DAA ASSIGNMENT-02

Que01.a.

#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node\* left;

    struct Node\* right;

};

struct Node\* createNode(int value) {

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

    newNode->data = value;

    newNode->left = newNode->right = NULL;

    return newNode;

}

int findMax(struct Node\* root) {

    if (root == NULL) {

        printf("BST is empty.\n");

        return -1;

    }

    while (root->right != NULL) {

        root = root->right;

    }

    return root->data;

}

int main() {

    struct Node\* root = createNode(10);

    root->left = createNode(5);

    root->right = createNode(15);

    root->left->left = createNode(3);

    root->left->right = createNode(7);

    int maxElement = findMax(root);

    printf("Maximum element in the BST: %d\n", maxElement);

    return 0;

}

Output:-



Que01.B

#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node\* left;

    struct Node\* right;

};

struct Node\* createNode(int value) {

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

    newNode->data = value;

    newNode->left = newNode->right = NULL;

    return newNode;

}

int findMin(struct Node\* root) {

    if (root == NULL) {

        printf("BST is empty.\n");

        return -1;

    }

    while (root->left != NULL) {

        root = root->left;

    }

    return root->data;

}

int main() {

    struct Node\* root = createNode(10);

    root->left = createNode(5);

    root->right = createNode(15);

    root->left->left = createNode(3);

    root->left->right = createNode(7);

    int minElement = findMin(root);

    printf("Minimum element in the BST: %d\n", minElement);

    return 0;

}

Output:-



QUE02.

for unsorted array

#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node\* left;

    struct Node\* right;

};

struct Node\* createNode(int value) {

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

    newNode->data = value;

    newNode->left = newNode->right = NULL;

    return newNode;

}

struct Node\* insertNode(struct Node\* root, int value) {

    if (root == NULL) {

        return createNode(value);

    }

    if (value < root->data) {

        root->left = insertNode(root->left, value);

    } else if (value > root->data) {

        root->right = insertNode(root->right, value);

    }

    return root;

}

struct Node\* arrayToBST(int arr[], int size) {

    struct Node\* root = NULL;

    for (int i = 0; i < size; ++i) {

        root = insertNode(root, arr[i]);

    }

    return root;

}

void inorderTraversal(struct Node\* root) {

    if (root == NULL) return;

    inorderTraversal(root->left);

    printf("%d ", root->data);

    inorderTraversal(root->right);

}

int main() {

    int unsortedArray[] = {7, 3, 9, 1, 5, 8, 10};

    int size = sizeof(unsortedArray) / sizeof(unsortedArray[0]);

    struct Node\* root = arrayToBST(unsortedArray, size);

    printf("Inorder Traversal of the BST: ");

    inorderTraversal(root);

    printf("\n");

    return 0;

}

Output:-



For sorted array

#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node\* left;

    struct Node\* right;

};

struct Node\* createNode(int value) {

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

    newNode->data = value;

    newNode->left = newNode->right = NULL;

    return newNode;

}

struct Node\* sortedArrayToBST(int arr[], int start, int end) {

    if (start > end) {

        return NULL;

    }

    int mid = (start + end) / 2;

    struct Node\* root = createNode(arr[mid]);

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

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

    return root;

}

void inorderTraversal(struct Node\* root) {

    if (root == NULL) return;

    inorderTraversal(root->left);

    printf("%d ", root->data);

    inorderTraversal(root->right);

}

int main() {

    int sortedArray[] = {1, 3, 5, 7, 9, 10};

    int size = sizeof(sortedArray) / sizeof(sortedArray[0]);

    struct Node\* root = sortedArrayToBST(sortedArray, 0, size - 1);

    printf("Inorder Traversal of the BST: ");

    inorderTraversal(root);

    printf("\n");

    return 0;

}

Output:-



QUE03.

#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node\* left;

    struct Node\* right;

};

struct Node\* createNode(int value) {

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

    newNode->data = value;

    newNode->left = newNode->right = NULL;

    return newNode;

}

struct Node\* insertNode(struct Node\* root, int value) {

    if (root == NULL) {

        return createNode(value);

    }

    if (value < root->data) {

        root->left = insertNode(root->left, value);

    } else if (value > root->data) {

        root->right = insertNode(root->right, value);

    }

    return root;

}

void printRange(struct Node\* root, int k1, int k2) {

    if (root == NULL) {

        return;

    }

    if (k1 <= root->data) {

        printRange(root->left, k1, k2);

    }

    if (k1 <= root->data && root->data <= k2) {

        printf("%d ", root->data);

    }

    if (root->data <= k2) {

        printRange(root->right, k1, k2);

    }

}

int main() {

    struct Node\* root = NULL;

    root = insertNode(root, 10);

    insertNode(root, 5);

    insertNode(root, 15);

    insertNode(root, 3);

    insertNode(root, 7);

    insertNode(root, 12);

    insertNode(root, 18);

    int k1 = 5;

    int k2 = 15;

    printf("BST nodes in the range [%d, %d]: ", k1, k2);

    printRange(root, k1, k2);

    printf("\n");

    return 0;

}

Output:-



QUE04.

#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node\* left;

    struct Node\* right;

};

struct Node\* createNode(int value) {

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

    newNode->data = value;

    newNode->left = newNode->right = NULL;

    return newNode;

}

struct Node\* insertNode(struct Node\* root, int value) {

    if (root == NULL) {

        return createNode(value);

    }

    if (value < root->data) {

        root->left = insertNode(root->left, value);

    } else if (value > root->data) {

        root->right = insertNode(root->right, value);

    }

    return root;

}

void inOrderTraversal(struct Node\* root) {

    if (root == NULL) return;

    inOrderTraversal(root->left);

    printf("%d ", root->data);

    inOrderTraversal(root->right);

}

void removeDuplicates(int arr[], int size) {

    struct Node\* root = NULL;

    for (int i = 0; i < size; ++i) {

        root = insertNode(root, arr[i]);

    }

    int index = 0;

    rebuildArray(root, arr, &index);

}

void rebuildArray(struct Node\* root, int arr[], int\* index) {

    if (root == NULL) return;

    rebuildArray(root->left, arr, index);

    if (\*index > 0 && arr[\*index - 1] == root->data) {

    } else {

        arr[\*index] = root->data;

        (\*index)++;

    }

    rebuildArray(root->right, arr, index);

}

int main() {

    int arr[] = {4, 2, 6, 3, 2, 5, 4, 7};

    int size = sizeof(arr) / sizeof(arr[0]);

    printf("Original Array: ");

    for (int i = 0; i < size; ++i) {

        printf("%d ", arr[i]);

    }

    printf("\n");

    removeDuplicates(arr, size);

    printf("Array after removing duplicates: ");

    for (int i = 0; i < size; ++i) {

        if (arr[i] != 0) {

            printf("%d ", arr[i]);

        }

    }

    printf("\n");

    return 0;

}

OUTPUT:-

