Lab Assignment-11

ROLL: 2005535 | NAME: SAHIL SINGH | DATE: 02/11/21

QUES 1: [1] Write a menu driven program to perform the following operations on a

Binary Search Tree (BST).

* Insert a node (process of creation)
* Check whether the tree constructed is an BST or not.
* Traverse the tree in Preorder
* Traverse the tree in Preorder (Non-recursive)
* Traverse the tree in Post-order
* Traverse the tree in In-order
* Traverse the tree in Level order
* Search whether a given node is present/not.
* Delete a node with degree-0, 1 & 2.

SOLUTION:

#include <stdio.h>

#include <stdlib.h>

#include "stack.h"

#include "queue.h"

typedef *struct* Node

{

*int* data;

*struct* Node \*right;

*struct* Node \*left;

} Node;

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

{

    Node \*temp = (Node \*)malloc(sizeof(Node));

    temp->data = *val*;

    temp->left = NULL;

    temp->right = NULL;

    if (!\**root*)

    {

        \**root* = temp;

        return;

    }

    Node \*ptrR = \**root*;

    Node \*ptr\_prev;

    while (ptrR)

    {

        ptr\_prev = ptrR;

        if (ptrR->data >= *val*)

            ptrR = ptrR->left;

        else

            ptrR = ptrR->right;

    }

    if (ptr\_prev->data > *val*)

        ptr\_prev->left = temp;

    else

        ptr\_prev->right = temp;

}

*int* isBST(Node \**root*)

{

    if (!*root* || (!*root*->left && !*root*->right))

        return 1;

    if (*root*->left && *root*->left->data > *root*->data)

        return 0;

    if (*root*->right && *root*->right->data < *root*->data)

        return 0;

    if (!isBST(*root*->right) || !isBST(*root*->left))

        return 0;

    return 1;

}

*int* search(Node \**root*, *int* *val*)

{

    if (!*root*)

        return 0;

    Node \*temp = *root*;

    while (temp && temp->data != *val*)

    {

        if (temp->data >= *val*)

            temp = temp->left;

        else

            temp = temp->right;

    }

    if (!temp)

        return 0;

    return 1;

}

*void* preorder\_recur(Node \**root*)

{

    if (!*root*)

        return;

    printf("%d->", *root*->data);

    preorder\_recur(*root*->left);

    preorder\_recur(*root*->right);

}

*void* inorder(Node \**root*)

{

    if (!*root*)

        return;

    inorder(*root*->left);

    printf("%d->", *root*->data);

    inorder(*root*->right);

}

*void* postorder(Node \**root*)

{

    if (!*root*)

        return;

    postorder(*root*->left);

    postorder(*root*->right);

    printf("%d->", *root*->data);

}

*void* preorder\_itr(Node \**root*)

{

    if (!*root*)

        return;

    Stack \*stack = NULL;

    push(&stack, *root*);

    while (!isEmpty\_stack(stack))

    {

*root* = pop(&stack);

        printf("%d->", *root*->data);

        if (*root*->right)

            push(&stack, *root*->right);

        if (*root*->left)

            push(&stack, *root*->left);

    }

}

*void* levelorder(Node \**root*)

{

    if (!*root*)

        return;

    Queue queue = {NULL, NULL};

    enqueue(&queue, *root*);

    while (!isEmpty(&queue))

    {

        printf("%d ", peek(&queue)->data);

        if (peek(&queue)->left)

            enqueue(&queue, peek(&queue)->left);

        if (peek(&queue)->right)

            enqueue(&queue, peek(&queue)->right);

        dequeue(&queue);

    }

    printf("\n");

}

*void* deleteNode(Node \*\**root*, *int* *val*)

{

    if (!\**root*)

        return;

    Node \*prev = NULL;

    Node \*ptr = \**root*;

    while (ptr)

    {

        prev = ptr;

        if (ptr->data > *val*)

            ptr = ptr->left;

        else if (ptr->data < *val*)

            ptr = ptr->right;

        if (ptr && ptr->data == *val*)

            break;

    }

    if (!ptr)

        return;

    if (!ptr->right && !ptr->left) //degree 0

    {

        if (\**root* == ptr)

            \**root* = NULL;

        else if (prev->left == ptr)

            prev->left = NULL;

        else

            prev->right = NULL;

        free(ptr);

    }

    else if (ptr->right) //degree 2 or degree 1 (right populated)

    {

        Node \*temp = ptr;

        Node \*curr = ptr->right;

        while (curr->left)

        {

            temp = curr;

            curr = curr->left;

        }

        ptr->data = curr->data;

        if (temp->right == curr)

            temp->right = curr->left;

        else

            temp->left = curr->left;

        free(curr);

    }

    else if (ptr->left) //degree 1 (left populated)

    {

        Node \*temp = ptr;

        Node \*curr = ptr->left;

        while (curr->right)

        {

            temp = curr;

            curr = curr->right;

        }

        ptr->data = curr->data;

        if (temp->right == curr)

            temp->right = curr->left;

        else

            temp->left = curr->left;

        free(curr);

    }

}

*void* replace(Node \**root*, *int* *key*, *int* *val*)

{

    if (!*root*)

        return;

    Node \*ptr = *root*;

    while (ptr)

    {

        if (ptr->data > *key*)

            ptr = ptr->left;

        else if (ptr->data < *key*)

            ptr = ptr->right;

        if (ptr && ptr->data == *key*)

            break;

    }

    if (ptr)

        ptr->data = *val*;

    else

        printf("Key not found!\n");

}

*int* main()

{

    Node \*root = NULL;

*int* choice, val, t\_hold;

    do

    {

        printf("1) Insert\n2) Preorder\n3) Postorder\n4) Inorder\n5) Level order\n");

        printf("6) Search\n7) Delete\n8) Is BST\n9) Replace any node value\n");

        printf("10) Exit\n->: ");

        scanf("%d", &choice);

        printf("\n");

        switch (choice)

        {

        case 1:

            printf("Enter value: ");

            scanf("%d", &val);

            insert(&root, val);

            break;

        case 2:

            printf("1) Recursive\n2) Iterative\n->: ");

            scanf("%d", &choice);

            switch (choice)

            {

            case 1:

                preorder\_recur(root);

                printf("\b\b \n");

                break;

            case 2:

                preorder\_itr(root);

                printf("\b\b \n");

                break;

            }

            break;

        case 3:

            postorder(root);

            printf("\b\b \n");

            break;

        case 4:

            inorder(root);

            printf("\b\b \n");

            break;

        case 5:

            levelorder(root);

            break;

        case 6:

            printf("Enter value to look for: ");

            scanf("%d", &val);

            if (search(root, val))

                printf("Found\n");

            else

                printf("Not Found\n");

            break;

        case 7:

            printf("Enter value to delete: ");

            scanf("%d", &val);

            deleteNode(&root, val);

            break;

        case 8:

            if (isBST(root))

                printf("True\n");

            else

                printf("False\n");

            break;

        case 9:

            printf("Enter value of a node to replace: ");

            scanf("%d", &t\_hold);

            printf("Enter a value to replace the node with: ");

            scanf("%d", &val);

            replace(root, t\_hold, val);

            break;

        default:

            printf("Exiting...\n");

        }

        printf("-------------------------------------------------\n");

    } while (choice >= 1 && choice <= 9);

    return 0;

}

OUTPUT:

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Level order6) Search

7) Delete8) Is BST

9) Replace any node value

10) Exit

->: 1

Enter value: 110

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Level order

6) Search

7) Delete

8) Is BST

9) Replace any node value

10) Exit

->: 1

Enter value: 20

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Level order

6) Search

7) Delete

8) Is BST

9) Replace any node value

10) Exit

->: 1

Enter value: 67

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Level order

6) Search

7) Delete

8) Is BST

9) Replace any node value

10) Exit

->: 1

Enter value: 44

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Level order

6) Search

7) Delete

8) Is BST

9) Replace any node value

10) Exit

->: 2

1) Recursive

2) Iterative

->: 1

110->20->67->44 >

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Level order

6) Search

7) Delete

8) Is BST

9) Replace any node value

10) Exit

->: 2

1) Recursive

2) Iterative

->: 2

110->20->67->44 >

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Level order

6) Search

7) Delete

8) Is BST

9) Replace any node value

10) Exit

->: 3

44->67->20->110 >

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Level order

6) Search

7) Delete

8) Is BST

9) Replace any node value

10) Exit

->: 4

20->44->67->110 >

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Level order

6) Search

7) Delete

8) Is BST

9) Replace any node value

10) Exit

->: 5

110 20 67 44

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Level order

6) Search

7) Delete

8) Is BST

9) Replace any node value

10) Exit

->: 6

Enter value to look for: 67

Found

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Level order

6) Search

7) Delete

8) Is BST

9) Replace any node value

10) Exit

->: 7

Enter value to delete: 67

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Level order

6) Search

7) Delete

8) Is BST

9) Replace any node value

10) Exit

->: 4

20->44->110 >

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Level order

6) Search

7) Delete

8) Is BST

9) Replace any node value

10) Exit

->: 8

True

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Level order

6) Search

7) Delete

8) Is BST

9) Replace any node value

10) Exit

->: 9

Enter value of a node to replace: 20

Enter a value to replace the node with: 10

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Level order

6) Search

7) Delete

8) Is BST

9) Replace any node value

10) Exit

->: 4

10->44->110 >

-------------------------------------------------

1) Insert

2) Preorder

3) Postorder

4) Inorder

5) Level order

6) Search

7) Delete

8) Is BST

9) Replace any node value

10) Exit

->: 10

Exiting...

-------------------------------------------------