CHANDANA

03-5-2024

**BINARY\_SEARCH\_TREE**

Binary Search Tree (BST) Implementation:

Create a Node structure with data, left, and right pointers.

Implement functions for:

Insertion: Recursively add nodes while maintaining the BST property (left subtree < node < right subtree).

Deletion: Handle different cases (leaf node, single child, two children).

Search: Recursively search for a specific value, returning the node or NULL if not found.

In-order Traversal: Print nodes in ascending order (left subtree, node, right subtree).

2. Pre-order and Post-order Traversal of a Binary Tree:

Modify the in-order traversal function from question 1 to perform pre-order (root, left, right) and post-order (left, right, root) traversals.

Check if a Binary Tree is a Binary Search Tree (BST):

Implement a recursive function that checks if the left subtree is less than the current node and the right subtree is greater than the current node.

Traverse the tree and return false if any violation is found, otherwise return true.

Find the Height of a Binary Tree:

Implement a recursive function that returns the maximum depth (height) from the root to a leaf node.

Find the Minimum and Maximum Values in a Binary Tree:

Modify the in-order traversal function to track the minimum and maximum values encountered so far.

**IMPLEMENTATION:**

**Bst.h**

#ifndef BST\_H // Header guard to prevent multiple inclusion of the header

#define BST\_H

#include <stdbool.h> // Include the standard library for boolean data type

// Node structure for BST

typedef struct Node {

int data; // Data stored in the node

struct Node\* left; // Pointer to the left child node

struct Node\* right; // Pointer to the right child node

} Node; // Definition of the Node structure

// Function declarations

Node\* createNode(int data); // Function to create a new node

Node\* insert(Node\* root, int data); // Function to insert a node in the BST

Node\* deleteNode(Node\* root, int data); // Function to delete a node from the BST

Node\* search(Node\* root, int data); // Function to search for a node in the BST

void inorderTraversal(Node\* root); // Function to perform in-order traversal of the BST

void preorderTraversal(Node\* root); // Function to perform pre-order traversal of the BST

void postorderTraversal(Node\* root); // Function to perform post-order traversal of the BST

bool isBST(Node\* root); // Function to check if the binary tree is a BST

int maxHeight(Node\* root); // Function to find the maximum height of the binary tree

void findMinMax(Node\* root, int\* min, int\* max); // Function to find the minimum and maximum values in the binary tree

#endif /\* BST\_H \*/ // End of header guard directive

**EXPLANATION :**

1.createNode(int data): Creates a new node with the given data.

insert(Node\* root, int data): Inserts a new node with the given data into the BST rooted at root.

2.deleteNode(Node\* root, int data): Deletes the node with the given data from the BST rooted at root.

search(Node\* root, int data): Searches for a node with the given data in the BST rooted at root.

3.inorderTraversal(Node\* root): Performs an in-order traversal of the BST rooted at root.

4.preorderTraversal(Node\* root): Performs a pre-order traversal of the BST rooted at root.

5.postorderTraversal(Node\* root): Performs a post-order traversal of the BST rooted at root.

isBST(Node\* root): Checks if the binary tree rooted at root is a valid BST.

6.maxHeight(Node\* root): Finds the maximum height of the binary tree rooted at root.

7.findMinMax(Node\* root, int\* min, int\* max): Finds the minimum and maximum values in the binary tree rooted at root.

The header file is enclosed in an include guard to ensure that it is only included once in a compilation unit. This prevents issues with duplicate declarations if the header is included multiple times.

**Bst.c**

#include "bst.h" // Include the header file defining the structures and function prototypes

#include <stdio.h> // Include standard I/O functions

#include <stdlib.h> // Include standard library functions

#include <limits.h> // Include library for INT\_MIN and INT\_MAX

// Function to create a new node with the given data

Node\* createNode(int data) {

Node\* newNode = (Node\*)malloc(sizeof(Node)); // Allocate memory for the new node

newNode->data = data; // Assign the data to the new node

newNode->left = newNode->right = NULL; // Set left and right child pointers to NULL

return newNode; // Return the newly created node

}

// Function to insert a node with the given data into the BST

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

if (root == NULL) // If the root is NULL, create a new node and return it

return createNode(data);

if (data < root->data) // If data is less than the root's data, insert into the left subtree

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

else if (data > root->data) // If data is greater than the root's data, insert into the right subtree

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

return root; // Return the root of the modified BST

}

// Function to delete a node with the given data from the BST

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

if (root == NULL) // If the root is NULL, return NULL

return root;

if (data < root->data) // If data is less than the root's data, delete from the left subtree

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

else if (data > root->data) // If data is greater than the root's data, delete from the right subtree

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

else { // If the node to be deleted is found

if (root->left == NULL) { // If the node has no left child

Node\* temp = root->right; // Store the right child temporarily

free(root); // Free the memory occupied by the node

return temp; // Return the right child to connect with the parent

} else if (root->right == NULL) { // If the node has no right child

Node\* temp = root->left; // Store the left child temporarily

free(root); // Free the memory occupied by the node

return temp; // Return the left child to connect with the parent

}

// If the node has both left and right children

Node\* temp = root->right; // Find the minimum node in the right subtree

while (temp->left != NULL)

temp = temp->left;

root->data = temp->data; // Copy the data of the minimum node to the current node

root->right = deleteNode(root->right, temp->data); // Delete the minimum node from the right subtree

}

return root; // Return the root of the modified BST

}

// Function to search for a node with the given data in the BST

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

if (root == NULL || root->data == data) // If root is NULL or data is found at the root, return root

return root;

if (data < root->data) // If data is less than the root's data, search in the left subtree

return search(root->left, data);

return search(root->right, data); // If data is greater than the root's data, search in the right subtree

}

// Function to perform in-order traversal of the BST

void inorderTraversal(Node\* root) {

if (root != NULL) { // If the root is not NULL

inorderTraversal(root->left); // Recursively traverse the left subtree

printf("%d ", root->data); // Print the data of the current node

inorderTraversal(root->right); // Recursively traverse the right subtree

}

}

// Function to perform pre-order traversal of the BST

void preorderTraversal(Node\* root) {

if (root != NULL) { // If the root is not NULL

printf("%d ", root->data); // Print the data of the current node

preorderTraversal(root->left); // Recursively traverse the left subtree

preorderTraversal(root->right); // Recursively traverse the right subtree

}

}

// Function to perform post-order traversal of the BST

void postorderTraversal(Node\* root) {

if (root != NULL) { // If the root is not NULL

postorderTraversal(root->left); // Recursively traverse the left subtree

postorderTraversal(root->right); // Recursively traverse the right subtree

printf("%d ", root->data); // Print the data of the current node

}

}

// Utility function to check if the binary tree rooted at the given node is a BST

bool isBSTUtil(Node\* root, int min, int max) {

if (root == NULL) // If the root is NULL, it's a BST

return true;

if (root->data < min || root->data > max) // If the data of the root is outside the valid range, not a BST

return false;

// Check recursively for left and right subtrees

return isBSTUtil(root->left, min, root->data - 1) && isBSTUtil(root->right, root->data + 1, max);

}

// Function to check if the binary tree is a BST

bool isBST(Node\* root) {

return isBSTUtil(root, INT\_MIN, INT\_MAX); // Call utility function with initial range as INT\_MIN and INT\_MAX

}

// Function to find the maximum height of the binary tree

int maxHeight(Node\* root) {

if (root == NULL) // If the root is NULL, height is 0

return 0;

// Calculate heights of left and right subtrees recursively

int leftHeight = maxHeight(root->left);

int rightHeight = maxHeight(root->right);

// Return the maximum of the heights plus 1 (for the current node)

return (leftHeight > rightHeight) ? leftHeight + 1 : rightHeight + 1;

}

// Function to find the minimum and maximum values in the binary tree

void findMinMax(Node\* root, int\* min, int\* max) {

if (root == NULL) // If the root is NULL, return

return;

findMinMax(root->left, min, max); // Recursively find min/max in the left subtree

if (root->data < \*min) // Update min if the data of the current node is smaller

\*min = root->data;

if (root->data > \*max) // Update max if the data of the current node is larger

\*max = root->data;

findMinMax(root->right, min, max); // Recursively find min/max in the right subtree

}

**EXPLANATION :**

1.createNode(int data): This function creates a new node with the given data and returns a pointer to the new node.

2.insert(Node\* root, int data): This function inserts a new node with the given data into the BST rooted at root. It recursively finds the correct position to insert the new node based on the BST properties.

3.deleteNode(Node\* root, int data): This function deletes a node with the given data from the BST rooted at root. It handles three cases: deleting a node with no children, deleting a node with one child, and deleting a node with two children.

4.search(Node\* root, int data): This function searches for a node with the given data in the BST rooted at root. It returns a pointer to the node if found, or NULL if not found.

5.inorderTraversal(Node\* root): This function performs an in-order traversal of the BST rooted at root, which visits nodes in ascending order.

6.preorderTraversal(Node\* root): This function performs a pre-order traversal of the BST rooted at root, which visits the root node first, then the left subtree, and finally the right subtree.

7.postorderTraversal(Node\* root): This function performs a post-order traversal of the BST rooted at root, which visits the left subtree, then the right subtree, and finally the root node.

1. isBSTUtil(Node\* root, int min, int max): This is a utility function used by isBST(Node\* root) to check if the binary tree rooted at root is a valid
2. BST. It recursively checks each node's value against a range (min, max) to ensure it is within the correct bounds.

10.isBST(Node\* root): This function checks if the binary tree rooted at root is a valid BST by calling the isBSTUtil function with initial range as INT\_MIN and INT\_MAX.

11.maxHeight(Node\* root): This function finds the maximum height of the binary tree rooted at root by recursively calculating the height of the left and right subtrees and returning the maximum of the two heights plus one for the current node.

12.findMinMax(Node\* root, int\* min, int\* max): This function finds the minimum and maximum values in the binary tree rooted at root by recursively traversing the tree and updating the min and max values accordingly.

**Main.c**

#include <stdio.h> // Include standard I/O functions

#include <stdlib.h> // Include standard library functions

#include <time.h> // Include time functions

#include <limits.h> // Include library for INT\_MIN and INT\_MAX

#include "bst.h" // Include the header file defining the structures and function prototypes

int main() {

Node\* root = NULL; // Declare a pointer to the root node and initialize it to NULL

root = insert(root, 50); // Insert nodes into the BST

insert(root, 30);

insert(root, 20);

insert(root, 40);

insert(root, 70);

insert(root, 60);

insert(root, 80);

printf("Inorder traversal: "); // Print the result of inorder traversal

inorderTraversal(root);

printf("\n");

printf("Preorder traversal: "); // Print the result of preorder traversal

preorderTraversal(root);

printf("\n");

printf("Postorder traversal: "); // Print the result of postorder traversal

postorderTraversal(root);

printf("\n");

int value = 70; // Value to be searched in the BST

Node\* found = search(root, value); // Search for the value in the BST

if (found)

printf("%d is found in the tree.\n", value); // Print if the value is found

else

printf("%d is not found in the tree.\n", value); // Print if the value is not found

int min = INT\_MAX, max = INT\_MIN; // Initialize variables to store min and max values

findMinMax(root, &min, &max); // Find the minimum and maximum values in the BST

printf("Minimum value: %d\n", min); // Print the minimum value

printf("Maximum value: %d\n", max); // Print the maximum value

printf("Is the binary tree a BST? %s\n", isBST(root) ? "Yes" : "No"); // Check if the binary tree is a BST

printf("Height of the binary tree: %d\n", maxHeight(root)); // Print the height of the binary tree

// CPU time calculation

clock\_t start = clock(); // Start the clock

// Perform some CPU intensive task here

clock\_t end = clock(); // End the clock

double cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC; // Calculate CPU time used

printf("CPU Time: %f seconds\n", cpu\_time\_used); // Print CPU time used

return 0; // Return 0 to indicate successful completion of the program

}

**EXPLANATION :**

1.\*Node root = NULL;\*\*: This declares a pointer root to the root node of the BST and initializes it to NULL.

2.Insert nodes into the BST: Nodes with values 50, 30, 20, 40, 70, 60, and 80 are inserted into the BST using the insert function.

Traversals:

3.Inorder traversal: Prints the values of the BST in sorted order using inorderTraversal.

4.Preorder traversal: Prints the root node first, followed by the left subtree and then the right subtree, using preorderTraversal.

5.Postorder traversal: Prints the left subtree, then the right subtree, and finally the root node, using postorderTraversal.

6.Search for a value: Searches for the value 70 in the BST using the search function and prints whether it is found or not.

7.Find minimum and maximum values: Finds and prints the minimum and maximum values in the BST using the findMinMax function.

8.Check if the binary tree is a BST: Checks if the binary tree is a valid BST using the isBST function and prints the result.

9.Calculate the height of the binary tree: Calculates and prints the height of the binary tree using the maxHeight function.

10.CPU time calculation:

Starts the clock using clock() to measure CPU time before the CPU-intensive task.

Performs some CPU-intensive task (not specified in the code).

Ends the clock using clock() to measure CPU time after the task.

Calculates and prints the CPU time used for the task.

**FLOW CHART FOR THE CODE**

**Start**

**|**

**V**

**Create an empty root node**

**|**

**V**

**Insert nodes into the binary tree (50, 30, 20, 40, 70, 60, 80)**

**|**

**V**

**Print "In-order traversal:"**

**|**

**V**

**Perform in-order traversal**

**|**

**V**

**Print the height of the binary tree**

**|**

**V**

**Find minimum and maximum values in the binary tree**

**|**

**V**

**Print minimum and maximum values**

**|**

**V**

**Check if the binary tree is a BST**

**|**

**V**

**Print whether it's a BST or not**

**|**

**V**

**Delete a node from the binary tree**

**|**

**V**

**Print "In-order traversal after deletion:"**

**|**

**V**

**Perform in-order traversal after deletion**

**|**

**V**

**Free memory allocated for the binary tree**

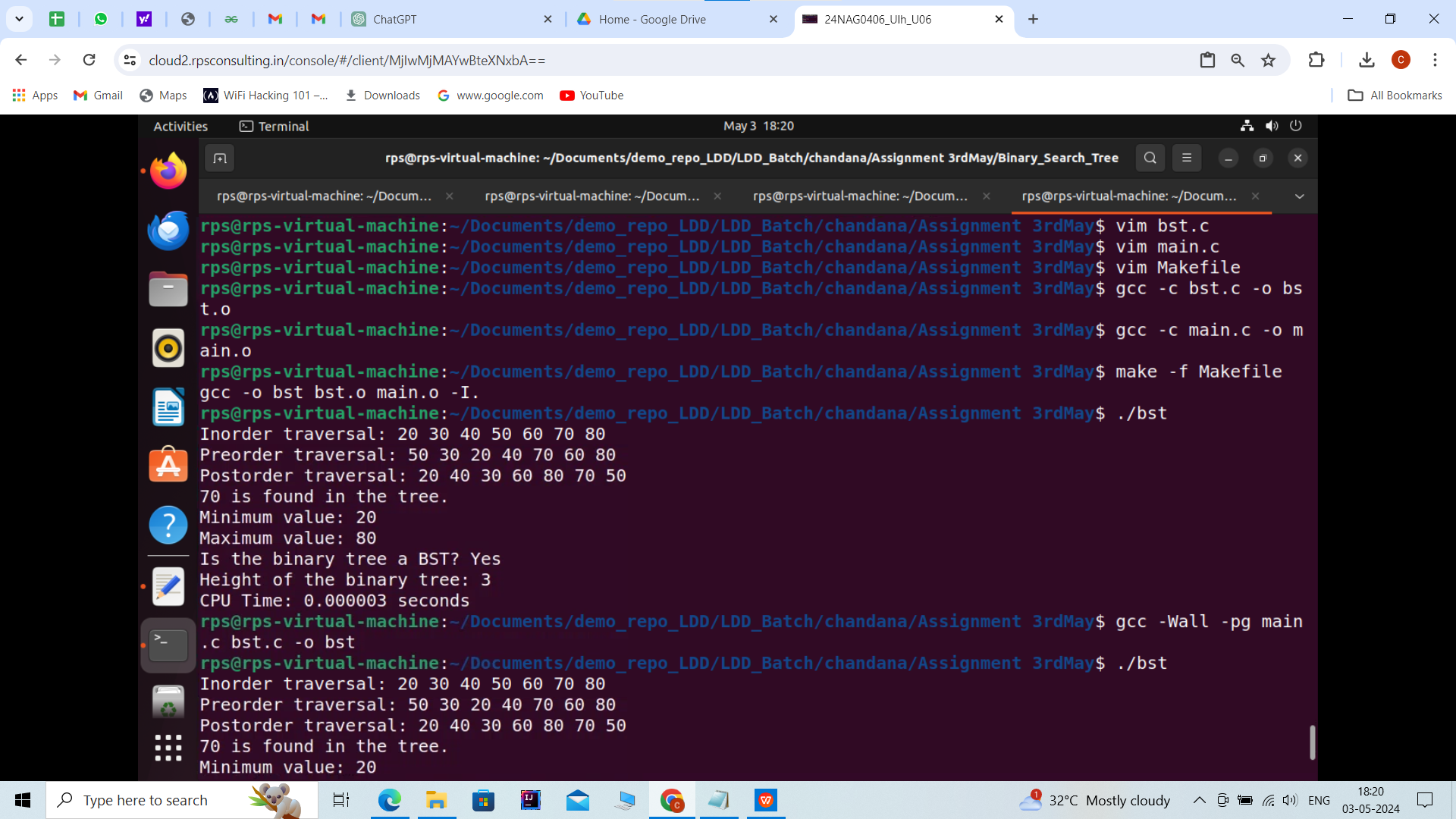
**|**

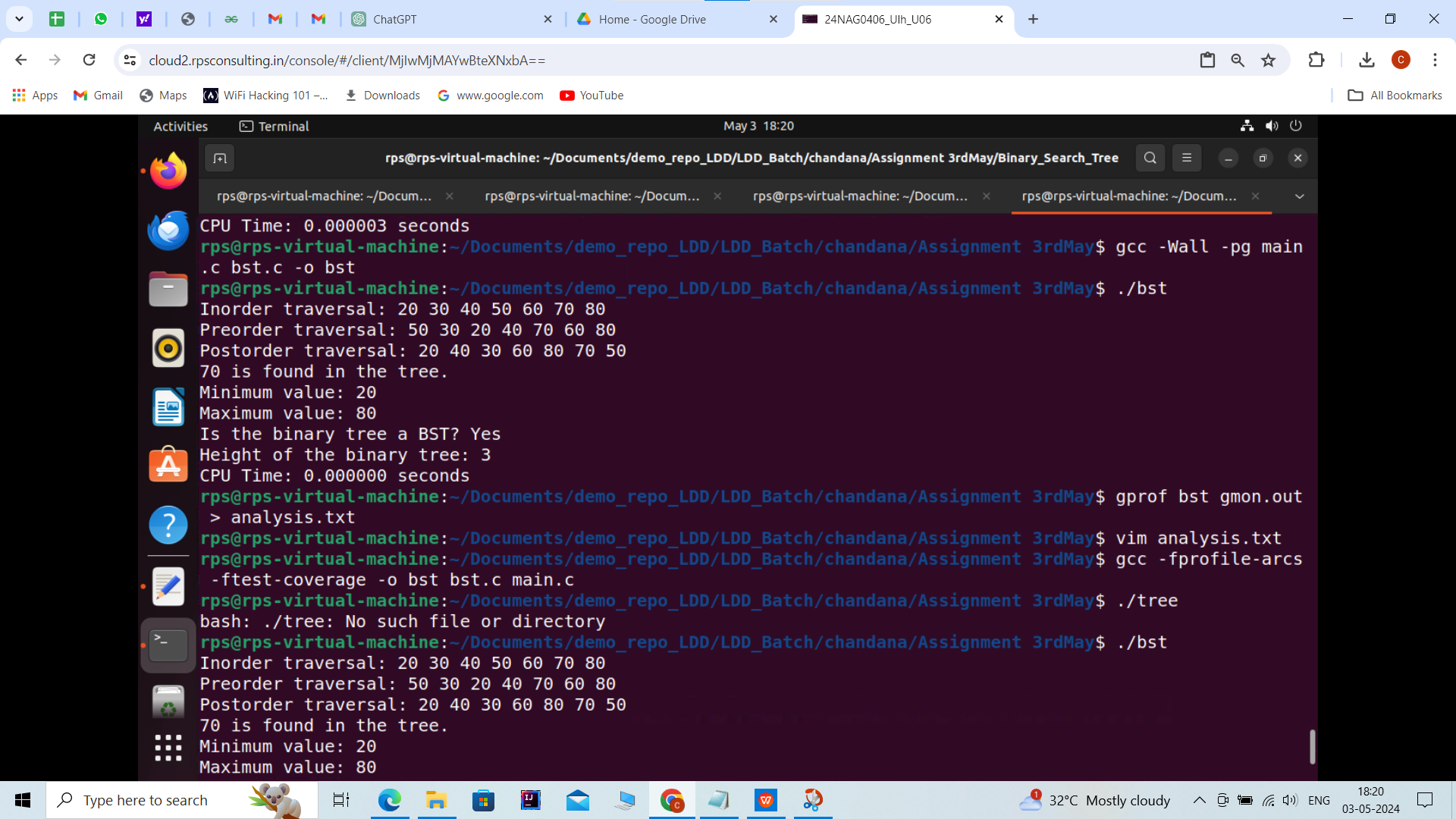
**V**

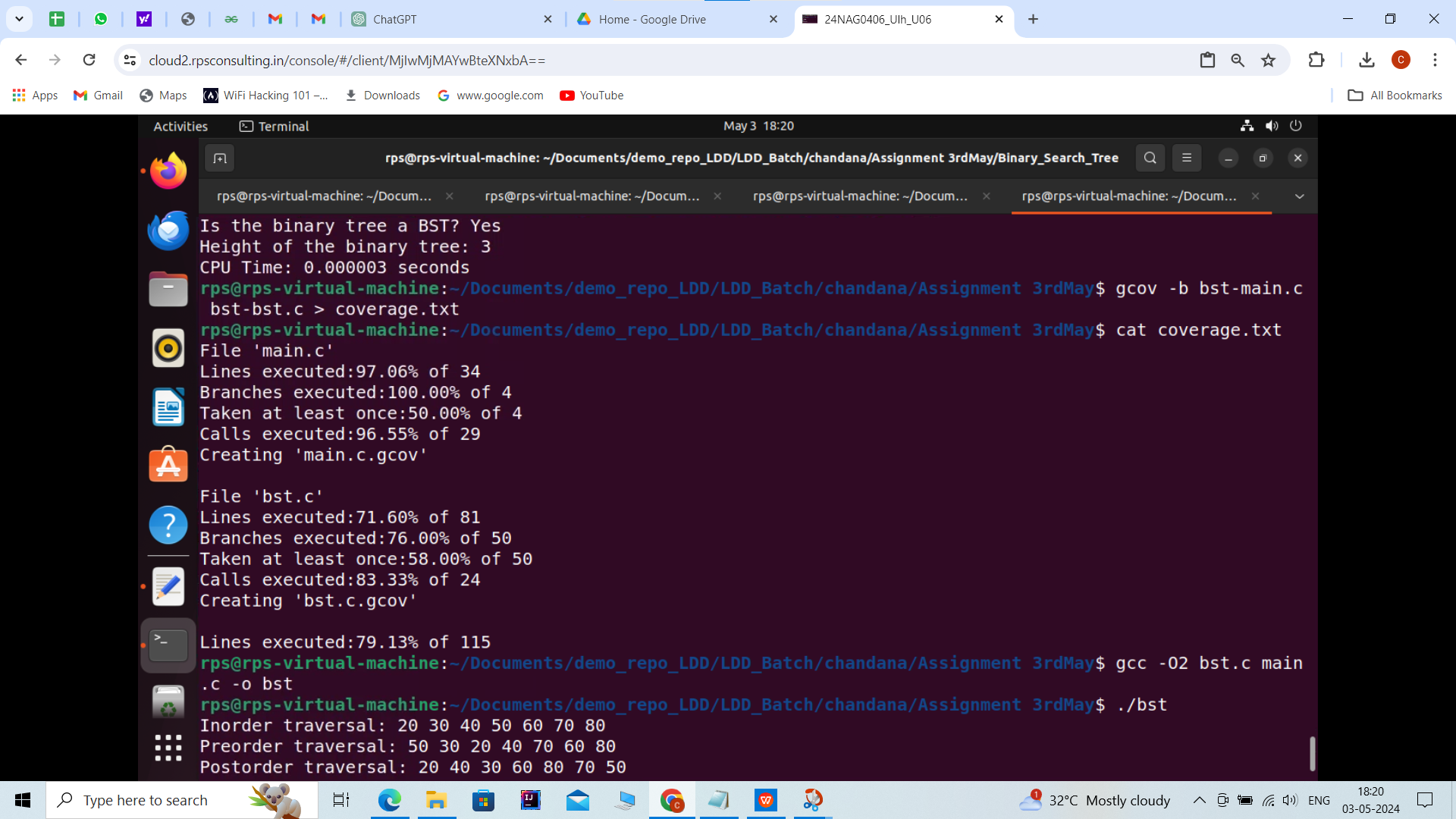
**End**

**OUTPUT:**

**AND COVERAGE TEXT AND ANALYSIS TXT.**







**OPTIMIZATION CODE FOR BINARY\_SEARCH\_TREE**

