CSC265 Fall 2021 Homework Assignment 2

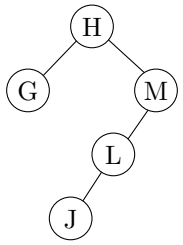
due Wednesday September 29, 2021

Suppose that when inserting a node x into a binary search tree, instead of inserting x as a new leaf of the tree, you would like to insert it as the the new root of the tree. This could be useful if you expect to search for nodes soon after you have inserted them.

A bad way to do this is to simply make the old root of the binary search tree the left or right child of the node you are inserting. The problem is that inserting any sequence of n nodes using this approach results in a binary search tree of height $n - 1$.

A better approach is to follow the search path for the node x, as in the usual insertion algorithm, splitting it into a path P containing each node with key less than $x.key$ and a path Q containing each node with key greater than $x.key$. Then make the first node of P the left child of x and the first node of Q the right child of x, and make x the root of the tree.

1. Draw the binary search tree that results from using an algorithm based on the better approach to insert a node with key K into the following binary search tree.



2. Write pseudocode for an insertion algorithm for binary search trees without parent pointers based on the better approach. You may assume, as preconditions, that the nodes in the binary search tree have distinct keys and the node that is being inserted does not have the same key as any of these nodes.

3. Prove that the tree resulting from performing any insertion using your algorithm in part 2 is a binary search tree containing the nodes that were in the binary search tree before the insertion plus the newly inserted node.

4. By how much can the height of a binary search tree increase as a result of the insertion of one node using your algorithm in part 2? Give matching upper and lower bounds on the maximum height increase. Justify your answer.

5. Give a high level description of an algorithm for deleting a node from a binary search tree without parent pointers by combining two binary search trees. Your algorithm must have the property that if you delete the root of any binary search tree T and then immediately re-insert it using your algorithm in part 2, you get the same tree T back.

6. Write pseudocode for your algorithm in part 5.

7. Explain why the tree resulting from performing any deletion using your algorithm in part 6 is a binary search tree containing the nodes that were in the tree before the deletion, except for the deleted node.

8. Suppose you delete the root of any binary search tree T using your algorithm in part 6 and then immediately re-insert it using your algorithm in part 2. Explain why you get the same tree T back.