CHAPTER 10.1
BINARY SEARCH TREES

ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH DATA STRUCTURES AND ALGORITHMS IN C++, GOODRICH, TAMASSIA AND MOUNT (WILEY 2004) AND SLIDES FROM NANCY M. AMATO AND JORY DENNY
A binary search tree is a binary tree storing entries \((k, e)\) (i.e., key-value pairs) at its internal nodes and satisfying the following property:

- Let \(u, v,\) and \(w\) be three nodes such that \(u\) is in the left subtree of \(v\) and \(w\) is in the right subtree of \(v\). Then \(\text{key}(u) \leq \text{key}(v) \leq \text{key}(w)\)

- External nodes do not store items

- An inorder traversal of a binary search trees visits the keys in increasing order
To search for a key k, we trace a downward path starting at the root.

The next node visited depends on the outcome of the comparison of k with the key of the current node.

If we reach a leaf, the key is not found.

Example: find(4)

Algorithms for floorEntry( ) and ceilingEntry( ) are similar.

**Algorithm Search(k, v)**

1. if v.isExternal( )
2. return v
3. if k < v.key( )
4. return Search(k, v.left( ))
5. else if k = v.key( )
6. return v
7. else //k > v.key()
8. return Search(k, v.right( ))
EXERCISE

- Show the search paths for the following keys: 8, 3, 2
To perform operation \text{put}(k, v), we search for key \( k \) (using \text{Search}(k))

Assume \( k \) is not already in the tree, and let \( w \) be the leaf reached by the search

We insert \( k \) at node \( w \) and expand \( w \) into an internal node

Example: insert 5
• Insert into an initially empty binary search tree items with the following keys (in this order). Draw the resulting binary search tree
  • 55, 68, 12, 9, 30, 59, 73
DELETION

- To perform operation \( \text{erase}(k) \), we search for key \( k \)
- Assume key \( k \) is in the tree, and let \( v \) be the node storing \( k \)
- If node \( v \) has a leaf child \( w \), we remove \( v \) and \( w \) from the tree with operation \( \text{removeAboveExternal}(w) \), which removes \( w \) and its parent
- Example: remove 4
DELETION (CONT.)

• We consider the case where the key $k$ to be removed is stored at a node $v$ whose children are both internal
  • we find the internal node $w$ that follows $v$ in an inorder traversal
  • we copy $w$.key() into node $v$
  • we remove node $w$ and its left child $z$ (which must be a leaf) by means of operation removeAboveExternal($z$)

• Example: remove 3
• Insert into an initially empty binary search tree items with the following keys (in this order). Draw the resulting binary search tree
  • 30, 40, 24, 58, 48, 26, 11, 13
• Now, remove the item with key 30. Draw the resulting tree
• Now remove the item with key 48. Draw the resulting tree.
Consider an ordered map with $n$ items implemented by means of a binary search tree of height $h$

- Space used is $O(n)$
- Methods `find(k)`, `floorEntry(k)`, `ceilingEntry(k)`, `put(k, v)`, and `erase(k)` take $O(h)$ time
- The height $h$ is $O(n)$ in the worst case and $O(\log n)$ in the best case