GROUP MEMBERS:

1. MWANGI ANN MWENDE SCT222-0302/2022
2. BETH WAMUGWE KAMAU SCT222-0297/2022
3. BRENDA MAINA SCT222-0194/2021

You are required to write a program in C programming language that has a function named bst\_construct(). This function receives two arrays from main() function storing in-order and post-order traversal outputs shown below. The function then construct a binary search tree (bst) using the in-order and post-order traversal outputs passed to it by main() function. The bst\_construct() function then returns address to the root node back to the main() function, the caller function. Using the address of the root node returned, the main() function traverse the created binary search tree (bst) in breadth-first-search (DFS) traversal while printing each element on standard output device, i.e. computer screen.

Post-order: 5, 15, 10, 25, 45, 30, 20

In-order: 5, 10, 15, 20, 25, 30, 45

SOLUTION

#include <stdio.h>

#include <stdlib.h>

// Structure for a node of the binary search tree

typedef struct TreeNode {

int data;

struct TreeNode \*left;

struct TreeNode \*right;

} TreeNode;

// Function to create a new node

TreeNode\* createNode(int data) {

TreeNode\* newNode = (TreeNode\*)malloc(sizeof(TreeNode));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

// Function to find index of a value in an array

int findIndex(int arr[], int start, int end, int value) {

for (int i = start; i <= end; i++) {

if (arr[i] == value)

return i;

}

return -1; // Value not found

}

// Function to construct BST from in-order and post-order traversals

TreeNode\* bstConstruct(int inOrder[], int postOrder[], int inStart, int inEnd, int\* postIndex) {

if (inStart > inEnd)

return NULL;

// Create new node with the current value from postOrder array

TreeNode\* node = createNode(postOrder[\*postIndex]);

(\*postIndex)--;

// If it's a leaf node, return

if (inStart == inEnd)

return node;

// Find index of current node in inOrder array

int inIndex = findIndex(inOrder, inStart, inEnd, node->data);

// Construct right subtree first

node->right = bstConstruct(inOrder, postOrder, inIndex + 1, inEnd, postIndex);

// Construct left subtree

node->left = bstConstruct(inOrder, postOrder, inStart, inIndex - 1, postIndex);

return node;

}

// Function to perform breadth-first-search traversal of the binary search tree

void bfsTraversal(TreeNode\* root) {

if (root == NULL)

return;

// Create a queue for BFS

TreeNode\* queue[100]; // Assuming maximum 100 nodes

int front = -1, rear = -1;

queue[++rear] = root;

while (front != rear) {

TreeNode\* current = queue[++front];

printf("%d ", current->data);

if (current->left != NULL)

queue[++rear] = current->left;

if (current->right != NULL)

queue[++rear] = current->right;

}

}

int main() {

int inOrder[] = {5, 10, 15, 20, 25, 30, 45};

int postOrder[] = {5, 15, 10, 25, 45, 30, 20};

int n = sizeof(inOrder) / sizeof(inOrder[0]);

int postIndex = n - 1; // Index of last element in postOrder array

// Construct the binary search tree

TreeNode\* root = bstConstruct(inOrder, postOrder, 0, n - 1, &postIndex);

// Print the constructed binary search tree

printf("BST Construction:\n");

printf("In-order traversal: ");

for (int i = 0; i < n; i++) {

printf("%d ", inOrder[i]);

}

printf("\nPost-order traversal: ");

for (int i = 0; i < n; i++) {

printf("%d ", postOrder[i]);

}

printf("\n");

// Perform breadth-first-search traversal

printf("BFS Traversal:\n");

bfsTraversal(root);

return 0;

}