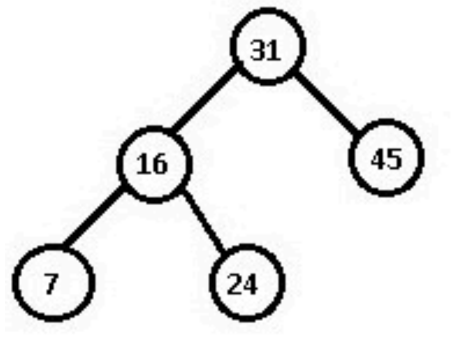
Lecture 6: Mutating Recursive Data Structures

Binary Search Trees:

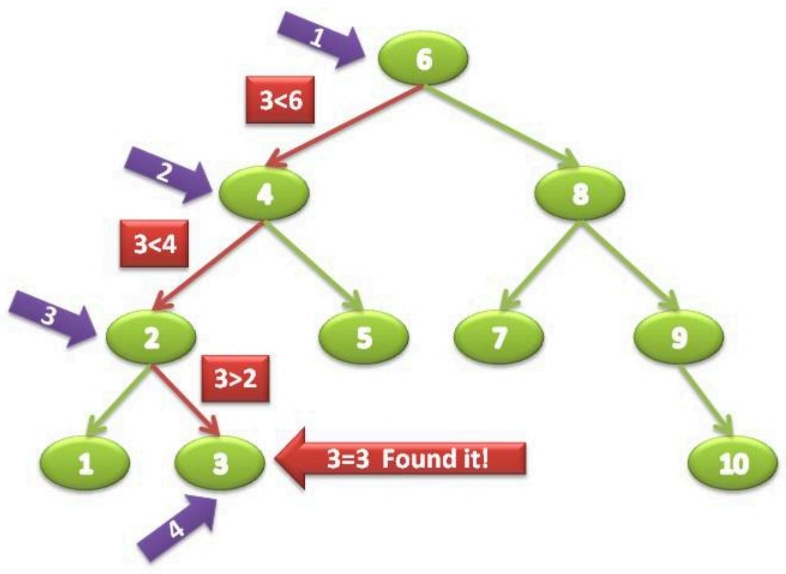
Binary Search Trees (BSTs) allow for efficient binary search like Python lists, but also allow for efficient insert. Search and insert each take O(h) time, where h is the depth of the BST. Typically, h grows as log n, though guaranteeing this is not easy and will be a notion left for 6.006.

A binary search tree is a rooted binary tree, whose internal nodes correspond to keys with associated numbers and each key has two distinguished sub-trees, commonly denoted left and right. The tree additionally satisfies the *binary search tree property*, which states that the key in **each** node must be greater than all keys stored in the left sub-tree, and smaller than all keys in right sub-tree. The leaves (final nodes) of the tree contain no key and have no structure to distinguish them from one another. Leaves are commonly represented by a special leaf symbol, None. An example is shown below – the leaves are not shown.

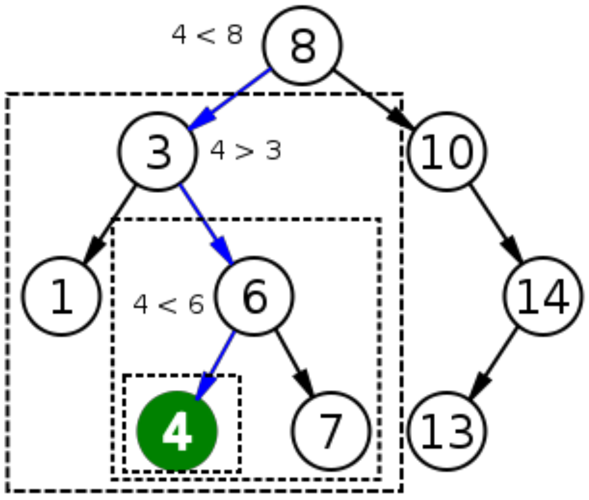


We will describe search, insert and delete pictorially below. Below we are searching the BST for an element with number 3.[1] The purple arrows indicate the steps of the algorithm.

[1] From Encrypt3D.

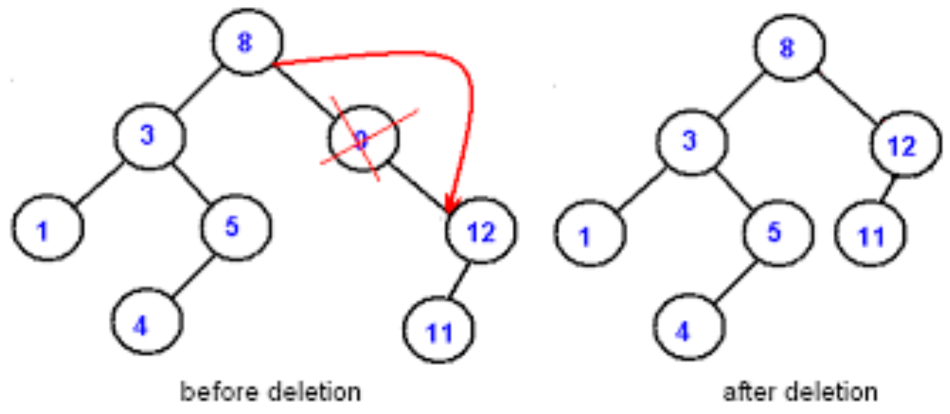


Below we show how a new node with number 4 is inserted into the BST.[2] The node is inserted based on the number associated with the node, since we have to maintain the BST property. We search for 4 and fail to find it, but in the process we figure out where to insert it.



Finally, here is an example of deleting from a BST.[3] Deletion is significantly more complicated than insert, and we show an example of deleting a node with a single child below.

[2] From Codeward Bound.
[3] From Victor Adamchik.



If a node has two children and needs to be deleted, the process is more involved. Below is a nice description of various cases in delete.[4]

Binary search tree: Deleting a node

The delete operation on binary search tree is more complicated, than insert and search. It can be divided into two stages:

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

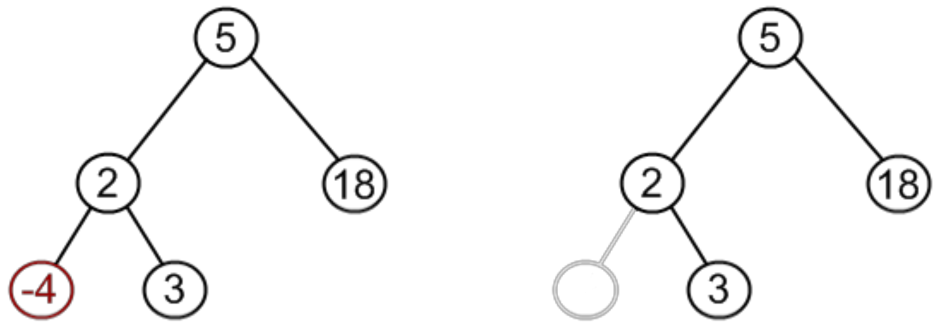
Here's a more detailed description of a delete algorithm. The first stage is identical to the algorithm for search except we should track the parent of the current node. The second part is trickier. There are three cases, which are described below.

1. Node to be deleted has no children.

   This case is quite simple. Algorithm sets corresponding link of the parent to NULL and disposes the node.

   Example. Delete -4 from a BST.

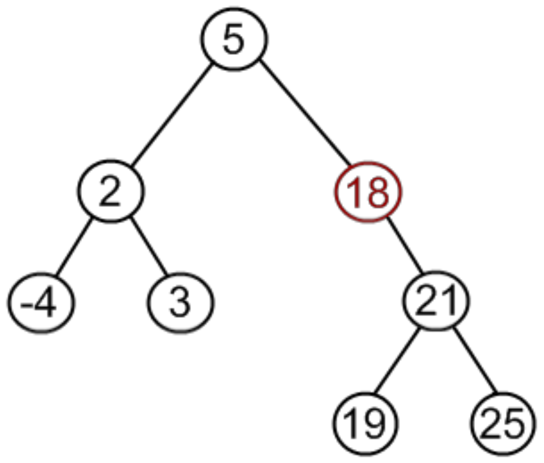
[4] http://www.algolist.net/Data_structures/Binary_search_tree/Removal

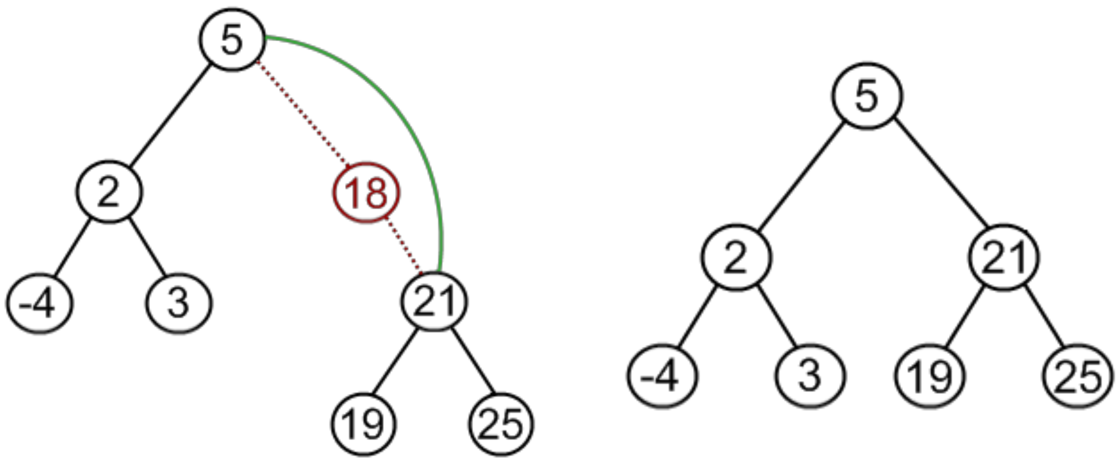


2. Node to be deleted has one child.

   It this case, node is cut from the tree and algorithm links single child (with it's subtree) directly to the parent of the deleted node.

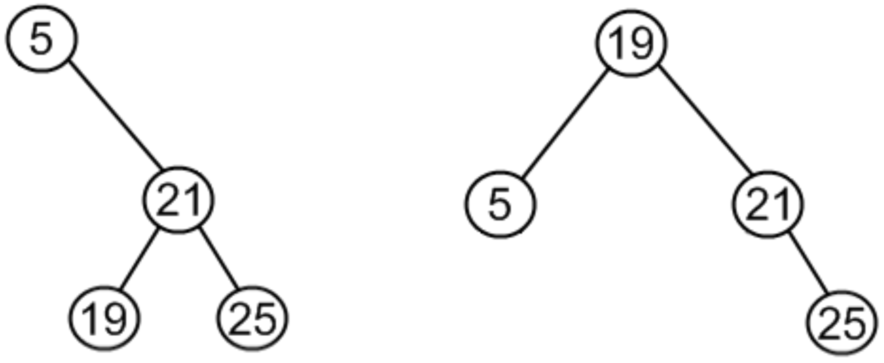
   Example. Delete 18 from a BST.





3. Node to be deleted has two children.

This is the most complex case. To solve it, let us see one useful BST property first. We are going to use the idea, that the same set of values may be represented as different binary-search trees. For example the BSTs below:



contain the same values {5, 19, 21, 25}. To transform first tree into second one, we can do following:

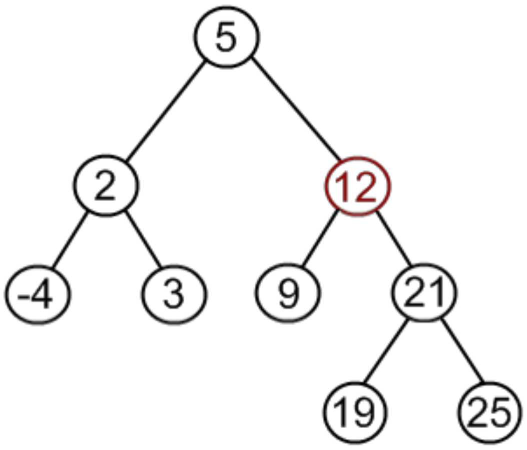
- choose minimum element from the right subtree (19 in the example);
- replace 5 by 19;
- hang 5 as a left child or otherwise insert it in the left subtree (this will always work since the original root element is less than the minimum of the right subtree)

The same approach can be utilized to delete a node, which has two children:

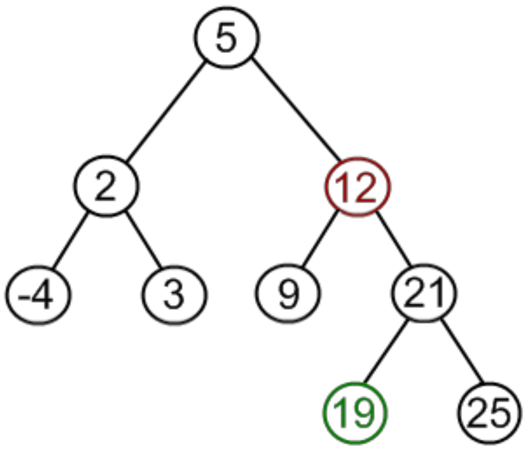
- find a minimum value in the right subtree, **which has to be a leaf**;
- replace value of the node to be deleted with found minimum. Now, the right subtree contains a duplicate!
- apply delete to the right subtree to delete a duplicate.

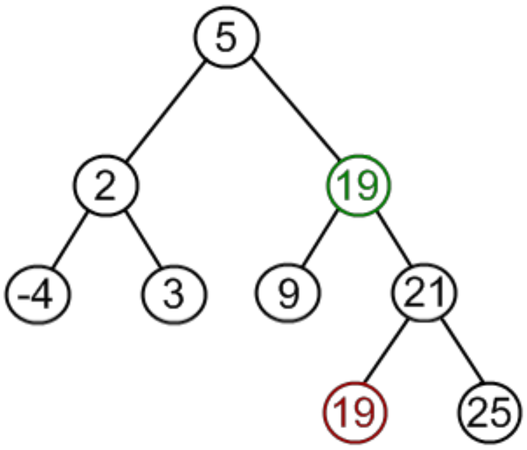
Notice, that the node with minimum value has no left child and, therefore, its removal may result in first or second cases only.

Example. Delete 12 from a BST.

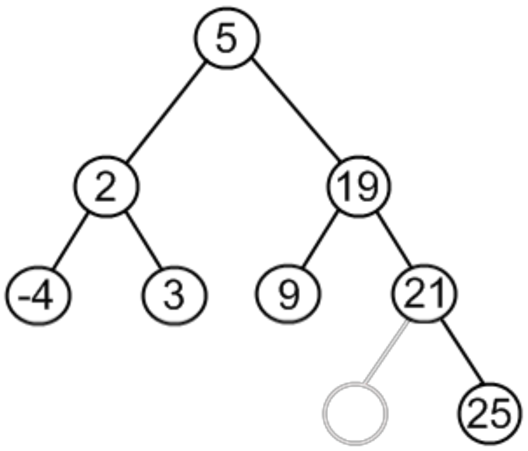


Find minimum element in the right subtree of the node to be deleted. In current example it is 19. Then, replace 12 with 19. Notice, that only values are replaced, not nodes. Now we have two nodes with the same value.





Now delete 19 from the subtree.



Code for BST search, insert and delete, can be found in **Lecture6.py**.

Iterators and Generators

A brief tutorial on `yield` and `yield from`:
http://simeonvisser.com/posts/python-3-using-yield-from-in-generators-part-1.html.