VED BECHARA_23BCE11745

#Algorithm for Building a Binary Search Tree (Iterative Insertion)

The following algorithm uses an iterative approach to construct a binary search tree from a sentence, with words sorted alphabetically.

1. **Define a Node Structure:** Create a Node class with a string field for the word and two Node pointers for the left and right children.

2. **Initialize the Tree:** Create a BinarySearchTree class with a root attribute, initially set to null.

3. **Prepare the Input:**
   - Take the sentence "The quick brown fox jumps over the lazy dog."
   - Convert the entire sentence to lowercase.
   - Split the sentence into an array of words, handling punctuation and multiple spaces.

4. **Iterate and Insert:** Loop through each word from the prepared list. For each word, call an insert function.

5. **The Iterative Insertion Function (insert):** This function places a new word into the tree without using recursion.
   - **Step 5a (Empty Tree):** If the tree's root is null, create a new node for the word and set it as the root.
   - **Step 5b (Iterative Traversal):** If the tree is not empty, use a while loop and a current pointer starting at the root.
     - **Comparison:** Compare the new word to the data of the current node.
     - **Go Left:** If the new word is alphabetically less than the current node's word, check its left child. If the left child is null, insert the new node there and break the loop. Otherwise, move the current pointer to the left child.
     - **Go Right:** If the new word is alphabetically greater than the current node's word, check its right child. If the right child is null, insert the new node there and break the loop. Otherwise, move the current pointer to the right child.
     - **Duplicate:** If the new word is equal to the current node's word, do nothing and break the loop.

6. **Verify the Tree:** After inserting all words, perform a standard in-order traversal to confirm the tree correctly stores the words in alphabetical order.

#Algo from rosen book:-

ALGORITHM 1 Locating and Adding Items to a Binary Search Tree.

procedure *insertion*(T: binary search tree, x: item)
$v :=$ root of T
{a vertex not present in T has the value *null*}
while $v \neq$ *null* and *label*(v) $\neq x$
begin
  if $x <$ *label*(v) **then**
    if left child of $v \neq$ *null* **then** $v :=$ left child of v
    else add *new vertex* as a left child of v and set $v :=$ *null*
  else
    if right child of $v \neq$ *null* **then** $v :=$ right child of v
    else add *new vertex* as a right child of v to T and set $v :=$ *null*
end
if root of $T =$ *null* **then** add a vertex v to the tree and label it with x
else if v is null or *label*(v) $\neq x$ **then** label *new vertex* with x and let v be this new vertex
{$v =$ location of x}

#code in JAVA for this problem solving:-

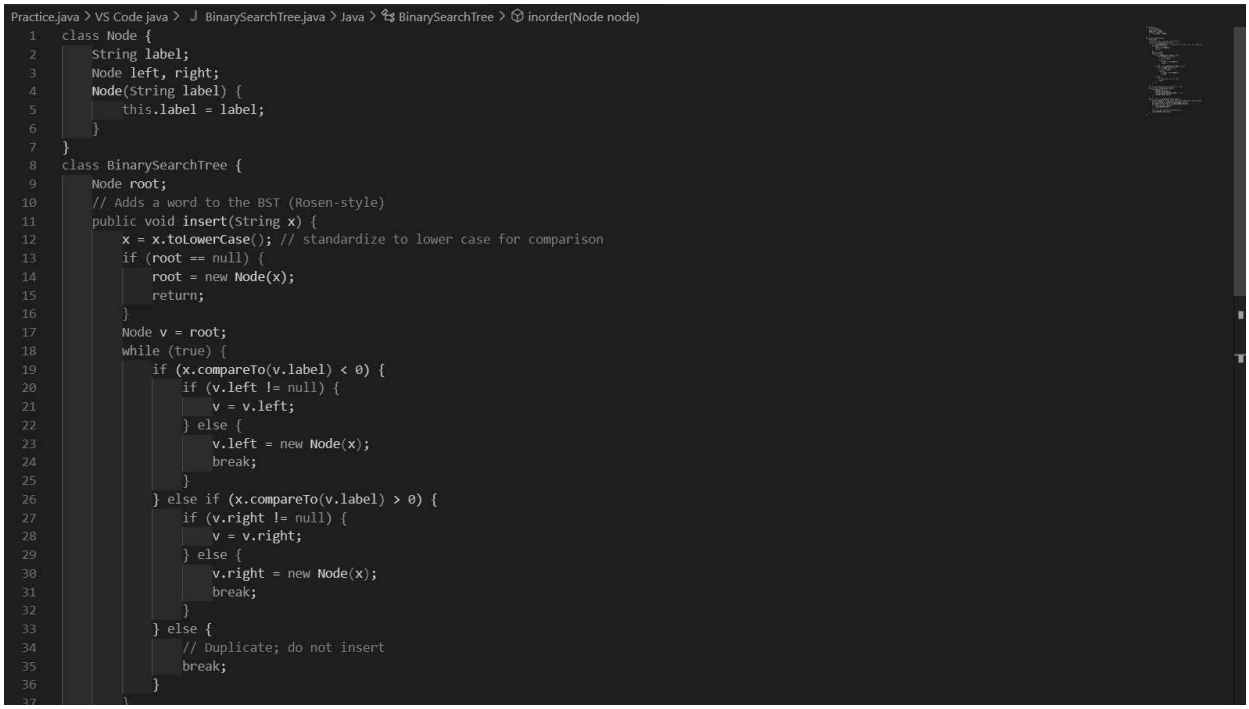
Que:

5. Using alphabetical order, construct a binary search tree for the words in the sentence "*The quick brown fox jumps over the lazy dog.*"

Ans : brown dog fox jumps lazy over quick the

```
C:\Users\Heetv\OneDrive\Desktop\Codes\Java>cd "c:\Users\Heetv\OneDrive\Desktop\Codes\Java\Practice.java\VS Code java\" && javac BinarySearchTree.java && j
ava BinarySearchTree
brown dog fox jumps lazy over quick the
c:\Users\Heetv\OneDrive\Desktop\Codes\Java\Practice.java\VS Code java>
```

Code :



```
Practice.java > VS Code java > BinarySearchTree.java > Java > BinarySearchTree > inorder(Node node)
class Node {
    String label;
    Node left, right;
    Node(String label) {
        this.label = label;
    }
}
class BinarySearchTree {
    Node root;
    // Adds a word to the BST (Rosen-style)
    public void insert(String x) {
        x = x.toLowerCase(); // standardize to lower case for comparison
        if (root == null) {
            root = new Node(x);
            return;
        }
        Node v = root;
        while (true) {
            if (x.compareTo(v.label) < 0) {
                if (v.left != null) {
                    v = v.left;
                } else {
                    v.left = new Node(x);
                    break;
                }
            } else if (x.compareTo(v.label) > 0) {
                if (v.right != null) {
                    v = v.right;
                } else {
                    v.right = new Node(x);
                    break;
                }
            } else {
                // Duplicate; do not insert
                break;
            }
        }
    }
    // Traverses BST and prints labels in order
    public void inorder(Node node) {
        if (node != null) {
            inorder(node.left);
            System.out.print(node.label + " ");
            inorder(node.right);
        }
    }
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        String sentence = "The quick brown fox jumps over the lazy dog";
        String[] words = sentence.split(regex:"\\s+");
        BinarySearchTree tree = new BinarySearchTree();
        for (String word : words) {
            tree.insert(word);
        }
        // Output BST contents alphabetically
        tree.inorder(tree.root);
    }
}
```

```
PROBLEMS 7  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

brown dog fox jumps lazy over quick the
c:\Users\Heetv\OneDrive\Desktop\Codes\Java\Practice.java\VS Code java>cd "c:\Users\Heetv\OneDrive\Desktop\Codes\Java\Practice.java\VS Code java\" && javac
 BinarySearchTree.java && java BinarySearchTree
brown dog fox jumps lazy over quick the
c:\Users\Heetv\OneDrive\Desktop\Codes\Java\Practice.java\VS Code java>

Code
Code
Code
```