Problem adapted from an original teaching example developed with assistance from ChatGPT (OpenAI, 2025).

"Kingdom Roadmap" Problem: Building and Querying a Medieval Kingdom's Messenger Network

Scenario:

The Kingdom of *Arboria* has a network of villages connected by roads. The King wants to build a communication network where each village has exactly one road leading to another (like a messenger chain), forming a tree structure with the capital at the root. Your task is to help model this network using a binary tree and support queries such as finding routes and relationships between villages.

Problem Tasks:

1. **Build the Tree:**
   Each village has a unique name. Insert villages into the network such that alphabetically earlier names go to the left, and later ones go to the right.
2. **Display the Network:**
   Show the village names in alphabetical order (in-order traversal).
3. **Find a Route:**
   Given two villages, print the path from one to the other (common ancestor + path logic).

```cpp
#include <iostream>
#include <string>
#include <vector>
using namespace std;

struct Village {
    string name;
    Village* left = nullptr;
    Village* right = nullptr;

    Village(const string& n) : name(n) {}
};


// Insert a village based on name (BST)
Village* insert(Village* root, const string& name) {
    if (!root) return new Village(name);
    if (name < root->name)
        root->left = insert(root->left, name);
    else if (name > root->name)
        root->right = insert(root->right, name);
    return root;
}

// In-order traversal
void printInOrder(Village* root) {
    if (!root) return;
    printInOrder(root->left);
    cout << root->name << " ";
    printInOrder(root->right);
}

// Find a node
Village* find(Village* root, const string& name) {
    if (!root) return nullptr;
    if (name == root->name) return root;
    if (name < root->name) return find(root->left, name);
    return find(root->right, name);
}

// Check if ancestor
bool isAncestor(Village* root, const string& ancestor, const string& descendant) {
    Village* anc = find(root, ancestor);
    if (!anc) return false;
    return find(anc, descendant) != nullptr;
}
```

```cpp
// Find path from root to target, store path in vector
bool pathTo(Village* root, const string& target, vector<string>& path) {
    if (!root) return false;
    path.push_back(root->name);
    if (root->name == target) return true;
    if ((root->left && pathTo(root->left, target, path)) ||
        (root->right && pathTo(root->right, target, path)))
        return true;
    path.pop_back();
    return false;
}

// Print path between two villages
void printPathBetween(Village* root, const string& from, const string& to) {
    vector<string> pathFrom, pathToResult;
    if (!pathTo(root, from, pathFrom) || !pathTo(root, to, pathToResult)) {
        cout << "One or both villages not found.\n";
        return;
    }

    // Find where paths diverge (Lowest Common Ancestor)
    size_t i = 0;
    while (i < pathFrom.size() &&
           i < pathToResult.size() &&
           pathFrom[i] == pathToResult[i])
        ++i;

    // Print path from 'from' up to LCA (excluding LCA)
    for (int j = pathFrom.size() - 1; j >= static_cast<int>(i); --j)
        cout << pathFrom[j] << " -> ";
    // Print path from LCA to 'to'
    for (size_t j = i - 1; j < pathToResult.size(); ++j)
        cout << pathToResult[j] << (j + 1 < pathToResult.size() ? " -> " : "\n");
}

// Recursively delete the tree to avoid memory leaks
void deleteTree(Village* root) {
    if (!root) return;
    deleteTree(root->left);
    deleteTree(root->right);
    delete root;
}
```

```cpp
int main() {
    Village* root = nullptr;
    vector<string> villageNames = { "Town-A", "Town-C", "Town-X",
                                    "Town-B", "Town-F", "Town-E", "Town-D" };

    for (const string& name : villageNames)
        root = insert(root, name);

    cout << "Kingdom network (in order): ";
    printInOrder(root);
    cout << "\n";

    string A = "Town-X", B = "Town-C";
    cout << "Is " << A << " an ancestor of " << B << "? ";
    cout << (isAncestor(root, A, B) ? "Yes\n" : "No\n");

    cout << "Path from " << A << " to " << B << ":\n";
    printPathBetween(root, A, B);

    deleteTree(root); // Clean up memory
    return 0;
}
```