CSC263 *Data Structures and Analysis* January 21, 2015

Lecture 06

Dictionaries - *Continued*

Direct Mapping

NL	2	NL	NL	5	...	100

Search,delete,insert $\in \Theta(1)$, but space is an issue! space $\in O(n)$ where n is the size of the keyspace

Hashing

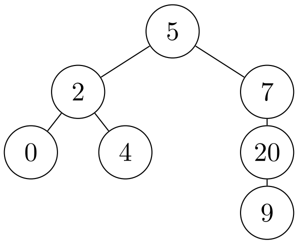
9	4	7	5	5	2	**8**	**5**	**6**	**7**

9	7	9	8	6	5	**8**	**5**	**6**	**7**

Binary Search Trees

- for every node, all items in left subtree $\leq$ node.item $\leq$ all items in right subtree
- dictionary S stores only $S.root$ (also $S.size$ usually)
- `TreeNode` has members `.item` (element stored in node), `.left` and `.right` (children)

5	7	2	0	20	4	9

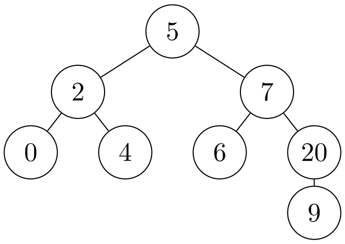


```
search(S,k):
  return treeSearch(S.root,k)

treeSearch(root, k):
  if root == NULL:
    # return null
    pass
  else if k < root.item.key:
    root = treeSearch(root.left, k)
  else if k > root.item.key:
    root = treeSearch(root.right, k)
  else:
    # k is root.item.key
    pass

  return root
```

insert 6:



```
insert(S,x):
  treeInsert(S.root, x)

treeInsert(root, x):
  if root == NULL:
    root = TreeNode(x)
  else if x.key < root.item.key:
    root.left = treeInsert(root.left, x)
  else if x.key > root.item.key:
    root.right = treeInsert(root.right, x)
  else
    # k is root.item.key

  return root
```

delete

If x is not found → do nothing
If x is found in a leaf → remove the leaf
If x has only one sub-tree, remove x and shift the subtree up to replace x (splice around x)
If x has two sub-trees:

- it must have a *successor* (the next node in the tree larger than x)
- the *successor* must be the minimum node in the right subtree of x
- the *minimum* is the left-most node in a tree

```
delete(S,x):
  S. root = treeDelete(S.root, x)

treeDelete(root, x):
  if root == NULL: # x.key not in $S \rightarrow$ should not happen!
    pass # nothing to remove
  else if x.key < root.item.key:
    root.left = treeDelete(root.left, x)
  else if x.key > root.item.key:
    root.right = treeDelete(root.right, x)
  else: # x.key == root.item.key
    # remove root.item
    if root.left == NULL or root.right == NULL:
      # root missing one child, replace with other child
      if root.left == NULL:
        root = root.right # NULL if both children are missing
      else:
        root = root.left
    else:
      # Root has two children: remove element w/ smallest key in
      # right subtree and move it to root
      return root.item, root.right = treeRemoveMini(root.right)
  return root

treeRemoveMini(root):
  # remove element w/ smallest key in root's subtree
  # return item and root of resulting subtree
  if root.left == NULL:
    # root stores item w/ smallest key; replace it w/ right child
    return root.item, root.right
  else:
    # left subtree not empty: root not the smallest
    item, root.left = treeRemoveMini(root.left)
    return item, root
```