Bansilal Ramnath Agarwal Charitable Trust's

Vishwakarma Institute of Technology

(An Autonomous Institute affiliated to Savitribai Phule Pune University)

**Assignment -2**

|  |  |
| --- | --- |
| **Subject** | Advance Data Structures |
| **Name** | Parth Petkar |
| **Class** | CS-C |
| **Roll No.** | 76 |

**Aim** Binary Tree: 1. Insertion 2. Depth 3. Mirror

#include<stdio.h>

#include<stdlib.h>

#include <stdbool.h>

#include <limits.h>

typedef struct node

{

    int data;

    struct node \*left  ;

    struct node \*right ;

}node;

typedef struct queue{

    node\*\* arr;

    int size;

    int front, rare;

}queue;

void enqueue(queue\* q, node\* root){

    if(root == NULL){

        return;

    }

    q->arr[q->rare++] = root;

}

int dequeue(queue\*q){

    int a = (q->arr[q->front])->data;

    q->front++;

    return a;

}

node\* SearchNode(node \*root, int parent\_data){

    queue\* q = (queue\*)malloc(sizeof(queue));

    q->arr = (node\*)malloc(sizeof(node)\*20);

    q->size = 20;

    q->front = 0;

    q->rare = 0;

    enqueue(q,root);

    while(q->front != q->rare){

        node\* current = q->arr[q->front];

        if(current->data == parent\_data){

            return current;

        }

        enqueue(q, current->left);

        enqueue(q, current->right);

        q->front++;

    }

    printf("\nInvalid Parent Entered\n");

    return root;

}

node\* create(node\* root, int data){

    node \*x = (node\*) malloc (sizeof(node));

    x->data = data;

    x->left = x->right = NULL;

    root = x;

    return root;

}

node\* insert (node \*root, int parent, int data, char pos) {

        node \*x = (node\*) malloc (sizeof(node));

        x->data = data;

        x->left = x->right = NULL;

    node\* temp = SearchNode(root,parent);

    if(pos == 'L' && !temp->left){

        temp->left = x;

    }

    else if(pos == 'R' && !temp->right){

        temp->right = x;

    }

    else{

        printf("\nKey already exists\n");

    }

    return root;

}

void LevelOrderTraversal(node\* root){

    queue\* q = (queue\*)malloc(sizeof(queue));

    q->arr = (node\*)malloc(sizeof(node)\*20);

    q->size = 20;

    q->front = 0;

    q->rare = 0;

    enqueue(q,root);

    while(q->front != q->rare){

        node\* current = q->arr[q->front];

        enqueue(q, current->left);

        enqueue(q, current->right);

        printf("%d ", dequeue(q));

    }

}

int maxfunc(int a, int b){

    if(a>b)

        return a;

    else

        return b;

}

int maxHeight(node\* root){

    //base case

    if (root == NULL){

        return 0;

    }

    //recursive case

    int leftDepth = maxHeight(root->left);

    int rightDepth = maxHeight(root->right);

    return maxfunc(leftDepth, rightDepth)+1;

}

void MirrorImage(node \*root){

    if (root != NULL)

    {

        node\* temp;

        MirrorImage(root->left);

        MirrorImage(root->right);

        temp = root->left;

        root->left  = root->right;

        root->right = temp;

    }

}

bool isMirror(node\* leftSubtree, node\* rightSubtree) {

    if (leftSubtree == NULL && rightSubtree == NULL)

        return true;

    if (leftSubtree == NULL || rightSubtree == NULL)

        return false;

    return (leftSubtree->data == rightSubtree->data) && isMirror(leftSubtree->left, rightSubtree->right) &&

            isMirror(leftSubtree->right, rightSubtree->left);

}

bool isSymmetric(node\* root) {

    if (root == NULL)

        return true;

    return isMirror(root->left, root->right);

}

bool isBSTUntil(node\* root, int min, int max) {

    if (root == NULL)

        return true;

    if (root->data < min || root->data > max)

        return false;

    return isBSTUntil(root->left, min, root->data - 1) &&

           isBSTUntil(root->right, root->data + 1, max);

}

bool isBST(node\* root) {

    return isBSTUntil(root, INT\_MIN, INT\_MAX);

}

void printLeafNodes(node\* root){

    if(root!=NULL){

        if(!(root->left) && !(root->right)){

            printf("%d ",root->data);

            return ;

        }

        if(root->left)

            printLeafNodes(root->left);

        if(root->right)

            printLeafNodes(root->right);

        return;

    }

}

int main() {

    node \*root = NULL;

    int parent;

    int n;

    char position;

    int choice = 1;

    int data;

    do {

        printf("What do you want to do:\n 1. Create Tree\n 2. Insert node\n 3. LevelOrder Traversal\n 4. Height \n 5. Mirror Image \n");

        printf(" 6. Check Symmetricity \n 7. Check BST \n 8. Print leaf nodes \n");

        scanf("%d", &n);

        switch(n) {

            case 1:

                printf("Enter the data of the root of the tree\n");

                scanf("%d", &data);

                root = create(root, data);

                break;

            case 2:

                printf("Enter the data, it's parent and whether it is left(L) or right(R) child\n");

                scanf("%d", &data);

                fflush(stdin);

                scanf("%d", &parent);

                fflush(stdin);

                scanf("%c", &position);

                root = insert(root, parent, data, position);

                break;

            case 3:

                LevelOrderTraversal(root);

                break;

            case 4:

                printf("The depth of the tree is : %d\n",maxHeight(root));

                break;

            case 5:

                printf("The Mirror Image of the Tree is \n");

                MirrorImage(root);

                LevelOrderTraversal(root);

                break;

            case 6:

                if(isSymmetric(root)){

                    printf("\nTHE TREE IS SYMMETRIC\n");

                }

                else{

                    printf("\nTHE TREE IS NOT SYMMETRIC\n");

                }

                break;

              case 7:

                if(isBST(root))

                    printf("\n The given tree is a BST\n");

                else

                    printf("\n The given tree is not a BST\n");

                break;

            case 8:

                printf("\nThe Leaf Nodes are : ");

                printLeafNodes(root);

                printf("\n");

                break;

        }

        printf("\nDo you want to continue? (1 for Yes, 0 for No): ");

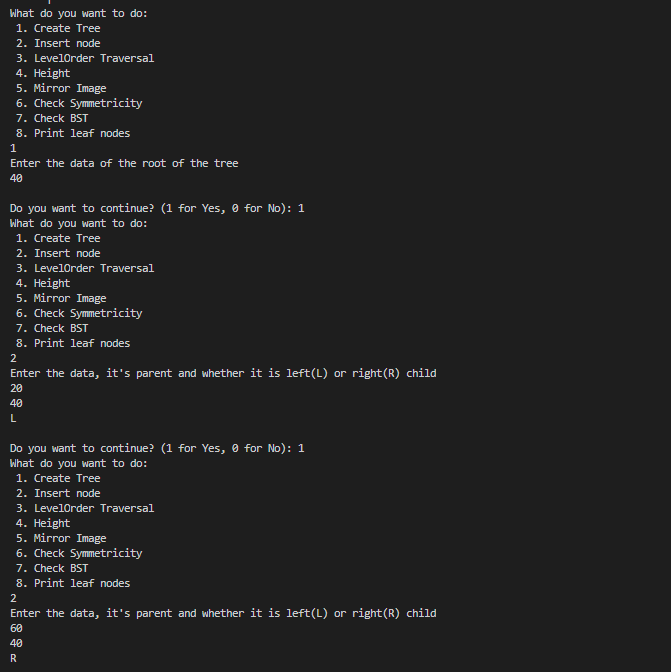
        scanf("%d", &choice);

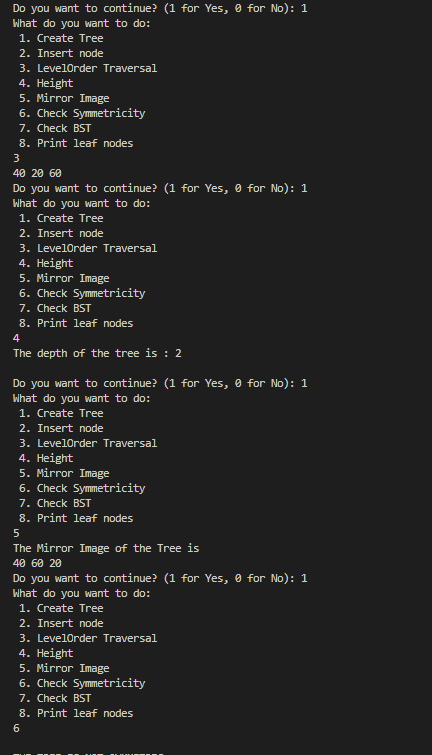
    } while(choice);

    return 0;

}

Output:





A screenshot of a computer program

Description automatically generated