Program - 8

FIND THE HEIGHT OF THE BINARY TREE

Algorithm:

1. Define the Node Structure:

- Create a structure for a binary tree node that includes:

    An integer data field to store the node's data.

    Pointers to the left and right children (left and right).

2. Create a Function to Calculate Height:

- Define a recursive function height that takes a binary tree node as input.
- If the input node is NULL, return -1 (the height of an empty tree is -1).
- Otherwise, calculate the height of the left subtree:

    Call the height function recursively on the left child of the current node and store the result in a variable leftHeight.
- Calculate the height of the right subtree:

    Call the height function recursively on the right child of the current node and store the result in a variable rightHeight.

Return the maximum of leftHeight and rightHeight, plus 1 for the current node.

3. Main Program:

- Create a sample binary tree by creating nodes and connecting them with left and right child pointers.
- Call the height function with the root of the tree to calculate the height of the binary tree.
- Print the calculated height.

4. Display the Result:

- Output the height of the binary tree obtained in the previous step.

```
#include <stdio.h>
#include <stdlib.h>
```

These lines include some standard C libraries that we'll use in our program.

```
// Definition of a binary tree node
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};
```

Here, we define a structure called Node that represents a node in a binary tree. Each node has three parts: data, which stores some information, and left and right, which point to the left and right children of the node.

```
// Function to create a new node
struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return node;
}
```

The newNode function is used to create a new node with the given data. It allocates memory for the node, initializes its data, and sets both left and right child pointers to NULL. Then, it returns the newly created node.

```
// Function to calculate the height of a binary tree
int height(struct Node* root) {
    if (root == NULL) {
        return -1; // Height of an empty tree is -1
    } else {
        int leftHeight = height(root->left);
        int rightHeight = height(root->right);

        // Return the maximum of leftHeight and rightHeight, plus 1 for the current node
        return (leftHeight > rightHeight) ? (leftHeight + 1) : (rightHeight + 1);
    }
}
```

The height function calculates the height of a binary tree. It takes the root of the tree as input and returns the height as an integer.

- *If the root is NULL (indicating an empty tree), it returns -1, as the height of an empty tree is defined to be -1.*

- *If the root is not NULL, it calculates the height recursively:*

- *It calculates the height of the left subtree using height(root->left) and stores it in leftHeight.*
- *It calculates the height of the right subtree using height(root->right) and stores it in rightHeight.*
- *It returns the maximum of leftHeight and rightHeight, adding 1 to account for the current node.*

```
int main() {
    // Creating a sample binary tree
    struct Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);

    // Calculate and print the height of the binary tree
    printf("Height of the binary tree is: %d\n", height(root));

    return 0;
}
```

In the main function:

- *We create a sample binary tree by creating nodes and connecting them with left and right child pointers.*
- *We then call the height function with the root of the tree to calculate the height.*
- *Finally, we print the calculated height of the binary tree.*