RAJIV GANDHI NATIONAL INSTITUTE OF YOUTH DEVELOPMENT
(Institution of National Importance by the Act of Parliament No.35/2012)
Ministry of Youth Affairs and Sports, Government Of India

DATA STRUCTURE AND ALGORITHM-PROJECT REPORT

Department of Computer Science
(1st semester, 22-24 batch)

RGNIYD, SRIPERUMBUDUR-602105

Group members-

Arundhati Sensharma	ID-MSAI22R005
Prerana Bora	ID-MSAI22R013
Chintapenta Surya Bharadwaja	ID-MSCS22R003
Junais Musthafa	ID-MSDS22R007
Riga Suresh M	ID-MSDS22R009
Suman Kumari	ID-MSDS22R014
Adithya K P	ID-MSAI22R001
Ritik Sahu	ID-MSCS22R012

Under supervision of **Dr.P.Thiyagarajan**,Associate Professor, HOD of Computer Science (Cyber Security), RGNIYD

February 10, 2023

CERTIFICATE

This is to certify that I have examined the project entitled submitted by **Arundhati Sensharma,Prerana Bora,Chintapenta Surya Bharadwaja,Junais Mustafa,Riga Suresh M,Adithya P K,Suman Kumari,Ritik Sahu** the postgraduate students of Department of Computer Science. I hereby accord my approval of it as a study carried out and it fulfilled the requirements as per the regulations of the institute as well as course instructor and has reached the standard needed for submission.

Dr.P.Thiyagarajan
Associate Professor
HOD of Computer Science (Cyber Security)
RGNIYD,Sriperumbudur,Tamilnadu

ACKNOWLEDGEMENT

We would like to express our sincere and deep gratitude to our supervisor and guide Dr.P.Thiyagarajan, Associate Professor, HOD of Computer Science (Cyber Security), for his kind and constant support during our project work. It has been an absolute privilege to work with Dr.P.Thiyagarajan for our project. His valuable advice, critical criticism and active supervision encouraged us to sharpen our research methodology and was instrumental in shaping our professional outlook.

THANK-YOU

ABSTRACT

The core goals of this project on Binary tree is to understand thoroughly the inherent concepts of basic data structure "Binary Tree" and the properties of binary tree like its different types of nodes like root node, parent node, child node, leaf node, child node and sibling node. Here in this project the high priority is given on the understanding the cousin node and sibling node concepts and constructed the code to check whether the two nodes are cousin nodes or not in any binary tree using with and without recursive approach and as a additional example we provide the code to check the sibling nodes of a binary tree with or without recursive approach. On constructing the code we're trying to follow the coding standards to some extent. To run the code one can use Visual Studio Code or any online compiler

Check whether two nodes of binary tree are cousin or not?

Project Report by group 6

Introduction

Our data structure and algorithm project is to **"Write a C program to construct a binary tree. Write function for all traversal with and without recursion. Given any two nodes find whether they are cousins?"**

The type of data structure used in this system is Binary Tree. The user can give input to the nodes and traverse the nodes in three orders that is Pre-order, Post-order and In-order with and without recursive approach. Then user can give two inputs and check whether these two nodes are cousins or not. In addition to checking cousin nodes we also provide different function to check whether or not two nodes are siblings.

In order to go in brief of this system first we need to give a look on the key concepts we use in this data structure.

TREE: A tree is a non-linear data structure. A tree represents a hierarchical structure. A tree contains nodes and 2 pointers. These two pointers are the left child and the right child of the parent node.

ROOT: The root of a tree is the topmost node of the tree that has no parent node.

PARENT NODE: The node which is a predecessor of a node is called the parent node of that node.

CHILD NODE: The node which is the immediate successor of a node is called the child node of that node.

SIBLING: Children of the same parent node are called siblings.

COUSIN: Two nodes of a binary tree are cousins if they have the same depth with different parents.

EDGE: Edge acts as a link between the parent node and the child node.

LEAF: A node that has no child is known as the leaf node. It is the last node of the tree. There can be multiple leaf nodes in a tree.

SUB-TREE: The sub-tree of a node is the tree considering that particular node as the root node.

DEPTH: The depth of the node is the distance from the root node to that particular node.

HEIGHT: The height of the node is the distance from that node to the deepest node of that sub-tree.

HEIGHT OF THE TREE: The Height of the tree is the maximum height of any node. This is the same as the height of the root node.

LEVEL: A level is the number of parent nodes corresponding to a given node of the tree.

DEGREE OF NODE: The degree of a node is the number of its children.

NULL: The number of NULL nodes in a binary tree is (N+1), where N is the number of nodes in a binary tree.

1 BINARY TREE

: A binary tree is a tree data structure in which each node can have at most two children, which are referred to as the left child and the right child.

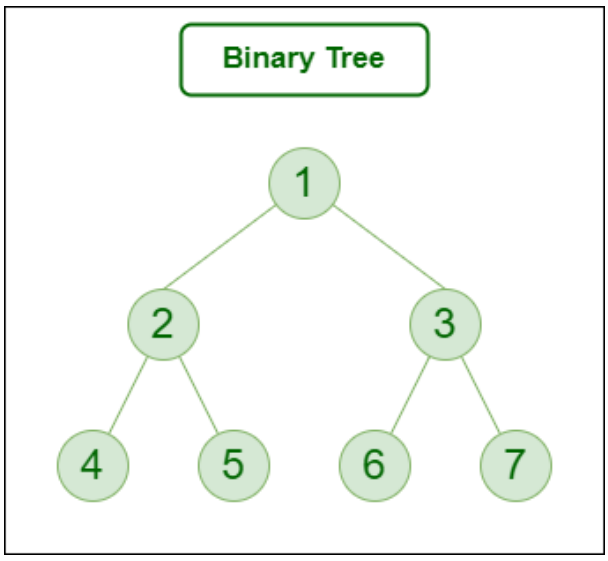

Figure 1: BINARY TREE

GIVEN TWO NODES IN A BINARY TREE, CHECK IF THEY ARE COUSIN:

: Let 'p' and 'q' are the two nodes in binary tree to determine whether the two nodes are cousins of each other or not :-

Two nodes are cousins if,

1.They are not siblings (Children of same parent).
2.They are on the same level.

Two nodes are cousins of each other if they are at same level and have different parents.

Lets see few example for better understanding on when two nodes are said to be cousin of each other in Binary Tree.

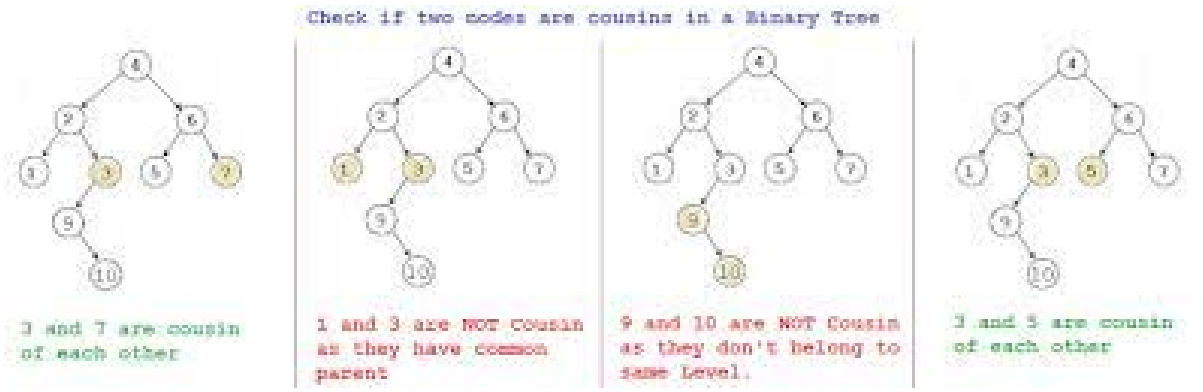

Figure 2: Check if cousins or not

2 Algorithms of the functions used in our systems with explanations-

List of functions:

1. Insert
2. Recursive In-Order Traversal
3. Non-Recursive In-Order Traversal
4. Recursive Pre-Order Traversal
5. Non-Recursive Pre-Order Traversal
6. Recursive Post-Order Traversal
7. Non-Recursive Post-Order Traversal
8. Check the level of a given node
9. Find whether two nodes are Cousins
10. Find whether two nodes are Sibling

As soon as the user run the program these will pop up on the screen:

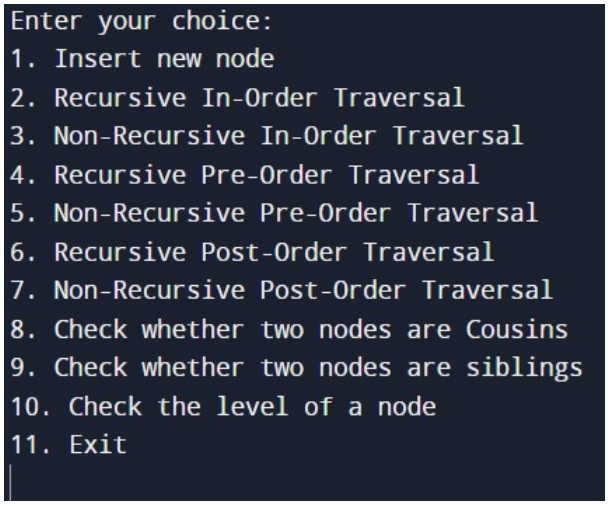

Figure 3: List of all the function

INSERT FUNCTION

Insert function is used to add a new element in a binary search tree at appropriate location. Insert function is to be designed in such a way that, it must not violate the property of binary search tree at each value.
The function insert takes a binary tree node and an integer value as input and returns the root of the modified binary tree. If the tree is empty, it returns a new node with the given data. If the data is less than or equal to the node's data, it inserts the data in the left sub-tree; otherwise, it inserts the data in the right sub-tree.

To insert an element into a binary tree, follow these steps:

1. Start at the root node. If the root node is empty, insert the element as the root node and return.

2.Compare the value of the element to be inserted with the value of the current node. If the value is less than the current node, move to the left child. If the value is greater, move to the right child.

3.If there is no left or right child, insert the element as a new left or right child.

4. Repeat the process, starting at step 2, until the element is inserted.

RECURSIVE IN-ORDER TRAVERSAL:

The function inorder takes a binary tree node as input and performs an inorder traversal of the binary tree. The inorder traversal visits the left subtree, the node itself, and then the right subtree. This is achieved by recursively calling inorder on the left child of the node until it reaches a NULL pointer, then printing the node's data, and finally calling inorder on the right child. This results in the elements of the binary tree being printed in sorted order, assuming the binary tree is a search tree.

Algorithm for recursive in-order traversal of a binary tree:

1.If the current node is not None:
2.Recursively call the in-order traversal function on the left child of the current node.
3.Process the data or print the value of the current node.
4.Recursively call the in-order traversal function on the right child of the current node.

Pseudo-code for recursive in-order traversal

```
procedure inorder(node)
if node is not NULL then
inorder(node.left)
print node.data
inorder(node.right)
end if
end procedure
```

For recursive in-order traversal:

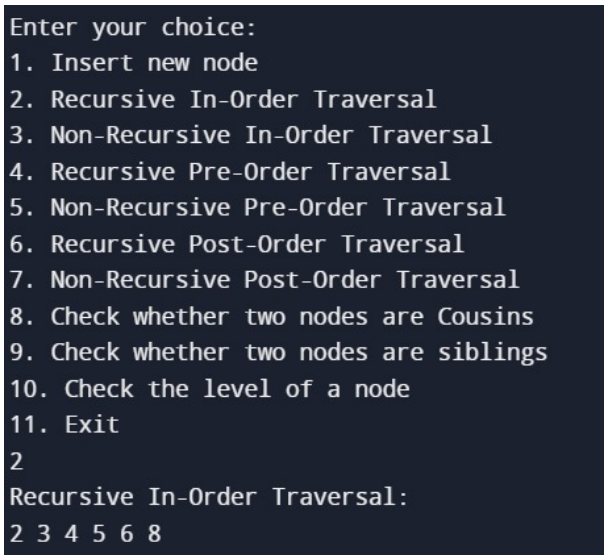

Figure 4: Recursive in-order traversal

NON-RECURSIVE IN-ORDER TRAVERSAL:

A non-recursive in-order traversal of a binary tree is a way to traverse the tree visiting its nodes in the order "left subtree - node - right subtree" using an auxiliary data structure, such as a stack, instead of recursion. The algorithm starts at the root node, pushes left children onto the stack and visits nodes by popping them from the stack, printing their data and pushing their right children onto the stack. The process continues until the stack is empty, resulting in an in-order traversal of the tree.

Algorithm for non recursive in-order traversal of a binary tree:

1.Create an empty stack.
2.Set the current node to the root of the binary tree.
3.Repeat steps 4-7 until the current node is null and the stack is empty.
4.If the current node is not null, push it onto the stack and move to its left child.
5.If the current node is null and the stack is not empty, pop the top node from the stack, set it as the current node, and move to its right child.
6.If the current node is not null and has no left child, print its value.
7.If the current node is null and has no right child, print its value.
8.End the loop.

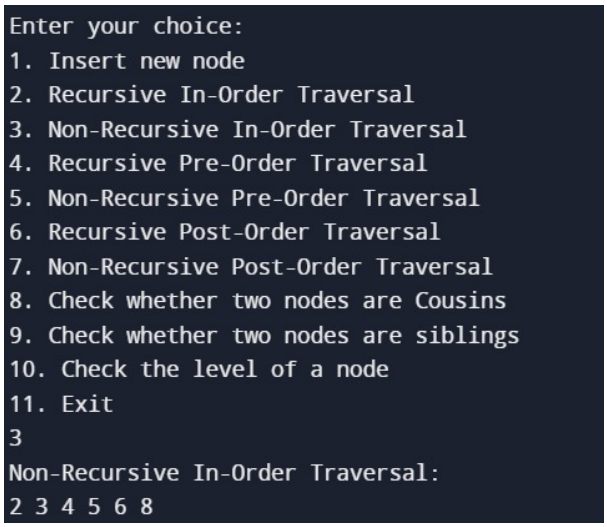

Figure 5: Non-recursive in-order traversal

RECURSIVE PRE-ORDER TRAVERSAL:

A recursive preorder traversal of a binary tree in C language is a way to traverse the tree in a specific order, visiting the root node first, then the left subtree, and finally the right subtree. The function recursively calls itself for the left and right subtrees of the current node.

Algorithm for recursive pre-order traversal of a binary tree:

1.If the current node is null, return.
2.Print the value of the current node.
3.Recursively traverse the left subtree of the current node.
4.Recursively traverse the right subtree of the current node.

For Recursive pre-order traversal

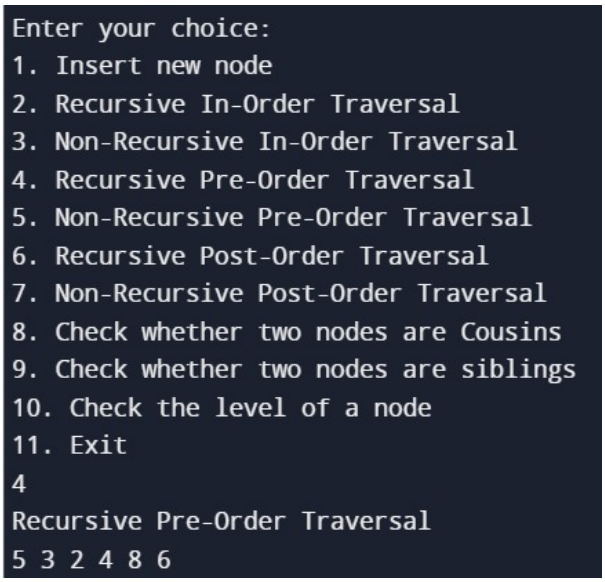

Figure 6: Recursive pre-order traversal

NON-RECURSIVE PRE-ORDER TRAVERSAL:

A non-recursive preorder traversal of a binary tree in C language is a way to traverse the tree without using recursion, visiting the root node first, then the left subtree, and finally the right subtree. Instead of recursion, the function uses a stack to keep track of the nodes to be visited.

Algorithm for non-recursive pre-order traversal of a binary tree:

1.Initialize an empty stack.
2.Push the root of the binary tree onto the stack.
3.While the stack is not empty:
a. Pop the top element from the stack, process its data or print its value.
b. If the popped node has a right child, push it onto the stack.
c. If the popped node has a left child, push it onto the stack.

For non recursive Pre-order traversal:

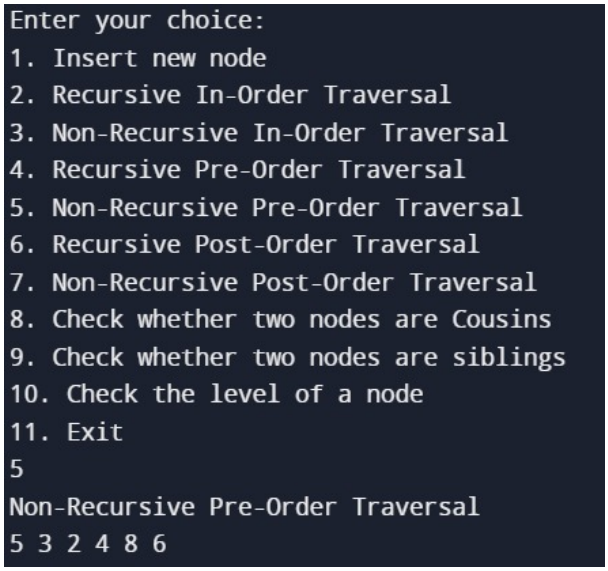

Figure 7: Non-recursive pre-order traversal

RECURSIVE POST-ORDER TRAVERSAL:

A recursive post-order traversal of a binary tree in C language is a way to traverse the tree in a specific order, visiting the left subtree first, then the right subtree, and finally the root node. The function recursively calls itself for the left and right subtrees of the current node.

Algorithm for recursive Post-order traversal of a binary tree:

1.If the current node is not None:
2.Recursively call the post-order traversal function on the left child of the current node.
3.Recursively call the post-order traversal function on the right child of the current node.
4.Process the data or print the value of the current node.

For recursive Post-order traversal:

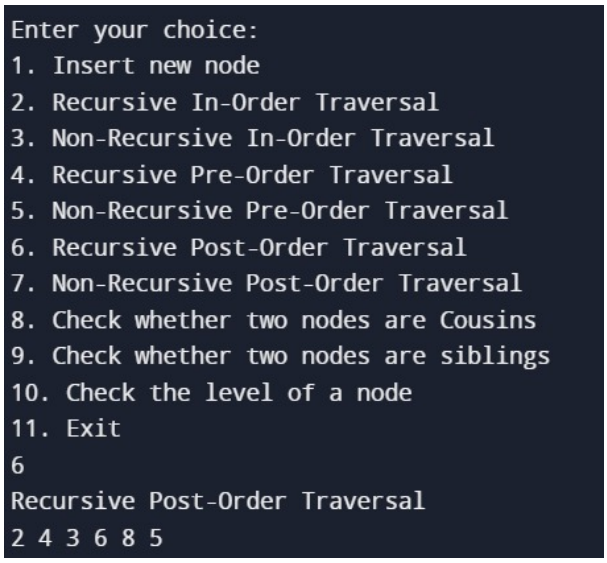

Figure 8: Recursive post-order traversal

NON-RECURSIVE POST-ORDER TRAVERSAL:

A non-recursive post-order traversal of a binary tree in C language is a way to traverse the tree without using recursion, visiting the left subtree first, then the right subtree, and finally the root node. Instead of recursion, the function uses two stacks to keep track of the nodes to be visited.

Algorithm for Non-recursive Post-order traversal of a binary tree:

1.Initialize two stacks, s1 and s2.
2.Push the root of the binary tree onto s1.
3.While s1 is not empty:
a. Pop the top element from s1 and push it onto s2.
b. If the popped node has a left child, push it onto s1.
c. If the popped node has a right child, push it onto s1.
4.While s2 is not empty: a. Pop the top element from s2 and process its data or print its value.
This algorithm uses two stacks to keep track of nodes that need to be processed. The first stack s1 is used to store nodes as they

are encountered during a pre-order traversal. The second stack s2 is used to store the nodes in reverse order, which is the order in which they would be processed in a post-order traversal. The nodes are popped from s2 and processed in order, resulting in a post-order traversal of the binary tree.

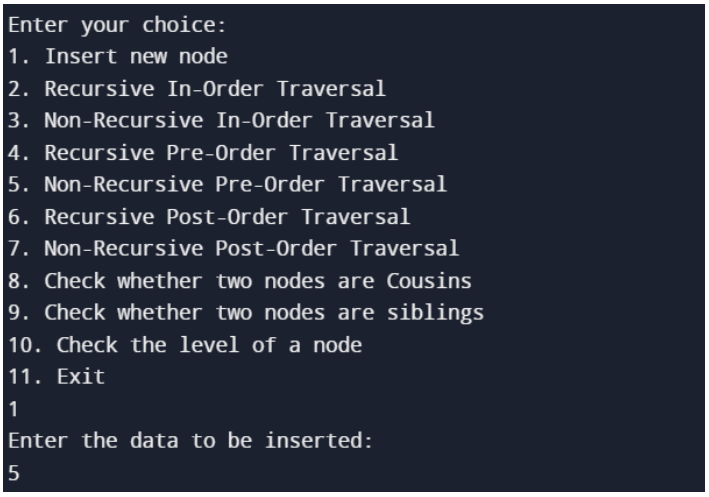

Figure: Insertion of new node

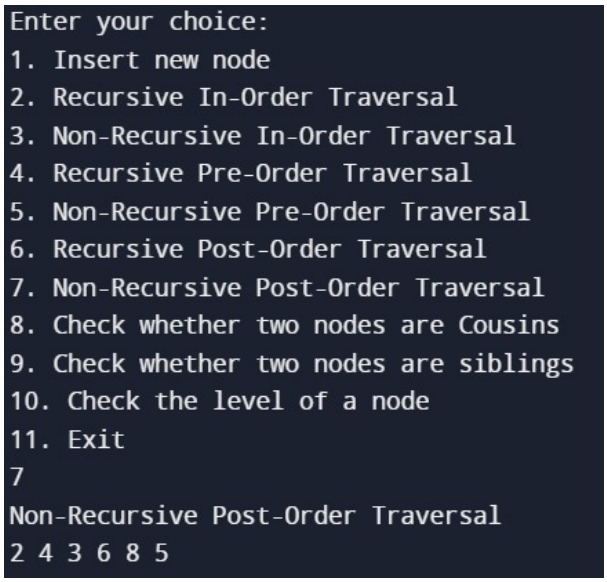

Figure 9: Non-recursive post-order traversal

CHECK THE LEVEL OF A GIVEN NODE:

The "check the level of a given node" function in a binary tree in C language is a function that determines the level of a particular node in the tree. The level of a node is the number of edges from the root node to the given node.

Algorithm for Checking the level of given node function in a binary tree:

1.Initialize a level counter to 1.
2.If the root of the binary tree is None, return 0.
3.Use a queue to keep track of nodes that need to be processed. Initialize the queue with the root node.
4.While the queue is not empty:
a. Dequeue the first node in the queue and store it in a temporary variable.
b. If the temporary node is equal to the target node, return the level counter.
c. If the temporary node has a left child, enqueue it into the

queue.
d. If the temporary node has a right child, enqueue it into the queue.
e. Increment the level counter.
5.Return 0 if the target node is not found in the binary tree.

This algorithm uses a breadth-first search (BFS) to traverse the binary tree, using a queue to keep track of nodes that need to be processed. The level of the node is incremented as it is dequeued from the queue, and the level counter is returned when the target node is found. If the target node is not found in the binary tree, the function returns 0.

For Level-checking of two nodes:

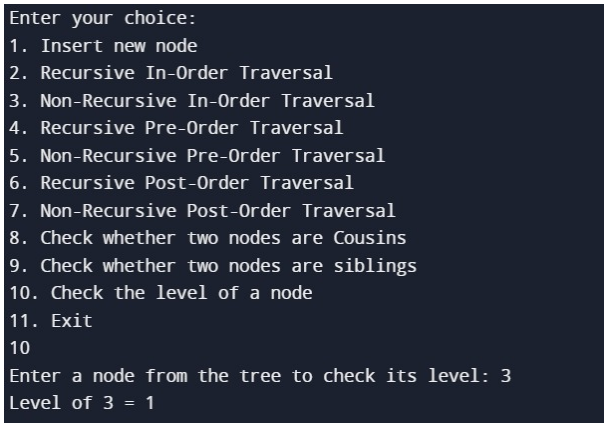

Figure 10: Level checking of two nodes

FIND WHETHER TWO NODES ARE COUSINS :

A cousin node in a binary tree is defined as a node that is at the same level as a given node but is not its sibling. To find the cousin nodes of a binary tree, the following steps can be followed: 1.Find the level of the given node: This can be done by performing a breadth-first search (BFS) or depth-first search (DFS) on the binary tree, marking the level of the given node as it is encountered. 2.Traverse the binary tree: After finding the level of the given node, traverse the binary tree again to find all the nodes that are at the same level as the given node.
3.Exclude the siblings: While traversing, if a node is found that is at the same level as the given node, check if it is the sibling of the given node. If it is, exclude it and continue the search. If it is not, add it to the list of cousin nodes.

Algorithm for determining whether two nodes are cousins in a binary tree: 1.If the root of the binary tree is None, return False. 2.Initialize two variables to store the depths of the two nodes.
3.Use a queue to keep track of nodes that need to be processed. Initialize the queue with a tuple containing the root node and a level counter set to 1.
4.While the queue is not empty:
a. Dequeue the first node in the queue and store it in a temporary

variable.
b. If the temporary node is equal to either of the target nodes, store its depth in the appropriate depth variable.
c. If both depth variables have been set, return True if they are equal and the parent nodes are different, otherwise return False.
d. If the temporary node has a left child, enqueue it into the queue with the level counter incremented by 1.
e. If the temporary node has a right child, enqueue it into the queue with the level counter incremented by 1.
5.Return False if either of the target nodes is not found in the binary tree.

This algorithm uses a breadth-first search (BFS) to traverse the binary tree, using a queue to keep track of nodes that need to be processed. The level of each node is stored in the queue, and the depths of the two target nodes are recorded when they are encountered. If both nodes are found and have the same depth, they are cousins if they have different parent nodes. If either node is not found, the function returns False.

```
Enter your choice:
1. Insert new node
2. Recursive In-Order Traversal
3. Non-Recursive In-Order Traversal
4. Recursive Pre-Order Traversal
5. Non-Recursive Pre-Order Traversal
6. Recursive Post-Order Traversal
7. Non-Recursive Post-Order Traversal
8. Check whether two nodes are Cousins
9. Check whether two nodes are siblings
10. Check the level of a node
11. Exit
8
Enter two nodes to check if they are cousins or not.
Enter the first node:
5
Enter the second node:
2
The two nodes are not cousins
```

Figure: Checking cousin: False Case

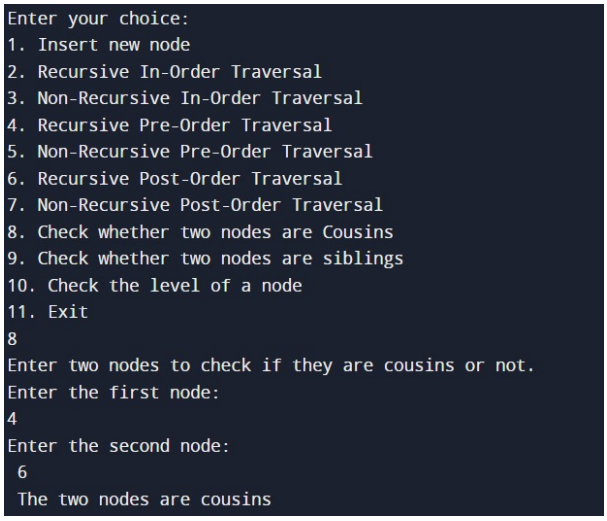

Figure 11: cousin nodes checking:True case

FIND WHETHER TWO NODES ARE SIBLING:

A sibling node in a binary tree is defined as a node that has the same parent as a given node. To find the sibling nodes of a binary tree, the following steps can be followed:
Find the parent node: This can be done by performing a breadth-first search (BFS) or depth-first search (DFS) on the binary tree, marking the parent node of the given node as it is encountered.
Check the left and right children of the parent node: If the parent node has a left child, check if it is the same as the given node. If it is not, add it to the list of sibling nodes. If the parent node has a right child, do the same.
Note: The steps mentioned above are just a general idea of how to find the sibling nodes of a binary tree. The implementation details may vary depending on the specific requirements.

Algorithm for determining whether two nodes are siblings in a binary tree:

1.If either of the nodes is None, return False.
2.Initialize a variable to store the parent node of the first target node.
3.Use a queue to keep track of nodes that need to be processed. Initialize the queue with a tuple containing the root node and its parent node (or None if it is the root).
4.While the queue is not empty:
a. Dequeue the first node in the queue and store it in a temporary variable.
b. If the temporary node is equal to either of the target nodes, store its parent node in the appropriate variable.
c. If both parent nodes have been set, return True if they are equal, otherwise return False.
d. If the temporary node has a left child, enqueue it into the queue with its parent node set to the temporary node.
e. If the temporary node has a right child, enqueue it into the queue with its parent node set to the temporary node.
5.Return False if either of the target nodes is not found in the binary tree.

This algorithm uses a breadth-first search (BFS) to traverse the binary tree, using a queue to keep track of nodes that need to be processed. The parent node of each node is stored in the queue, and the parent node of each target node is recorded when it is encountered. If both nodes are found and have the same parent node, they are siblings. If either node is not found, the function returns False.

```
Enter your choice:
1. Insert new node
2. Recursive In-Order Traversal
3. Non-Recursive In-Order Traversal
4. Recursive Pre-Order Traversal
5. Non-Recursive Pre-Order Traversal
6. Recursive Post-Order Traversal
7. Non-Recursive Post-Order Traversal
8. Check whether two nodes are Cousins
9. Check whether two nodes are siblings
10. Check the level of a node
11. Exit
9
Enter two nodes to check if they are siblings or not.
Enter the first node:
2
Enter the second node:
4
The two nodes are siblings.
```

```
Enter your choice:
1. Insert new node
2. Recursive In-Order Traversal
3. Non-Recursive In-Order Traversal
4. Recursive Pre-Order Traversal
5. Non-Recursive Pre-Order Traversal
6. Recursive Post-Order Traversal
7. Non-Recursive Post-Order Traversal
8. Check whether two nodes are Cousins
9. Check whether two nodes are siblings
10. Check the level of a node
11. Exit
9
Enter two nodes to check if they are siblings or not.
Enter the first node:
4
Enter the second node:
6
The two nodes are not siblings.
```

3 GLOBAL CONCLUSION

The project on binary trees implements various tree traversals and functions to check if given nodes are cousins or siblings. This project demonstrates how to traverse a binary tree using both recursion and non-recursive methods, as well as how to determine the relationship between two nodes in a binary tree. The implementation is done in C programming language, making use of a struct to represent a node in the binary tree and various functions to traverse and manipulate the tree. This project provides a comprehensive understanding of binary trees and its various operations, making it an excellent foundation for further exploration and learning in the field of tree data structures.

References:

1.Javatpoint: Tutorials Listhttps://www.javatpoint.com
2.GeeksforGeeks | A computer science portal for geekshttps://www.geeksforgeeks.org
3.Programiz: Learn to Code for Freehttps://www.programiz.com
4.Algorithms(S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani)
5.Fundamentals of Data Structures by Ellis Horowitz and Sartaj Sahni

Contribution by each group member:

S.No.	Name	Contribution
1.	JUNAIS MUSTHAFA	• areSiblings function, • edepth function, • Function to check whether two nodes are cousins • Report file
2.	CHINTAPENTA SURYA BHARADWAJA	• Creating the binary tree, • Non-recursive function for in-order traversal • Readme.txt file • Non-recursive function for post order traversal
3.	PRERANA BORA	• Function to insert new node in the binary tree, • Non-recursive pre-order function, • Non-recursive function for post order traversal • Report file
4.	RIGA SURESH M	• Algorithm design, • Recursive function for in-order traversal, • Non-recursive function for in-order traversal • Switch case
5.	RITIK SAHU	• license.txt
6.	ADITHYA K.P	• Install.txt, • Non-recursive function for pre-order traversal • isSiblings function • Report file
7.	SUMAN KUMARI	• authors.txt, • Recursive post-order traversal, • Non-recursive in-order traversal • license.txt file
8.	ARUNDHATI SENSHARMA	• isSibling function, • Function to check whether two nodes are cousins, • isSiblings function • Switch case