Insertion in a BST

A Binary Search Tree (BST) is a rooted binary tree, whose nodes each store a key (and optionally, an associated value) and

each has two distinguished subtrees, commonly denoted left and right. The tree should satisfy the BST property,

which states that each node’s key must be greater than all keys stored in the left subtree and not greater

than all keys in the right subtree. Ideally, unique values should be present in the tree.

15

/ \

10 20

/ \ / \

8 12 16 25

Binary search trees are a fundamental data structure used to construct more abstract data structures such as sets,

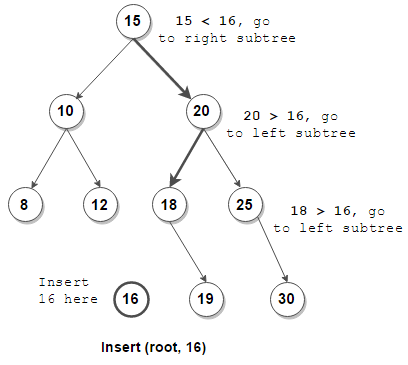
multisets, and associative arrays (maps, multimaps, etc.).

When looking for a place to insert a new key, traverse the tree from root-to-leaf,

making comparisons to keys stored in the tree’s nodes and deciding based on the comparison to continue searching in the left or right subtrees.

In other words, we examine the root and recursively insert the new node to the left subtree if its key is less than that of the root or

the right subtree if its key is greater than or equal to the root.



package com.gl.dsa;

class Node

{

int data;

Node left, right;

// Function to create a new binary tree node having a given key

Node(int key)

{

data = key;

left = right = null;

}

}

public class BSTSample {

public static void inorder(Node root)

{

if (root == null) {

return;

}

*inorder*(root.left);

System.*out*.print(root.data + " ");

*inorder*(root.right);

}

// Recursive function to insert a key into a BST

public static Node insert(Node root, int key)

{

// if the root is null, create a new node and return it

if (root == null) {

return new Node(key);

}

// if the given key is less than the root node,

// recur for the left subtree

if (key < root.data) {

root.left = *insert*(root.left, key);

}

// otherwise, recur for the right subtree

else {

// key >= root.data

root.right = *insert*(root.right, key);

}

return root;

}

// Function to construct a BST from given keys

public static Node constructBST(int[] keys)

{

Node root = null;

for (int key: keys) {

root = *insert*(root, key);

}

return root;

}

public static void main(String[] args) {

// TODO Auto-generated method stub

int[] keys = { 15, 10, 20, 8, 12, 16, 25 };

Node root = *constructBST*(keys);

*inorder*(root);

}

}

**-----------------------**

**Construct a balanced BST from the given keys (See attached Notepad for more info**

**In the same folder)**

Construct a balanced BST from the given keys

Write a Program to Construct a BST from the given Keys

Given an unsorted integer array that represents binary search tree (BST) keys, construct a height-balanced BST from it.

For each node of a height-balanced tree, the difference between its left and right subtree height is at most 1.

For example,

Input: keys = [15, 10, 20, 8, 12, 16, 25]

Output:

15

/ \

10 20

/ \ / \

8 12 16 25

OR

12

/ \

10 20

/ / \

8 16 25

/

15

