KATHMANDU UNIVERSITY
DHULIKHEL KAVRE

Subject COMP 202

Lab Sheet No.4

SUBMITTED BY

Name: - Sumesh Shrestha

Roll No: - 52

Group: - CE(II/I)

Level: - 2nd year / 1st semester

SUBMITTED TO

Rajani Chulyadyo

Department of Computer Science and Engineering

Date of submission: - 02/06/2023

Implement Binary Search Tree with the following operations:
(a) isEmpty(): Returns true if the tree is empty, and false otherwise
(b) addBST(key, value): Inserts an element to the BST
(c) removeBST(keyToDelete): Removes the node with the given key from the BST
(d) searchBST(targetKey): Returns true if the key exists in the tree, and false otherwise implement it using linked data structure.
Also, write a test program to check if the implementation works properly.

Explanation:

The LinkedListBST implementation uses a linked data structure where each node in the tree is represented by a struct called "Node". Each Node contains a key, a value, and pointers to its left and right child nodes.

For Insertion(adsdBST): To insert a new element into the BST, we start from the root node. If the tree is empty, we create a new node and make it the root.If the tree is not empty, we compare the key of the new element with the key of the current node. If the new key is less than the current node's key, we move to the left child node. If the new key is greater than the current node's key, we move to the right child node. We repeat this process until we find an empty spot where the new node can be inserted. Once we find the appropriate spot, we create a new node with the given key and value and attach it as a left or right child of the last node we visited.

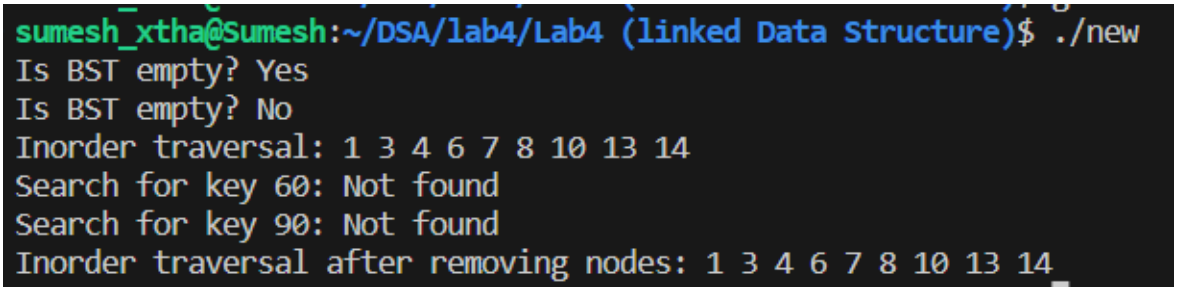
For Removal (removeBST):
To remove a node from the BST, we start from the root node and search for the node with the given key.
If the key is not found, we simply exit the function.
If the key is found, we handle three cases based on the number of children the node has:

- Case 1: Node has no children
  We update the parent's reference to the node to be deleted and delete the node.
- Case 2: Node has one child
  We update the parent's reference to point to the child of the node to be
- Case 3: Node has two children
  We find the in-order successor (the node with the smallest key in the right subtree) of the node to be deleted.

We copy the key and value from the in-order successor to the node to be deleted. We recursively call removeBST on the in-order successor node to delete it.

For Searching (searchBST):
To search for a key in the BST, we start from the root node and compare the target key with the key of the current node. If the target key matches the current node's key, we return true. If the target key is less than the current node's key, we move to the left child node. If the target key is greater than the current node's key, we move to the right child node. We repeat this process until we find a matching key or reach a null node. If we reach a null node, it means the key does not exist in the BST, and we return false.

Output:



Github link: https://github.com/Saumyashrestha/CE2021_Lab4_51_52.git