

**PROJECT TITLE**

**Submitted by: Abdul Samad Islam,Attaullah**

**Student ID: 4089,4868**

**Submitted to: MAM ARSHI**

**Subject: DATA STRUCTURE**

**Semester: 3 (Computer Science)**

**Section: BSCS (C)**

**Project Proposal: Lowest Common Ancestor (LCA) in a Binary Tree/Binary Search Tree**

**1. Project Title:** Lowest Common Ancestor (LCA) in a Binary Tree/BST

**2. Project Overview:** The objective of this project is to implement and analyze algorithms for finding the Lowest Common Ancestor (LCA) of two given nodes in a Binary Tree (BT) or a Binary Search Tree (BST). The LCA of two nodes is defined as the lowest node in the tree that has both nodes as descendants. This problem has applications in various domains, including computing hierarchies, network routing, and computational biology.

**3. Objectives:**

* Develop efficient algorithms to find the LCA in both a Binary Tree and a BST.
* Compare different approaches such as naive traversal, path-based solutions, and optimized methods using recursion and binary lifting.
* Implement the solution in C++/Python.
* Analyze the time and space complexity of different methods.
* Provide a user interface or command-line tool for testing the implementation.

**4. Methodology:**

* **Step 1:** Understand the problem statement and different approaches to solving LCA.
* **Step 2:** Implement the basic naive solution using node traversal and parent tracking.
* **Step 3:** Implement optimized solutions:
  + Using recursion for Binary Trees.
  + Using BST properties for faster computation in BSTs.
  + Using Binary Lifting for advanced optimization.
* **Step 4:** Test the implementations on different test cases.
* **Step 5:** Analyze and compare performance metrics.
* **Step 6:** Document the results and provide insights into the efficiency of different approaches.

**5. Expected Outcome:**

* A functional implementation of LCA for Binary Trees and BSTs.
* A comparative analysis of various approaches based on time complexity and efficiency.
* A user-friendly way to input trees and query LCA for two nodes.

Conclusion:

The project aims to develop efficient algorithms for finding the **Lowest Common Ancestor (LCA)** in Binary Trees and BSTs. It compares multiple approaches to determine the most optimal solution for different scenarios. The implementation will be tested and analyzed for performance efficiency. A user-friendly interface or CLI will be developed for easy interaction. Ultimately, this project provides valuable insights into tree algorithms and their practical applications.

#include <iostream>

using namespace std;

// Definition of Tree Node

struct Node {

int data;

Node\* left;

Node\* right;

Node(int val) {

data = val;

left = right = NULL;

}

};

// Function to find LCA in a Binary Tree

Node\* findLCA(Node\* root, int n1, int n2) {

if (root == NULL) return NULL;

if (root->data == n1 || root->data == n2) return root;

Node\* leftLCA = findLCA(root->left, n1, n2);

Node\* rightLCA = findLCA(root->right, n1, n2);

if (leftLCA && rightLCA) return root;

return (leftLCA != NULL) ? leftLCA : rightLCA;

}

// Function to find LCA in a BST

Node\* findLCA\_BST(Node\* root, int n1, int n2) {

if (root == NULL) return NULL;

if (root->data > n1 && root->data > n2) {

return findLCA\_BST(root->left, n1, n2);

}

if (root->data < n1 && root->data < n2) {

return findLCA\_BST(root->right, n1, n2);

}

return root;

}

// Function to insert a node into BST

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

if (root == NULL) return new Node(key);

if (key < root->data) root->left = insert(root->left, key);

else root->right = insert(root->right, key);

return root;

}

// Main function to test LCA functions

int main() {

// Creating a Binary Search Tree (BST)

Node\* root = NULL;

root = insert(root, 20);

insert(root, 10);

insert(root, 30);

insert(root, 5);

insert(root, 15);

insert(root, 25);

insert(root, 35);

int n1 = 5, n2 = 15;

Node\* lcaBST = findLCA\_BST(root, n1, n2);

cout << "LCA of " << n1 << " and " << n2 << " in BST: " << lcaBST->data << endl;

// Creating a Binary Tree

Node\* rootBT = new Node(1);

rootBT->left = new Node(2);

rootBT->right = new Node(3);

rootBT->left->left = new Node(4);

rootBT->left->right = new Node(5);

rootBT->right->left = new Node(6);

rootBT->right->right = new Node(7);

n1 = 4, n2 = 5;

Node\* lcaBT = findLCA(rootBT, n1, n2);

cout << "LCA of " << n1 << " and " << n2 << " in Binary Tree: " << lcaBT->data << endl;