#include <stdio.h> //prac7

#include <stdlib.h>

typedef struct Node {

    int data;

    int height;

    struct Node\* left;

    struct Node\* right;

} Node;

int height(Node\* node) {

    if (node == NULL)

        return 0;

    return node->height;

}

int max(int a, int b) {

    return (a > b) ? a : b;

}

Node\* createNode(int data) {

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

    newNode->data = data;

    newNode->height = 1;

    newNode->left = NULL;

    newNode->right = NULL;

    return newNode;

}

Node\* rotateRight(Node\* y) {

    Node\* x = y->left;

    Node\* T2 = x->right;

    x->right = y;

    y->left = T2;

    y->height = max(height(y->left), height(y->right)) + 1;

    x->height = max(height(x->left), height(x->right)) + 1;

    return x;

}

Node\* rotateLeft(Node\* x) {

    Node\* y = x->right;

    Node\* T2 = y->left;

    y->left = x;

    x->right = T2;

    x->height = max(height(x->left), height(x->right)) + 1;

    y->height = max(height(y->left), height(y->right)) + 1;

    return y;

}

int getBalance(Node\* node) {

    if (node == NULL)

        return 0;

    return height(node->left) - height(node->right);

}

Node\* insertNode(Node\* node, int data) {

    if (node == NULL)

        return createNode(data);

    if (data < node->data)

        node->left = insertNode(node->left, data);

    else if (data > node->data)

        node->right = insertNode(node->right, data);

    else

        return node;

    node->height = max(height(node->left), height(node->right)) + 1;

    int balance = getBalance(node);

    if (balance > 1 && data < node->left->data)

        return rotateRight(node);

    if (balance < -1 && data > node->right->data)

        return rotateLeft(node);

    if (balance > 1 && data > node->left->data) {

        node->left = rotateLeft(node->left);

        return rotateRight(node);

    }

    if (balance < -1 && data < node->right->data) {

        node->right = rotateRight(node->right);

        return rotateLeft(node);

    }

    return node;

}

Node\* findMinNode(Node\* node) {

    Node\* current = node;

    while (current->left != NULL)

        current = current->left;

    return current;

}

Node\* deleteNode(Node\* root, int data) {

    if (root == NULL)

        return root;

    if (data < root->data)

        root->left = deleteNode(root->left, data);

    else if (data > root->data)

        root->right = deleteNode(root->right, data);

    else {

        if (root->left == NULL || root->right == NULL) {

            Node\* temp = root->left ? root->left : root->right;

            if (temp == NULL) {

                temp = root;

                root = NULL;

            } else

                \*root = \*temp;

        } else {

            Node\* temp = findMinNode(root->right);

            root->data = temp->data;

            root->right = deleteNode(root->right, temp->data);

        }

    }

    if (root == NULL)

        return root;

    root->height = max(height(root->left), height(root->right)) + 1;

    int balance = getBalance(root);

    if (balance > 1 && getBalance(root->left) >= 0)

        return rotateRight(root);

    if (balance > 1 && getBalance(root->left) < 0) {

        root->left = rotateLeft(root->left);

        return rotateRight(root);

    }

    if (balance < -1 && getBalance(root->right) <= 0)

        return rotateLeft(root);

    if (balance < -1 && getBalance(root->right) > 0) {

        root->right = rotateRight(root->right);

        return rotateLeft(root);

    }

    return root;

}

void inorderTraversal(Node\* root) {

    if (root == NULL)

        return;

    inorderTraversal(root->left);

    printf("%d ", root->data);

    inorderTraversal(root->right);

}

void freeTree(Node\* root) {

    if (root == NULL)

        return;

    freeTree(root->left);

    freeTree(root->right);

    free(root);

}

int main() {

    Node\* root = NULL;

    root = insertNode(root, 10);

    root = insertNode(root, 20);

    root = insertNode(root, 30);

    root = insertNode(root, 40);

    root = insertNode(root, 50);

    root = insertNode(root, 25);

    printf("Inorder traversal after insertion: ");

    inorderTraversal(root);

    printf("\n");

    root = deleteNode(root, 30);

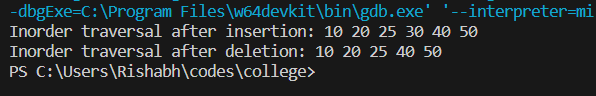
    printf("Inorder traversal after deletion: ");

    inorderTraversal(root);

    printf("\n");

    return 0;

}



#include <stdio.h>// prac8

#include <string.h>

void search(char\* pat, char\* txt){

    int M = strlen(pat);

    int N = strlen(txt);

    for (int i = 0; i <= N - M; i++) {

        int j;

        for (j = 0; j < M; j++)

            if (txt[i + j] != pat[j])

                break;

        if (j== M)

            printf("Pattern found at index %d \n", i);

    }

}

int main(){

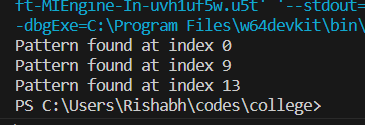
    char txt[] = "AABAACAADAABAAABAA";

    char pat[] = "AABA";

    search(pat, txt);

    return 0;

}



#include <stdio.h> //prac6

#include <stdlib.h>

struct node{

    struct node \*left;

    int info;

    struct node \*right;

};

struct node \*insert(struct node \*root, int ikey){

    struct node \*tmp, \*par, \*ptr;

    int found = 0;

    ptr = root;

    par = NULL;

    while (ptr != NULL){

        if (ikey == ptr->info){

            found = 1;

            break;

        }

        par = ptr;

        if (ikey < ptr->info){

            ptr = ptr->left;

        }else{

            ptr = ptr->right;

        }

    }

    if (found)

        printf("\nDuplicate key");

    else{

        tmp = (struct node \*)malloc(sizeof(struct node));

        tmp->info = ikey;

        tmp->left = NULL;

        tmp->right = NULL;

        if (par == NULL){

            root = tmp;

        }

        else if (ikey < par->info){

            par->left = tmp;

        }

        else{

            par->right = tmp;

        }

    }

    return root;

}

struct node \*case\_a(struct node \*root, struct node \*par, struct node \*ptr){

    if (par == NULL)

        root = NULL;

    else if (ptr == par->left){

        par->left = NULL;

    } else {

        par->right = NULL;

    }

    free(ptr);

    return root;

} /\*End of case\_a()\*/

struct node \*case\_b(struct node \*root, struct node \*par, struct node \*ptr){

    struct node \*child, \*s, \*p;

    if (ptr->left != NULL) /\* node to be deleted has left child \*/

        child = ptr->left;

    else /\* node to be deleted has right child \*/

        child = ptr->right;

    if (par == NULL)

        root = child;

    else if (ptr == par->left)

        par->left = child;

    else

        par->right = child;

    s = in\_succ(ptr);

    p = in\_pred(ptr);

    if (ptr->left != NULL) /\* if ptr has left subtree \*/

        p->right = s;

    free(ptr);

    return root;

}

struct node \*case\_c(struct node \*root, struct node \*par, struct node \*ptr){

    struct node \*succ, \*parsucc;

    parsucc = ptr;

    succ = ptr->right;

    while (succ->left != NULL){

        parsucc = succ;

        succ = succ->left;

    }

    ptr->info = succ->info;

    root = case\_b(root, parsucc, succ);

    return root;

}

struct node \*in\_succ(struct node \*ptr){

    if (ptr->right == NULL)

        return NULL;

    ptr = ptr->right;

    while (ptr->left != NULL)

        ptr = ptr->left;

    return ptr;

}

struct node \*in\_pred(struct node \*ptr){

    if (ptr->left == NULL)

        return NULL;

    ptr = ptr->left;

    while (ptr->right != NULL)

        ptr = ptr->right;

    return ptr;

}

void inorder(struct node \*root){

    struct node \*ptr;

    if (root == NULL){

        printf("Tree is empty");

        return;

    }

    ptr = root;

    while (ptr != NULL){

        if (ptr->left == NULL){

            printf("%d ", ptr->info);

            ptr = ptr->right;

        } else {

            struct node \*pred = ptr->left;

            while (pred->right != NULL && pred->right != ptr)

                pred = pred->right;

            if (pred->right == NULL){

                pred->right = ptr;

                ptr = ptr->left;

            } else {

                pred->right = NULL;

                printf("%d ", ptr->info);

                ptr = ptr->right;

            }

        }

    }

}

int main() {

    int choice, num;

    struct node \*root = NULL;

    root = insert(root, 10);

    root = insert(root, 43);

    root = insert(root, 32);

    root = insert(root, 52);

    root = insert(root, 12);

    inorder(root);

    return 0;

}

