Binary Search Tree (BST) notes

Prof Bill - Feb 2020

The reading:

- Sedgewick Algos 3.2 BST, algs4.cs.princeton.edu/32bst
- Sedgewick Java 4.4 Symbol tables, introcs.cs.princeton.edu/java/44st/
- This animation is great: www.cs.usfca.edu/~galles/visualization/BST.html
- A nice lecture: www.cs.cmu.edu/~adamchik/15-121/lectures/Trees/trees.html

Sections are:

A. Binary trees
B. Binary search trees (BST)
C. BST ADT
D. Terms

thanks...yow, bill
A. Binary Trees

Binary trees have nodes like a linked list. Each node has data (key, value) and then left and right node pointers. The root is the first node in the tree.

![Binary tree diagram]

Source: cppbetterexplained.com/binary-search-trees/

Binary trees are recursive structures because nodes have nodes in them. Also, subtrees behave just like the overall tree. Makes for easy recursive methods.

![Binary tree consisting of 3 binary trees]

Source: www.sqa.org.uk/e-learning/LinkedDS04CD/page_30.htm

Traversal:

- **Preorder**: root, left, right
- **Inorder**: left, root, right  // sorted order in a BST
- **Postorder**: left, right, root
  /* memory helper: 1) root determines pre, in, or post and 2) left always before right */

Pseudocode... start process with call: inorder( root):

```java
// inorder traversal to print binary tree
void inorder( Node n)
if n == null then return
inorder( n.left)
print n
inorder( n.right)
```
B. Binary Search Trees (BST)

A binary tree + this magic...**two rules** for every node:

1. left child is less than (<) node
2. right child is greater than (>) node

That’s it. Let’s build one. An empty BST is root = null (not shown below). Below: the root is the first node added; in this case 31.

![BST Diagram](source: csegeek.com/csegeek/view/tutorials/algorithms/trees/tree_part2.php)

Notice in our example:
- The root doesn’t change when adding to the tree
- Every new node is added as a leaf

Performance for BST magic,

- Average performance is $O(\log n)$, problem cut in half with each subtree
- Worst-case performance is $O(n)$, an unbalanced tree turns into a linked list (dop!)
There are 3 important methods in the BST ADT:

1. put( K key, V value) - we just did this
2. V get( K key)
3. V remove( K key)

With put() - Often, we just show the keys. The value is there or it's just keys (like a set). Use the same left/right algorithm as get() below. New node is always a leaf!

Here's get() pseudocode... it's a recursive search:

```java
get( K key) {
    return getNode( root, key) // start at root
}

V getNode( Node n, K key) {
    if n == null then return null // NOT found
    if key == node.key return node.value // FOUND
    if key < node.key return getNode( node.left, key) // look LEFT
    else return getNode( node.right, key) // look RIGHT
}
```

Remove? Well, it's a little tougher.
Three cases, removing a node with no children (leaf), one child, and two children:

➔ No children (leaf) - null out the parent’s link to the node (easy)

Source: www.techiedelight.com/deletion-from-bst/

➔ One child - replace node with its child (pretty easy)

Source: www.cs.cmu.edu/~adamchik/15-121/lectures/Trees/pix/del01.bmp

➔ Two children - replace node with predecessor (largest node in left subtree, tougher)

Source: www.cs.cmu.edu/~adamchik/15-121/lectures/Trees/pix/del01.bmp

About two children - Successor is ok too, smallest node in right subtree; Pred/Succ are always a leaf or one-child node. Yes?
C. BST ADT

Binary Search Tree (BST) - All operations are average O( log n), worst case O( n).

The worst case performance happens when the BST becomes unbalanced, where one subtree is much larger (and longer) than another.

Methods: put(key, value), value get(key), value min(), value max(), print()

Pseudocode… each public method starts at the root and calls a corresponding private, recursive method that uses BST nodes:

```java
BinarySearchTree
    Node {
        K, V (key, value)
        Node left
        Node right
    }

private Node root;

// two put methods: public BST method and private recursive node method
put(K key, V value)
    n = new BST node
    if root == null
        then root = n
    else
        putNode(root, n)

private putNode(Node n, Node putNode)
    if putNode.key < n.key  // put in LEFT subtree
        if n.left == null
            n.left = putNode
        else
            putNode(n.left, putNode)
    else if putNode.key > n.key  // put in RIGHT subtree
        if n.right == null
            n.right = putNode
        else
            putNode(n.right, putNode)
```
// two get methods; get value for this key if found in BST
V get( K key)
    if root == null return null    // empty
    return getNode( root, key)

private V getNode( Node n, K key)
    if n == null, then return null    // not found
    if n.key == key
        return n.value    // FOUND - return it
    else if key < n.key
        return getNode( n.left, key)    // look LEFT
    else
        return getNode( n.right, key)    // look RIGHT

// two min methods; return the leftmost node, which is min
V min()
    if root == null, then return null
    return minNode( root)

private V minNode( Node n)
    if n.left == null
        return n.value    // no more left nodes, this is MIN
    else
        return minNode( n.left)    // go left again

// print BST keys in sorted order
print()
    printNode( root)

private printNode( Node n)
    if n == null, then return
    printNode( n.left)
    print n.data
    printNode( n.right)

Notes:
- All this recursion is **tail recursion** and can be easily replaced by iteration.
D. Terms

Binary tree and BST terms include:

node
root
parent
child
leaf

get, put, min, print
inorder, preorder, postorder

balanced
expected: $O(\log n)$, worst: $O(n)$

Java: Comparable
Java, TreeMap, www.baeldung.com/java-treemap