CS232 - Homework 6: Assigned: November 4, 2016
Due: before class November 18, 2016 (2 weeks! Nov 11 is a holiday)

This assignment uses the code for Binary Search Trees that we developed in class (with slightly different names), and includes a new driver.

The jGrasp debugger and it's ability to draw trees will be your friend on this assignment.

**Part 1: Traversal**
Write a method that does a level-order traversal of the tree. Pseudocode:

```
Create a Queue (or an ArrayList, but use it like a queue)
Enqueue the root
while the queue is not empty:
     enqueue the left and right children of the head of the list
     (if they exist)
     dequeue and print the head of the list
```

Use the driver to make sure this prints the correct results.

**Part 2: Search**
Create a method that checks to see if an element is already in the tree and returns null if the element is not found, or returns the BSTNode that contains the element. Write test statements in the driver that look for both elements that are in the tree and elements that aren't in the tree and print the results.

**Part 3: Removing nodes**
The following pages contain descriptions of how to remove nodes from a Binary Search tree. They break the problem down into 3 different cases: when the node being removed has no children, when it has only 1 child, and when it has 2 children.

For this assignment, implement the first two cases. Write code in the driver that calls the remove function only on nodes that you know have one or two children. Make sure that you remove multiple elements so that you can each combination of cases and the removed node being the left- or right- child of the parent. More specifically:

- Case 1 (no children):
  - Test 1: node being removed is left child of parent
  - Test 2: node being removed is right child of parent
- Case 2 (one child):
  - Test 1: node being removed is left child of parent and has one left child
  - Test 2: node being removed is left child of parent and has one right child
  - Test 3: node being removed is right child of parent and has one left child
  - Test 4: node being removed is right child of parent and has one right child

**Extra credit**
Implement the third removal case (the node with two children)

Modified from http://www.algolist.net/Data_structures/Binary_search_tree/Removal

# Binary search tree: Removing a node

Remove operation on binary search tree is more complicated than add and search. Basically, in can be divided into two stages:

- search for a node to remove;
- if the node is found, run remove algorithm.

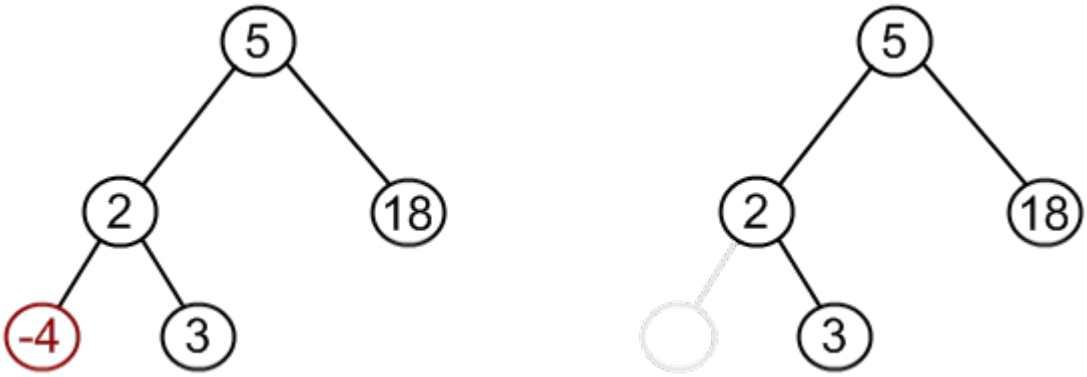
## Remove algorithm in detail

Now, let's see more detailed description of a remove algorithm. First stage is identical to the search function. Second part is more tricky. There are three cases, which are described below.

1. Node to be removed has no children.

   This case is quite simple. First, find the node you are searching for (we'll call it b). Second, find the parent of b (call it p). Call b.setParent(null), determine whether b is p's left or right child. If it is the left child, call p.setLeft(null); if it is the right call p.setRight(null). This removes the node from the tree

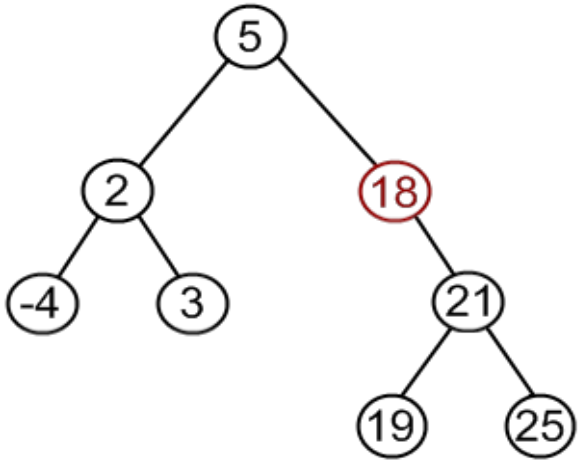
   **Example.** Remove -4 from a BST.

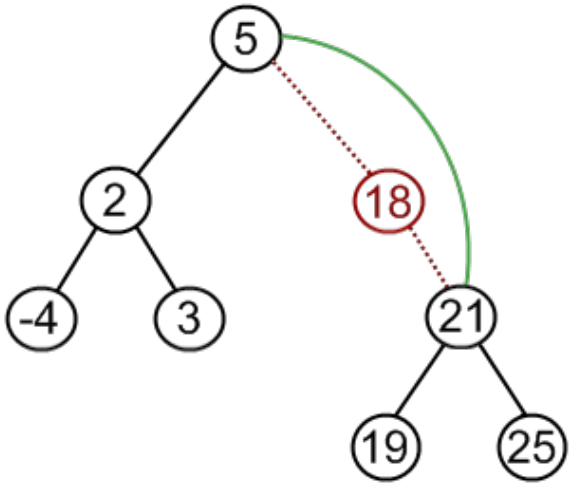


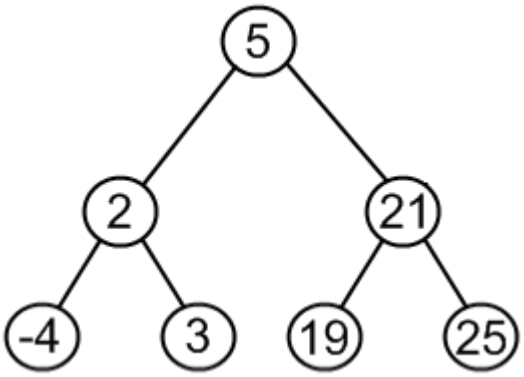
2. Node to be removed has one child.

   It this case, node is cut from the tree and the algorithm links the single child (with it's subtree) directly to the parent of the removed node, and sets the appropriate child pointer for the parent and parent pointer for the new child.

   **Example.** Remove 18 from a BST.





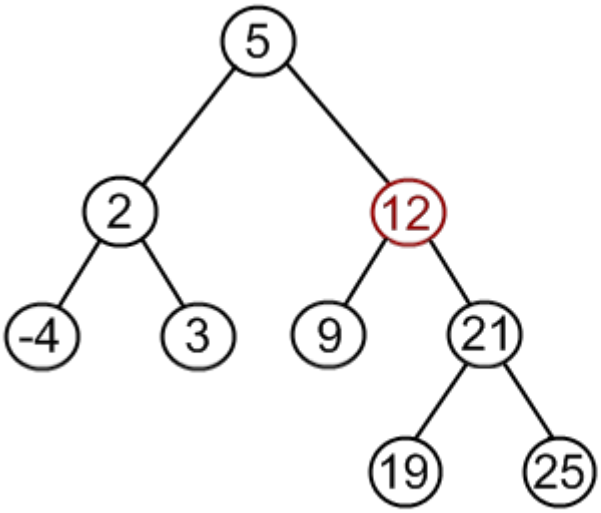

3. Node to be removed has two children.

   This appears to be the most complex case, but the actual implementation is relatively simple:

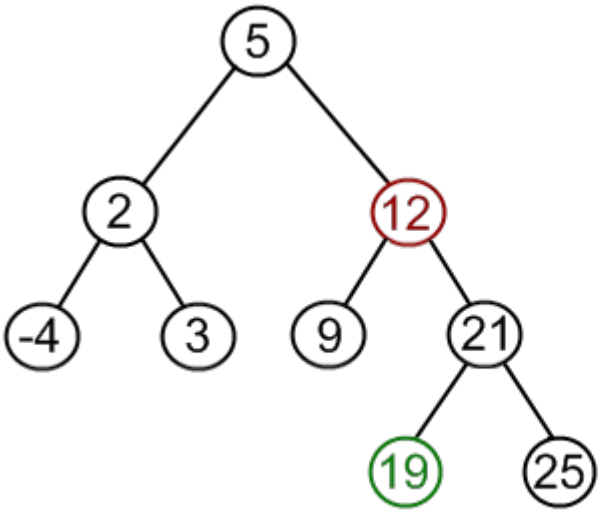
   - find a minimum value in the right subtree;
   - save the minimum value in a placeholder
   - call the remove function on the minimum value
   - put the minimum value in the node where the removed value was being stored.

   Note: the node with minimum value has no left child and, therefore, it's removal may result in first or second cases only.

   **Example.** Remove 12 from a BST.




   Find minimum element in the right subtree of the node to be removed. In current example it is 19.




Create an element that stores the value 19. Then, call remove again, this tine removing 19 from the tree, and then put the value 19 in the node where the 12 was: