Data Structures and Algorithms I

Thu 16.2.2017

Exercise 4

No mandatory X-task this week, but there will be a binary tree X-task on next week. Use *BinTreeExample17.java* as a main program/skeleton. Draw a picture of all exercises.

19. Write an algorithm that inserts an element to an in-ordered binary tree so that the tree remains in in-order. If there was the same element already in the tree, no insert will be made and algorithm returns false. If insert is made, the algorithm creates a new node, places it to the correct position in the tree and returns true. Parameters are a tree and an element. The idea of the algorithm was presented at lectures. What is the time complexity of your algorithm?

20. Write an algorithm that calculates the height if a given binary tree. I.e., the longest path from root node to a leaf node. Hint: recursion. What is the time complexity of your algorithm?

21. Write an algorithm that compares two binary trees and returns true if the *structures* of the trees are the same. Hint: recursion on two tree "in tandem". Trees/nodes are the same if both subtrees are the same. Ignore the elements, just compare existence of nodes ("shape" of the tree). What is the time complexity of your algorithm?

22. Write algorithm (operation) *inorderFirst*() that returns the first node in in-order of the given binary tree. Do not check elements, just find the leftmost node of the tree. If there is no first node, return *null*. What is the time complexity of your algorithm?

23. Write algorithm (operation) *inorderNext*() that returns the successor in in-order of the given binary tree node. Do not check elements, just find the next node within the tree. If there is no next node, return *null*. If node n has a right child, the successor is the leftmost descendant of the right child. Otherwise, the successor the closest ancestor in whose left sub-tree the node n is. Using this algorithm and 22, we can traverse the whole binary tree in in-order. What is the time complexity of your algorithm? What is the time complexity of the whole traversal?.

24. Simulate manually the function of an AVL-tree. Insert to an initially empty AVL tree elements 8, 17, 24, 10, 12, 20, 15, 1, 2, 5, 6, 3. Draw the tree again after each such insert that required a rotation. Mark the balance ($--$,$-$,0,+,++) of each node in all steps. Fix the balances from bottom up depending on which child the insert was done. Propagation up of balance changes stops when balance changes to 0, or when rotation is done. After inserting element 5, there will be a rotation to right and root is changed.