/\*BST USING LL AND ARRAY\*/

#include <iostream>

#include <vector>

#include <algorithm> // For std::sort

using namespace std;

// Node structure for the BST

struct Node {

    int data;

    Node\* left;

    Node\* right;

    Node(int val) {

        data = val;

        left = nullptr;

        right = nullptr;

    }

};

// Node structure for linked list

struct ListNode {

    int data;

    ListNode\* next;

    ListNode(int val) {

        data = val;

        next = nullptr;

    }

};

// Function to insert a node into the BST

Node\* insert(Node\* root, int val) {

    if (root == nullptr) {

        return new Node(val);

    }

    if (val < root->data) {

        root->left = insert(root->left, val);

    } else {

        root->right = insert(root->right, val);

    }

    return root;

}

// Function to construct a BST from a sorted array

Node\* constructBSTFromArray(const vector<int>& arr, int start, int end) {

    if (start > end) {

        return nullptr;

    }

    int mid = (start + end) / 2;

    Node\* root = new Node(arr[mid]);

    root->left = constructBSTFromArray(arr, start, mid - 1);

    root->right = constructBSTFromArray(arr, mid + 1, end);

    return root;

}

// Function to construct a BST from a linked list

Node\* constructBSTFromLinkedList(ListNode\* head) {

    vector<int> arr;

    ListNode\* curr = head;

    while (curr) {

        arr.push\_back(curr->data);

        curr = curr->next;

    }

    // Sorting the array

    sort(arr.begin(), arr.end());

    // Constructing BST from sorted array

    return constructBSTFromArray(arr, 0, arr.size() - 1);

}

// Function to traverse the BST in inorder (left-root-right)

void inorderTraversal(Node\* root) {

    if (root == nullptr) {

        return;

    }

    inorderTraversal(root->left);

    cout << root->data << " ";

    inorderTraversal(root->right);

}

// Function to traverse the BST in preorder (root-left-right)

void preorderTraversal(Node\* root) {

    if (root == nullptr) {

        return;

    }

    cout << root->data << " ";

    preorderTraversal(root->left);

    preorderTraversal(root->right);

}

// Function to traverse the BST in postorder (left-right-root)

void postorderTraversal(Node\* root) {

    if (root == nullptr) {

        return;

    }

    postorderTraversal(root->left);

    postorderTraversal(root->right);

    cout << root->data << " ";

}

int main() {

    // Accepting user input for linked list

    int n;

    cout << "Enter the number of elements for the linked list: ";

    cin >> n;

    cout << "Enter the elements for the linked list: ";

    ListNode\* head = nullptr;

    ListNode\* tail = nullptr;

    for (int i = 0; i < n; ++i) {

        int element;

        cin >> element;

        ListNode\* newNode = new ListNode(element);

        if (!head) {

            head = newNode;

            tail = newNode;

        } else {

            tail->next = newNode;

            tail = newNode;

        }

    }

    // Displaying linked list

    cout << "Linked list: ";

    ListNode\* curr = head;

    while (curr) {

        cout << curr->data << " ";

        curr = curr->next;

    }

    cout << endl;

    // Constructing BST from linked list

    Node\* root = constructBSTFromLinkedList(head);

    // Printing traversals of BST constructed from linked list

    cout << "Preorder traversal of BST constructed from linked list: ";

    preorderTraversal(root);

    cout << endl;

    cout << "Inorder traversal of BST constructed from linked list: ";

    inorderTraversal(root);

    cout << endl;

    cout << "Postorder traversal of BST constructed from linked list: ";

    postorderTraversal(root);

    cout << endl;

    // Displaying array

    cout << "Enter the number of elements for the array: ";

    cin >> n;

    cout << "Enter the elements for the array: ";

    vector<int> arr(n);

    for (int i = 0; i < n; ++i) {

        cin >> arr[i];

    }

    // Sorting the array

    sort(arr.begin(), arr.end());

    // Constructing BST from array

    root = constructBSTFromArray(arr, 0, arr.size() - 1);

    // Printing traversals of BST constructed from array

    cout << "Preorder traversal of BST constructed from array: ";

    preorderTraversal(root);

    cout << endl;

    cout << "Inorder traversal of BST constructed from array: ";

    inorderTraversal(root);

    cout << endl;

    cout << "Postorder traversal of BST constructed from array: ";

    postorderTraversal(root);

    cout << endl;

    return 0;

}

#include <iostream>

#include <memory>

#include <queue>

using namespace std;

struct Node {

int data;

shared\_ptr<Node> left;

shared\_ptr<Node> right;

Node(int data) : data(data), left(nullptr), right(nullptr) {

cout << "New node created with data: " << data << endl;

}

};

shared\_ptr<Node> newNode(int data) {

return make\_shared<Node>(data);

}

void insertNode(shared\_ptr<Node>& root, int data) {

if (!root) return;

if (data < root->data) {

if (!root->left) {

root->left = newNode(data);

} else {

insertNode(root->left, data);

}

} else {

if (!root->right) {

root->right = newNode(data);

} else {

insertNode(root->right, data);

}

}

}

shared\_ptr<Node> buildBinaryTree() {

shared\_ptr<Node> root = nullptr;

int numNodes, data;

cout << "Enter the number of nodes you want in the binary tree: ";

cin >> numNodes;

if (numNodes <= 0) {

cout << "Invalid number of nodes. Exiting." << endl;

exit(1);

}

cout << "Enter the value for the root: ";

cin >> data;

root = newNode(data);

for (int i = 1; i < numNodes; ++i) {

cout << "Enter the value for node " << i + 1 << ": ";

cin >> data;

insertNode(root, data);

}

return root;

}

void levelOrder(const shared\_ptr<Node>& root) {

if (!root) return;

queue<shared\_ptr<Node>> q;

q.push(root);

while (!q.empty()) {

shared\_ptr<Node> current = q.front();

q.pop();

cout << current->data << " ";

if (current->left) {

q.push(current->left);

}

if (current->right) {

q.push(current->right);

}

}

}

int main() {

shared\_ptr<Node> root = buildBinaryTree();

cout << "Level-order traversal of the binary tree: " << endl;

levelOrder(root);

return 0;

}