DS2030 Data Structures and Algorithms for Data Science
Lab 5 (In Person) Due on October 1st, 5.00pm

Instructions

- You are to use Python as the programming language. Use may use Visual Studio Code (or any other editor you are comfortable with) as the IDE.
- You have to work individually for this lab.
- You are not allowed to share code with your classmates nor allowed to use code from the internet. You are encouraged to engage in high-level discussions with your classmates; however, ensure to include their names in the report/code documentation. If you refer to any source on the Internet, include the corresponding citation in the report/code documentation. If we find that you have copied code from your classmate or from the Internet, you will get a straight fail grade in the course.
- The submission must be a zip file with the following naming convention - rollnumber.zip. The zip filer should contain a single Python file corresponding to your implementation.
- Include appropriate comments to document the code. Include a `read me` file containing the instructions on for executing the code. The code should run on institute linux machines.
- Upload your submission to moodle by the due date and time. Do not email the submission to the instructor or the TA.

This lab will improve your understanding of tree structures and traversals.

Objective

In this lab, you will implement the fundamental operations of a Binary Search Tree (BST) in Python. You will:

- Implement the recursive **search**, **insert**, and **inorder traversal** operations.
- Explore how different insertion orders can result in the same BST.

By the end of this lab, you should have a deeper understanding of how BSTs work and how they maintain their structure.

1 Binary Search Tree Overview

A **binary search tree (BST)** is a tree in which each node contains a value. For each node:

- Values in the left subtree are smaller than the node's value.
- Values in the right subtree are larger than the node's value.

The three main operations we will focus on are:

- **Insertion**: Adding a value into the correct position in the tree.
- **Search**: Finding if a value exists in the tree.
- **Inorder Traversal**: Printing the values in the tree in sorted order (left-root-right).

2 Starter Code

Below is the starter code. The class `TreeNode` defines a single node of the tree, and the class `BinarySearchTree` defines the structure of the tree.

```python
# Define the structure of the binary tree node
class TreeNode:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None

# Binary Search Tree Class
class BinarySearchTree:
    def __init__(self):
        self.root = None

    # TODO: Implement this function
    def insert(self, val):
        pass

    # TODO: Implement this function
    def search(self, val):
        pass

    # TODO: Implement this function
    def inorder_traversal(self):
        pass

# Example usage
if __name__ == "__main__":
    bst = BinarySearchTree()
    elements = [5, 3, 8, 2, 4, 7, 9]
    for elem in elements:
        bst.insert(elem)

    # Call the inorder_traversal to see the tree's sorted values
    print("Inorder Traversal:", bst.inorder_traversal())

    # Search for a value
    search_value = 4
    print(f"Is {search_value} in the tree? {bst.search(search_value)}")
```

3 Tasks

Your task is to implement the three key functions: **insert**, **search**, and **inorder traversal**. Below are the details for each function.

3.1 Task 1: Recursive Insert

You need to implement a recursive function that inserts a value into the correct position in the BST. The function should:

- Start at the root.
- If the value is less than the current node's value, move to the left child. If there is no left child, insert the new node there.
- If the value is greater than the current node's value, move to the right child. If there is no right child, insert the new node there.

Hints:

- You may need to create a helper recursive function inside `insert`.

3.2 Task 2: Recursive Search

You need to implement a recursive function to search for a value in the tree. The function should:

- Return `True` if the value is found, otherwise `False`.
- If the value is less than the current node, search in the left subtree.
- If the value is greater than the current node, search in the right subtree.

Hints:

- Base case: If the current node is `None`, the value is not in the tree.

3.3 Task 3: Recursive Inorder Traversal

The inorder traversal prints the elements of the tree in ascending order. The traversal follows these steps:

- Recursively traverse the left subtree.
- Print the root node's value.
- Recursively traverse the right subtree.

Hints:

- You may need a helper recursive function for this task that returns the traversal as a list.

4 Determine the Orderings of Insertion Resulting in the Same BST

Once you have implemented the insertion function, try to explore how different sequences of insertion result in the same BST. For example, the sequences:

- `[5, 3, 8, 2, 4, 7, 9]`
- `[5, 8, 3, 2, 4, 7, 9]`

will both result in the same BST structure.
Write a function that generates all possible reorderings of the input list that result in the same BST. You'll need to consider the relative order of elements in the left and right subtrees to ensure that each subtree maintains the BST property. The function should print all possible reorderings.

Hints:

- You can use a recursive approach to interleave the left and right subtree insertions. For each node, consider all possible ways to interleave its left and right subtrees while maintaining their relative order.