ShanghaiTech University

CS101 Algorithms and Data Structures

Fall 2019

Homework 5

Due date: 23:59, October 27, 2019

1. Please write your solutions in English.
2. Submit your solutions to gradescope.com.
3. Set your FULL Name to your Chinese name and your STUDENT ID correctly in Account Settings.
4. If you want to submit a handwritten version, scan it clearly. Camscanner is recommended.
5. When submitting, match your solutions to the according problem numbers correctly.
6. No late submission will be accepted.
7. Violations to any of above may result in zero score.

1 (5') Binary Search Tree and AVL Tree

Each question has one or more correct answer(s). Select all the correct answer(s). For each question, you get 0 point if you select one or more wrong answers, but you get 0.5 point if you select a non-empty subset of the correct answers.
Note that you should write you answers of section 1 in the table below.

Question 1	Question 2	Question 3	Question 4	Question 5
C	C	A C D	A B	A C

Question 1. *Consider the BST created by inserting the following keys in the given order in an initially empty BST: 1 5 7 2 3 6 4(ordered by value). Which other sequence(s), if any, produce the same BST?*

(A) 3 1 2 5 7 4 6

(B) 1 6 5 7 4 2 3

(C) 1 5 2 7 3 4 6

(D) 1 5 6 7 2 4 3

Question 2. *Which of the followings are true?*

(A) For a min-heap inorder traversal gives the elements in ascending order.

(B) For a min-heap preorder traversal gives the elements in ascending order.

(C) For a BST inorder traversal gives the elements in ascending order.

(D) For a BST preorder traversal gives the elements in ascending order.

Question 3. *Which of the following statements are true for an AVL-tree?*

(A) Inserting an item can unbalance non-consecutive nodes on the path from the root to the inserted item before the restructuring.

(B) Inserting an item can cause at most one node imbalanced before the restructuring.

(C) Removing an item in leaf nodes can cause at most one node imbalanced before the restructuring.

(D) Only at most one node-restructuring has to be performed after inserting an item.

Question 4. *Consider an AVL tree whose height is h, which of the following are true?*

(A) this tree contains $\Omega(\alpha^h)$ keys, where $\alpha = \dfrac{1+\sqrt{5}}{2}$.

(B) this tree contains $\theta(2^h)$ keys.

(C) this tree contains $O(h)$ keys in the worst case.

(D) none of the above.

Question 5. *Which of the following is TRUE?*

(A) The cost of searching an AVL tree is $\theta(\log n)$ but that of a binary search tree is $O(n)$

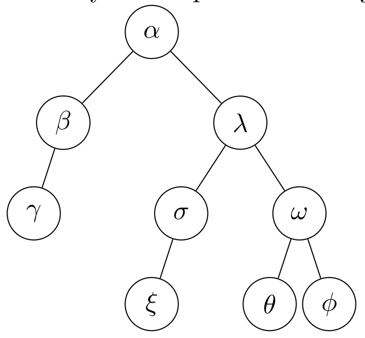
(B) The cost of searching an AVL tree is $\theta(\log n)$ but that of a complete binary tree is $\theta(n \log n)$

(C) The cost of searching a binary search tree with height h is $O(h)$ but that of an AVL tree is $\theta(\log n)$

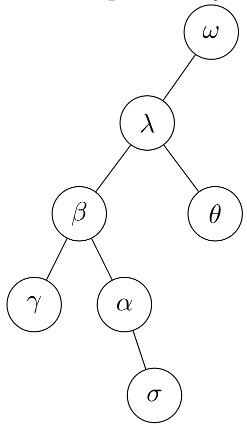
(D) The cost of searching an AVL tree is $\theta(nlogn)$ but that of a binary search tree is $O(n)$

2 (6') Magic BST

Consider the binary search tree below. Each symbol represents an object stored in the BST.



Question 6. *(3') Based on the ordering given by the tree above, fill in the BST below with valid symbols. Symbols must be unique. You may only use the 7 printed symbols (do not include any symbols from part b).*



Question 7. *(3') For each of the insertion operations below, use the information given to "insert" the element into the* ***TOP TREE WITH PRINTED SYMBOLS, NOT THE TREE WITH YOUR HANDWRITTEN SYMBOLS*** *by drawing the object (and any needed links) onto the tree. You can assume the objects are inserted in the order shown below. You should not change anything about the original tree; you should only add links and nodes for the new objects. If there is not enough information to determine where the object should be inserted into the tree, circle "not enough information". If there is enough information, circle "drawn in the tree above" and* ***draw in the tree AT THE TOP OF THE PAGE.***

insert(ϕ):	$\phi > \omega$	[✓ **Draw In Tree Above**]	Not enough Information
insert(χ):	$\chi > \gamma$	Draw In Tree Above	[✓ **Not enough Information**]
insert(ξ):	$\alpha < \xi < \sigma$	[✓ **Draw In Tree Above**]	Not enough Information
insert(μ):	$\lambda < \mu < \omega$	Draw In Tree Above	[✓ **Not enough Information**]

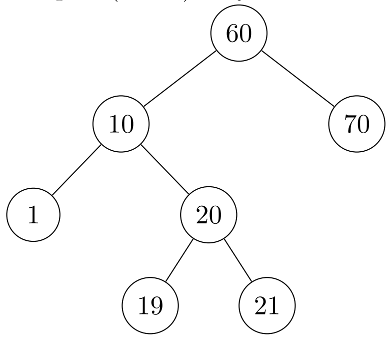
3 (2')Property of BST

Question 8. *(2') Teaching assistant Keyi thinks that he has discovered a critical property of the binary search tree. If you want to find a key in this tree, you will get a search path from the root to that key. Then he defines three sets of all the nodes in this tree. The first one A: the nodes are on the left to the searched node and not on the search path. The second one B: the nodes are on the search path. The third one C: the nodes are on the right to the searched node and not on the search path. Then he claims that any three keys: $a \in A$, $b \in B$ and $c \in C$ must satisfied $a \leq b \leq c$. Do you think his claim is right? If true, prove it. If not, give a counterexample.*

Solution:
It is false. Suppose the searched node is named *key*, and its parent node is named *parent*. If it has a left child, the tree rooted by its left child is named *left sub-tree*, and with the similar definition we have a *right sub-tree*.

1. The nodes on the left of *key*(set A) consist of two parts:
   (a) *key*'s *left sub-tree*: It is clear that all the keys of the nodes of the sub-tree is smaller than *key*'s key.
   (b) Some ancestors of *key*'s left sub-tree(those ancestors' right sub-tree contains *key*): For *key* is in the right sub-tree of those nodes, the key of the nodes in their left sub-trees is smaller than the key of *key*.
2. The nodes on the right of *key*(set C) consist of two parts:
   (a) *key*'s *right sub-tree*: It is clear that all the keys of the nodes of the sub-tree is larger than *key*'s key.
   (b) Some ancestors of *key*'s right sub-tree(those ancestors' left sub-tree contains *key*): For *key* is in the left sub-tree of those nodes, the key of the nodes in their right sub-trees is larger than the key of *key*.

However, the other nodes in the search path(set B) may not have the same situation with *key*.



For example, in the binary tree above: Suppose the searched key node is 21.
A = {1,19}; B = {21,20,10,60}; C = {70}.
$19 \in A$; $10 \in B$, while $19 > 10$.

4 (9')Only-child

We define that the node is only-child if its parent node only have one children(Note: The root does not qualify as an only child). And we define a function of the binary tree: $OC(T)$ =the number of only-child node.

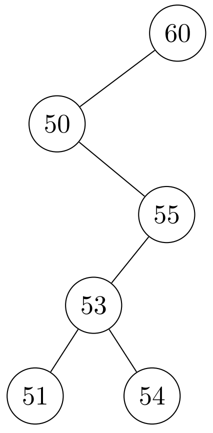
Question 9. *(3')Prove that for any nonempty AVL tree T with n nodes, we have that OC(T)*$\leq \frac{1}{2}n$

Solution: Suppose that the number of only-child node is m. Every only child has a parent and only-child has no siblings. Then the total number of nodes now is 2m. If the parent of the only-child is also an only-child, the parent's parent would has a sub-tree with height 2 and another with height 0, meaning that the balance of the AVL tree would be broken. Therefore, the parent must has a sibling. Then we can conclude that the total number of nodes is more than 2m.

$$\Longrightarrow 2m \leq n$$
$$m \leq \frac{1}{2}n$$
$$\Longrightarrow OC(T) \leq \frac{1}{2}n$$

Question 10. *(3')Is it true for any binary tree T with n nodes, that if $OC(T) \leq \frac{1}{2}n$ then $height(T) = O(\log n)$? If true, prove it. If not, give a counterexample.*

Solution: It is false.



The binary tree above has 6 nodes. The node 50, 53 and 55 are only-child nodes, then OC(T) = 3. OC(T) $\leq \frac{1}{2}n$. However, height(T)=4 and $O(\log n) = 3$.

Question 11. *(3') Is it true for any binary tree T, that if there are n node which is only-child, all of which are leaves, then $height(T) = O(\log n)$? If true, prove it. If not, give a counterexample.*

Solution: It is true.
Since all the leaves are only-child nodes and there are no other only-child nodes, the parent of these only-child nodes have siblings. In other words, the only-child nodes' parent's parent must have two children. Take off all the leaves, it would be a full tree. In this full binary tree, it has n leaves which equals the number of only-child nodes.

$$f(n) = 1 + f(n-1)$$

where

$$f(2) = 1$$

Then it has $n-1$ internal nodes. The total number of the nodes of the original tree is $n+n+n-1 = 3n-1$. Then we can get:

$$\Longrightarrow height(T) = O(\log n)$$