Habib University

CS 102 – Data Structures & Algorithm

Lab# 14

Binary Search Tree

Objectives: In this lab, we will implement Binary Search Tree.

Binary Search Tree Visualizer:

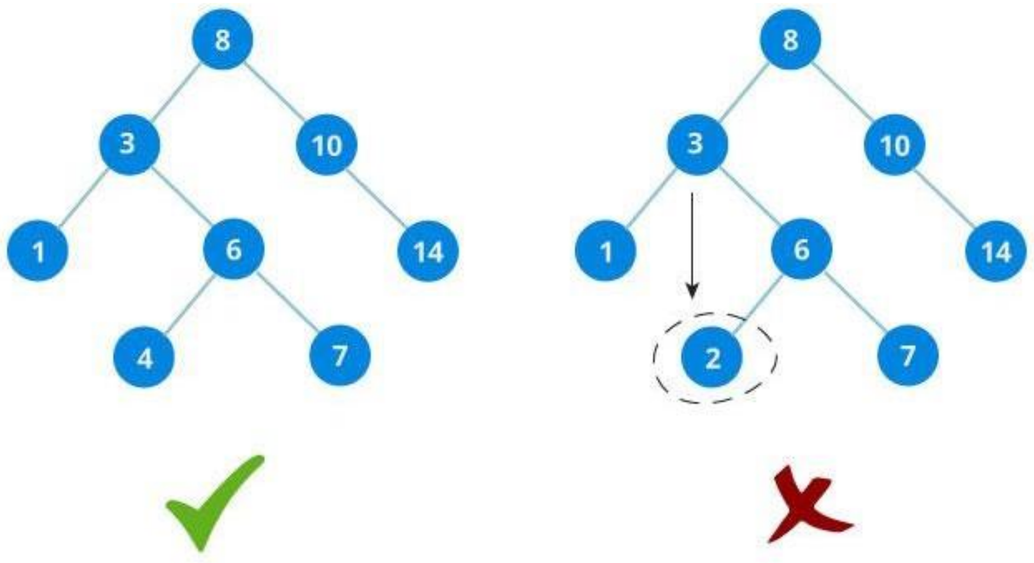
https://visualgo.net/bn/bst

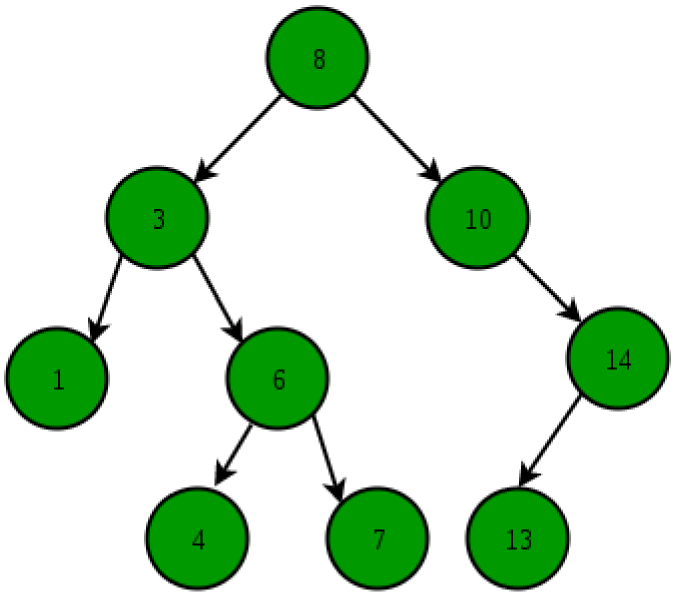
http://btv.melezinek.cz/binary-search-tree.html

Binary Search Tree, is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.
- There must be no duplicate nodes.

The above properties of Binary Search Tree provide an ordering among keys so that the operations like search, minimum and maximum can be done fast. If there is no ordering, then we may have to compare every key to search a given key.





This Binary Search Tree has resulted from inserting the following keys in the order given: 8, 3, 10, 1, 6, 14, 4, 7 and 13.

We can represent the Binary Search Tree using Nested Dictionary in Python.

{'value': 8, 'left': {'value': 3, 'left': {'value': 1, 'left': {}, 'right': {}}, 'right': {'value': 6, 'left': {'value': 4, 'left': {}, 'right': {}}, 'right': {'value': 7, 'left': {}, 'right': {}}}}, 'right': {'value': 10, 'left': {}, 'right': {'value': 14, 'left': {'value': 13, 'left': {}, 'right': {}}, 'right': {}}}}

Binary Search Tree Helper Functions

Insertion of a key

A new key is always inserted at leaf. We start searching a key from root till we hit a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node.

Helper Function #1. Write a function **insert (bst, key)** that takes a Binary Search Tree and key as a parameter and insert key into the Binary Search Tree.

Searching a key

To search a given key in Binary Search Tree, we first compare it with root, if the key is present at root, we return root. If key is greater than root's key, we recur on the right subtree; otherwise we recur on the left subtree.

Helper Function #2. Write a function **exist (bst, key)** that takes a Binary Search Tree and key as a parameter and returns True if the key exists; False otherwise.

Minimum Value in Binary Search Tree

Just traverse the node from root to left recursively until left is NULL. The node whose left is NULL is the node with minimum value.

Helper Function #3. Write a function **minimum (bst, starting_node)** that takes a Binary Search Tree & a starting node as a parameter and returns the minimum value in the binary search tree from the starting node.

Maximum Value in Binary Search Tree

Just traverse the node from root to right recursively until right is NULL. The node whose right is NULL is the node with maximum value.

Helper Function #4. Write a function **maximum (bst, starting_node)** that takes a Binary Search Tree and a starting node as a parameter and returns the maximum value in the binary search tree from the starting node.

In-order Traversal in Binary Search Tree

1. Traverse the left subtree, i.e., call Inorder(left-subtree)
2. Visit the root.
3. Traverse the right subtree, i.e., call Inorder(right-subtree)

Helper Function #5. Write a function **inorder_traversal (bst)** that takes a Binary Search Tree and print in-order traversal of the Binary Search Tree.

Pre-order Traversal in Binary Search Tree

1. Visit the root.
2. Traverse the left subtree, i.e., call Preorder(left-subtree)
3. Traverse the right subtree, i.e., call Preorder(right-subtree)

Helper Function #6. Write a function **preorder_traversal (bst)** that takes a Binary Search Tree and print pre-order traversal of the Binary Search Tree.

Post-order Traversal in Binary Search Tree

1. Traverse the left subtree, i.e., call Postorder(left-subtree)
2. Traverse the right subtree, i.e., call Postorder(right-subtree)
3. Visit the root.

Helper Function #7. Write a function **postorder_traversal (bst)** that takes a Binary Search Tree and prints post-order traversal of the Binary Search Tree.

Question 1:

a) Using the helper function **insert (bst, key)**, create the binary search tree that results from inserting the following keys in the order given: 68, 88, 61, 89, 94, 50, 4, 76, 66, and 82.

b) Using the helper function **exist (bst, key)**, check whether key 50 exists in resultant Binary Search Tree.

c) Using the helper function **exist (bst, key)**, check whether key 49 exists in resultant Binary Search Tree.

d) Using the helper function **minimum (bst, starting_node)**, find the node with the minimum value in resultant Binary Search Tree from starting node = 68.

e) Using the helper function **minimum (bst, starting_node)**, find the node with the minimum value in resultant Binary Search Tree from starting node = 88.

f) Using the helper function **maximum (bst, starting_node)**, find the node with the maximum value in resultant Binary Search Tree from starting node = 68.

g) Using the helper function **maximum (bst, starting_node)**, find the node with the maximum value in resultant Binary Search Tree from starting node = 61.

h) Using the helper function **inorder_traversal (bst)**, perform in-order traversal of the Binary Search Tree.

i) Using the helper function **preorder_traversal (bst)**, perform pre-order traversal of the Binary Search Tree.

j) Using the helper function **postorder_traversal (bst)**, perform post-order traversal of the Binary Search Tree.

Question 2:

Organize the following keys in a binary search tree such that the keyword in the root is alphabetically bigger than all the keywords in the left subtree and smaller than all the keywords in the right subtree (and this holds for all nodes).

begin, do, else, end, if, then, while