

**Reading:** GT 3.1, pp. 141–151 [up to 60 minutes]

The stuff about binary search is just a reminder from the first semester programming course. The core stuff is sections 3.1.3–1.1.5.

**Ingredients:** Binary Search, BSTs, search, insertion, removal

BST is a data structure implementing a dictionary—an alternative to hash-tables with guaranteed worst case performance.

## Binary Search Trees

**Definition 1.** A binary search tree (BST) is a binary tree that has a key associated with each of its nodes, such that all its nodes maintain the BST invariant:

1. The key in the node is larger than keys of all the nodes in the left subtree.
2. The key in the node is smaller than keys of all nodes in the right subtree.

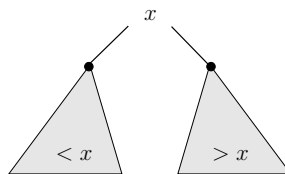


Figure 1: The BST invariant.

For clarity we assume no duplicates in the tree in this lecture.

**Question.** How do we search for an element with a given key in a BST?

To search in a BST descent into the proper subtrees using key values of nodes to navigate, until you reach a leaf or you find the node:

**Question.** What is the running time of search? What is the worst-case?

Linear in the size of the tree in worst case, for example when the tree is a list and we search for a leaf.

**Insert** To insert a node with key  $t$  first search for  $t$ , then insert it at the node where search has terminated.

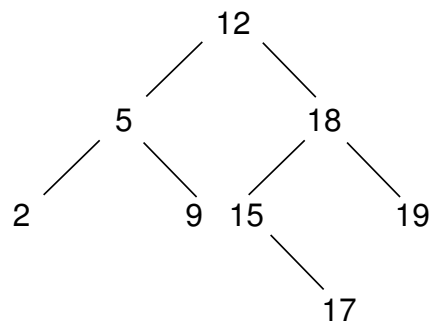


Figure 2: An example of a BST (binary search tree).

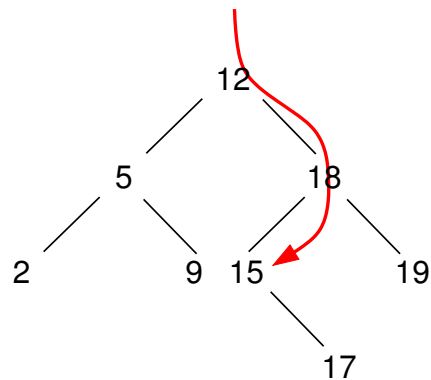


Figure 3: Searching for 15.

**Delete** To delete an element we consider several cases. If the node to delete is childless, then we simply remove it once it is found:

If the deleted node has only one child then we simply “splice it out”:

If the node has two children then splice-out its successor and place it in place of the deleted node:

**Question.** How do we compute the successor?

For a node  $n$  with two children the successor of  $n$  is the minimal element of  $n$ 's right subtree.

Follow the link right down once and then left down as long as possible. Successor is the last element you can reach. This is the only case that can occur in deletion.

**Question.** What is the cost of computing the successor?  $O(h)$

**Question.** What is the total cost of deletion?  $O(h)$ , so  $O(n)$  in the worst case.

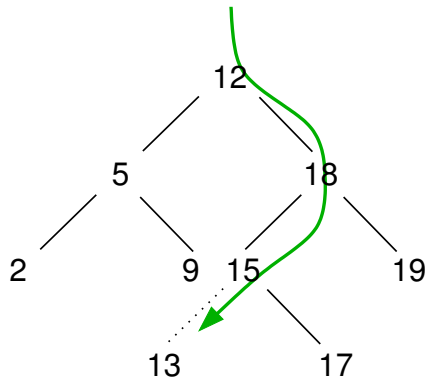


Figure 4: Inserting 13.

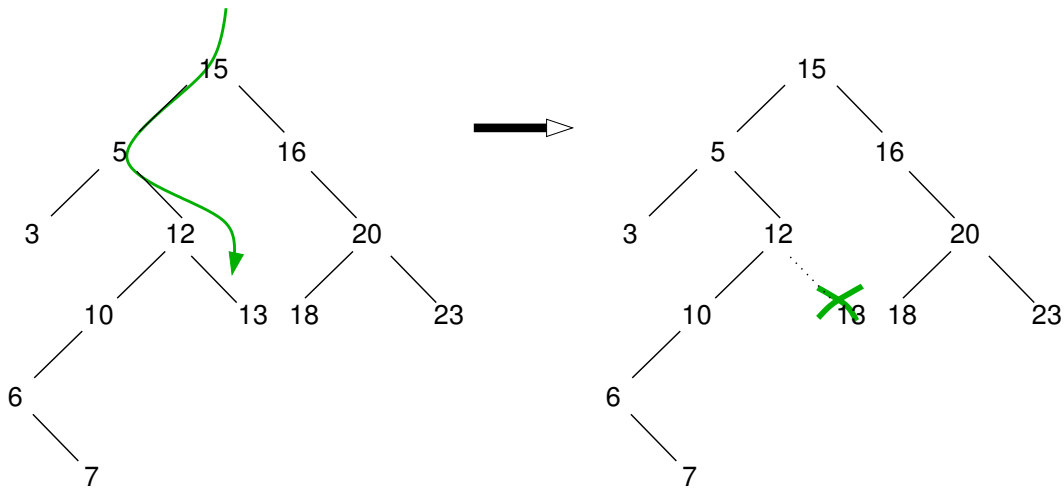


Figure 5: Deleting 13.

**Important** In the best case, when the tree is perfectly balanced, all  $O(h)$  operations run in  $O(\lg n)$  time!

**Problem** Worst-case is relatively likely (sorted input, presorted input, reversely sorted input, etc).

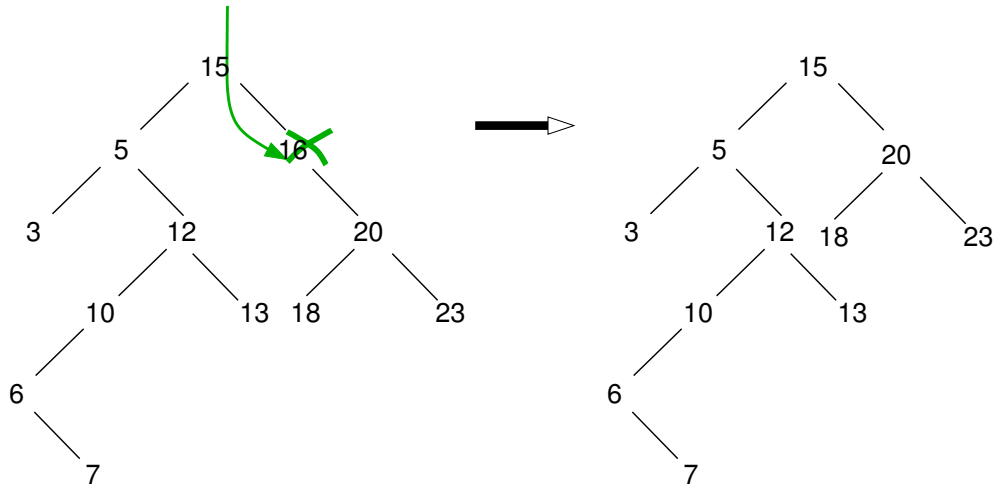


Figure 6: Deleting 16 (same tree as above).

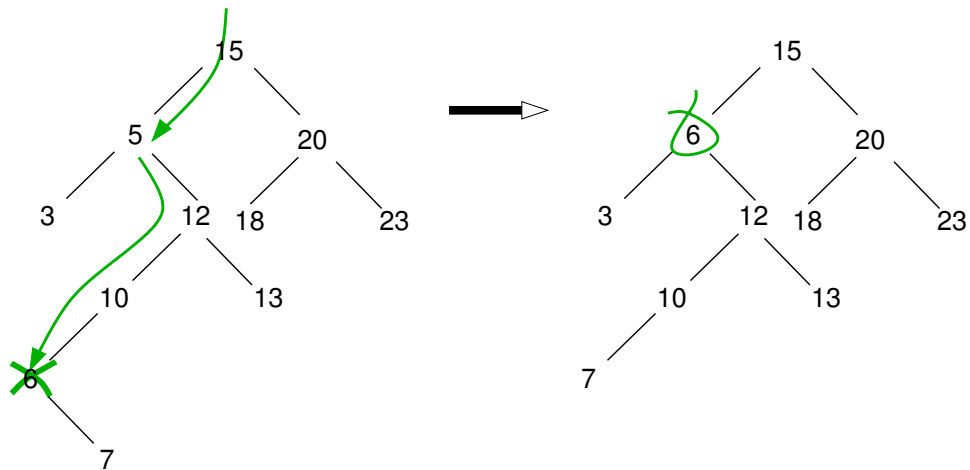


Figure 7: Deleting 5 from the last result.