Binary Search Tree ADT

Prof Bill - Apr 2018

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:

```
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.