
Rotations: [CLRS p 277-279] for pseudocode.

Determine if each of the following is 1) a valid BST 2) maintains the AVL property and label the nodes w/ their height, indicate where AVL violation is invariant.

Node deletion in a BST - no need to balance!

2  
A delete 11, del 17  
B delete 3  
C delete 10
Reminders
- notes + code posted & will be in future
- Pset 1: idea for timing, timeit
- run test code!

Warmup Answers

<table>
<thead>
<tr>
<th></th>
<th>BST</th>
<th>AVL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. A.</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>B.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>C.</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>D.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>F.</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

AVL trees

idea: balance trees are $O(\log n)$, pay extra to keep balanced to maintain
good order of growth running time for operations.

force: height(left child) and height(right child) to differ by
at most 1

how? perform BST operations then rebalance trees through
rotations.

only 2 rotations needed to rebalance tree!
1) Write a method to perform left (or right) rotation on a node \( x \).
   - You may assume \( x \) is a python object with fields \( x.left \) and \( x.right \).
   - Return the new root to the subtree previously rooted at \( x \).

2) Write a recursive function to determine the height (as defined in class) of a given node.

3) If I replaced the subtree rooted at \( x \) with another arbitrary AVL subtree, which nodes would potentially break the AVL invariant?

4) When the AVL invariant is broken, why do we correct it starting at the deepest node that violates the invariant rather than working from the root down?
Exercise 1:  
right rot.  

```python
def right_rot(x):
    y = x.left
    A = y.left
    B = y.right
    C = x.left
    y.right = x
    x.left = B

    return y
```

: error checking omitted if similar for left:

Exercise 2:

```python
def height(x):
    if leaf(x): return 0
    return max(height(x.right), height(x.left)) + 1
```

ex3:  
all parents of x may break the invariant

ex4:  
it takes fewer rotations to fix the tree from the bottom up.
we will need to do at most 2, because once the location
of the change is made AVL compliant, its parents will be too.

...  
if we started 'fixing' here
the lower would still be violated.

△  
violation causes parents to be violated too → So fixing it fixes parents.

review from lecture: violation fixing.

x violates AVL invariant then

- why only a difference of 2? not 3 or 4?

because we assume the violation occurred before another operation like insert or delete.

uh oh! use mutable nodes.
So then we break down the tree more:

Case 1: \( K-1 \) \( K \) \( \Rightarrow \) left rotate

Case 2: \( K \) \( K \) \( \Rightarrow \) left rotate

Case 3: \( K \) \( K-1 \) \( \Rightarrow \) break down further
Answers

left

right

k=0

k=1

Oops: 2 steps; write out each.