CS 313
- Kangmei Yang
Binary Search Tree

- It’s a binary Tree
- Elements in tree are Comparable
- Apply to any subtree:
  - All elements in left subtree are smaller than the root element
  - All elements in right subtree are greater than the root element
- No duplicate values

This property allows us to efficiently search an element.
Sample BST

```
    10
   /  
  7    13
 / 
4  12
  
5  15
```
Search in BST

- Start comparing the key to the root element
- If it’s the value (key == root’s element)
  - We found it
- If key is smaller (key < root’s element)
  - Search left subtree
- If key is greater (key > root’s element)
  - Search right subtree
- Repeat this until we find it in the tree, or we have no more value in the range (reach null)
- Find 8

- Not in tree

Search BST

8 < 10

8 > 7

null
Search BST

- Find 12

- Found it in tree
Add to BST

- Insertion is similar to searching.
- Search the element in the tree
- If found it, do not add (no duplicates)
- If not found by reach a null reference
  - Replace with a new node

Think, how do we set the reference?
Add to BST

- Add 8

Not in tree
- Create a new node with value 8
- Go back to set the reference
Remove from BST

- Deletion first perform searching.
- If not in the tree, do nothing
- If found the node contains the element we want to delete, there’re 3 cases.
  - The node has no children: remove by update parent’s reference to null
  - The node has only 1 child: remove by update parent’s reference, replace itself by it’s child
  - The node has 2 children: replace itself with it’s successor (smallest value in it’s right subtree), then remove the successor

Think why is the successor?
Remove from BST

- Remove 12

- Found it in tree
- Has no children
- Go back to it’s parent
- Update it’s parent’s right child reference to null
Remove from BST

- Remove 11
- Found it in tree
- Has 1 child
- Go back to it’s parent with it’s child reference
- Update it’s parent’s left child reference to it’s child
Remove from BST

- Remove 10

- Found it in tree
- Replace with it’s successor
- Remove it’s successor from right subtree
- Has 1 child
- Go back to it’s parent with it’s child reference
- Update it’s parent’s left child reference to it’s child
BST Runtime Analysis

- Add, Remove, and Search, in most case traverse down to a leaf node, and the longest path to a leaf node is the height of the tree

- The runtime of each method is proportional to the height of the tree

- To ensure an optimal runtime = ensure the height doesn’t grow too large
  - self-balancing tree