Binary Search Tree (BST) Guide

Binary Search Tree (BST) Guide

A Binary Search Tree (BST) is a binary tree where:

- The left child of a node contains a key less than the node's key.
- The right child of a node contains a key greater than the node's key.
- No duplicate keys exist in the tree.

Each node in a BST consists of:

- key: The value stored in the node.
- left: A reference to the left child.
- right: A reference to the right child.

BST Insertion

To insert a key into a BST:

1. Start at the root node.
2. If the key is smaller than the current node, move to the left subtree.
3. If the key is larger, move to the right subtree.
4. Repeat until a nil (empty) position is found, then insert the new node.

Example insertion function:

```
class TreeNode:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

def insert(root, key):
    if root is None:
        return TreeNode(key)
    if key < root.key:
        root.left = insert(root.left, key)
    else:
        root.right = insert(root.right, key)
    return root
```

BST Search

To search for a key in a BST:

1. Start at the root.
2. If the key matches, return the node.
3. If the key is smaller, search in the left subtree.
4. If the key is larger, search in the right subtree.
5. If the key is not found, return None.

Example search function:

```
def search(root, key):
    if root is None or root.key == key:
        return root
    if key < root.key:
        return search(root.left, key)
    return search(root.right, key)
```

BST Deletion

To delete a node with key k:

1. If the node has no children, remove it.
2. If the node has one child, replace it with its child.
3. If the node has two children:
    - Find the successor (smallest node in the right subtree).
    - Replace the node with its successor.
    - Delete the successor from its original position.

Example deletion function:

```
def find_min(node):
    while node.left:
        node = node.left
    return node

def delete(root, key):
    if root is None:
        return root
    if key < root.key:
        root.left = delete(root.left, key)
    elif key > root.key:
        root.right = delete(root.right, key)
    else:
        if root.left is None:
            return root.right
```

```
        elif root.right is None:
            return root.left
        temp = find_min(root.right)
        root.key = temp.key
        root.right = delete(root.right, temp.key)
    return root
```

BST Traversal Methods

Inorder Traversal (Left, Root, Right)
Visits nodes in ascending order for a BST.

```
def inorder(root):
    if root:
        inorder(root.left)
        print(root.key, end=" ")
        inorder(root.right)
```

Example Output:
For a BST with nodes {7, 4, 12, 2, 6, 9, 15}, the inorder traversal prints:
2 4 6 7 9 12 15

Preorder Traversal (Root, Left, Right)
Used for copying a tree.

```
def preorder(root):
    if root:
        print(root.key, end=" ")
        preorder(root.left)
        preorder(root.right)
```

Postorder Traversal (Left, Right, Root)
Used for deleting a tree (deletes children before the parent)

```
def postorder(root):
    if root:
        postorder(root.left)
        postorder(root.right)
        print(root.key, end=" ")
```

Building a BST from a List

If you provide a list of values, you can construct a BST dynamically.

Example function:

```
def build_bst_from_list(values):
    root = None
    for value in values:
        root = insert(root, value)
    return root
```

Example usage:

```
values = [10, 5, 15, 2, 7, 12, 18]
root = build_bst_from_list(values)
inorder(root)  # Output: 2 5 7 10 12 15 18
```

BST Rules for LLM Completion

To ensure the LLM can complete a BST when given an incomplete tree, it should:

- Learn insertion patterns from partial trees.
- Understand traversal outputs and expected orders.
- Recognize BST properties, ensuring left children are smaller and right children are larger.

Prompt Examples

- "Given the BST {10, 5, 15}, insert 7, 12, and 18."
  Expected Response: {10, 5, 15, 7, 12, 18}
- "What is the inorder traversal of {7, 4, 12, 2, 6, 9, 15}?"
  Expected Response: 2, 4, 6, 7, 9, 12, 15
- "Delete node 6 from {7, 4, 12, 2, 6, 9, 15}."
  Expected Response: {7, 4, 12, 2, 9, 15}