4->nData=33;
    BTNode *BTL5 = malloc(sizeof(BTNode));
    BTL5->nData=69;
    BTNode *BTL6 = malloc(sizeof(BTNode));
    BTL6->nData=120;
    BTNode *BTR1 = malloc(sizeof(BTNode));
    BTR1->nData=18;
    BTNode *BTR2 = malloc(sizeof(BTNode));
    BTR2->nData=12;
    BTNode *BTR3 = malloc(sizeof(BTNode));
    BTR3->nData=5;
    BTNode *BTR4 = malloc(sizeof(BTNode));
    BTR4->nData=6;
    BTNode *BTR5 = malloc(sizeof(BTNode));
    BTR5->nData=34;

    BT->pLeft=BTL1, BT->pRight=BTR1, BT->pParent=NULL;
    BTL1->pLeft=BTL2, BTL1->pRight=BTR2, BTL1->pParent=BT;
    BTL2->pLeft=BTL3, BTL2->pRight=BTR3, BTL2->pParent=BTL1;
    BTL3->pLeft=NULL, BTL3->pRight=NULL, BTL3->pParent=BTL2;
    BTR3->pLeft=NULL, BTR3->pRight=BTR4, BT->pParent=BTL2;
    BTR4->pLeft=NULL, BTR4->pRight=NULL, BTR4->pParent=BTR3;
    BTR2->pLeft=NULL, BTR2->pRight=NULL, BTR2->pParent=BTL1;
    BTR1->pLeft=BTL4, BTR1->pRight=BTR5, BTR1->pParent=BT;
    BTL4->pLeft=NULL, BTL4->pRight=NULL, BTL4->pParent=BTR1;
    BTR5->pLeft=BTL5, BTR5->pRight=NULL, BTR5->pParent=BTR1;
    BTL5->pLeft=BTL6, BTL5->pRight=NULL, BTL5->pParent=BTR5;
    BTL6->pLeft=NULL, BTL6->pRight=NULL, BTL6->pParent=BTL5;
    }

    // use the functions above on the given tree
    printf("PreOrder Traversal: ");
    PreOrderTraversal(BT);
    printf("\nInOrder Traversal: ");
    InOrderTraversal(BT);
    printf("\nPostOrder Traversal: ");
    PostOrderTraversal(BT);
    printf("\nCount: %d", count(BT));
    printf("\nHeight: %d", height(BT));
    printf("\nTreeMaxBT: %d", TreeMaxBT(BT));
    printf("\nTreeMinBT: %d", TreeMinBT(BT));



    return 0;
}
```