**Project Overview**

This project aims to implement a Binary Search Tree (BST) using C++. The BST will support three primary operations: search, insert, and delete. These operations are fundamental for managing and organizing data efficiently in various applications such as databases, file systems, and memory management.

**Technologies and Tools**

* **Programming Language**: C++
* **Development Environment**: Visual Studio
* **Version Control**: Git, GitHub

**Project Details**

1. **Binary Search Tree (BST) Overview**:
   * A BST is a node-based binary tree data structure.
   * Each node has a key, and every node's key is larger than all keys in its left subtree and smaller than all keys in its right subtree.
   * The primary operations on a BST include search, insert, and delete.
2. **BST Functions**:
   * **Search**:
     + Function: Node\* search(Node\* root, int key)
     + Description: This function takes the root node of the BST and a key as input, and returns the node containing the key if it exists in the BST. If the key is not found, it returns nullptr.
     + Algorithm:
       - Start from the root node.
       - If the key is equal to the root's key, return the root.
       - If the key is smaller than the root's key, recursively search the left subtree.
       - If the key is larger than the root's key, recursively search the right subtree.
   * **Insert**:
     + Function: Node\* insert(Node\* root, int key)
     + Description: This function takes the root node of the BST and a key as input, and inserts the key into the BST while maintaining the BST properties.
     + Algorithm:
       - If the root is nullptr, create a new node with the given key and return it.
       - If the key is smaller than the root's key, recursively insert it into the left subtree.
       - If the key is larger than the root's key, recursively insert it into the right subtree.
       - Return the root node after insertion.
   * **Delete**:
     + Function: Node\* deleteNode(Node\* root, int key)
     + Description: This function takes the root node of the BST and a key as input, and deletes the node with the given key from the BST while maintaining the BST properties.
     + Algorithm:
       - If the root is nullptr, return nullptr.
       - If the key is smaller than the root's key, recursively delete it from the left subtree.
       - If the key is larger than the root's key, recursively delete it from the right subtree.
       - If the key is equal to the root's key:
         * If the node has no children, delete the node and return nullptr.
         * If the node has one child, delete the node and return the child.
         * If the node has two children, find the inorder successor (smallest node in the right subtree), replace the root's key with the successor's key, and recursively delete the successor.
       - Return the root node after deletion.

Conclusion:

This overview provides understanding and working with Binary Search Trees in C++. By implementing search, insert, and delete operations, the program illustrates the dynamic nature of BSTs and their ability to maintain order while allowing efficient data retrieval and modification. The provided code can serve as a base for more advanced tree operations and optimizations, making it a valuable resource for both learning and practical applications.