CSC1048 Computability and Complexity

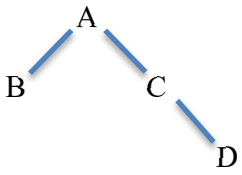
Functional Programming Lab 6: More Abstract Data Types

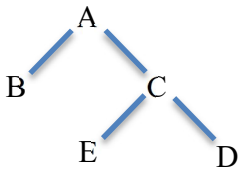
Aim

The aim of this week's exercise is to help you to learn more about the to use algebraic data types.

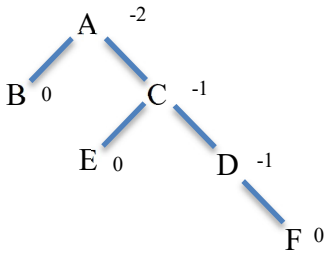
1. AVL Trees

AVL trees are a form of self balancing binary tree where the difference between the height of the left and right subtrees is no more than 1 throughout the AVL tree. So the following are AVL trees.



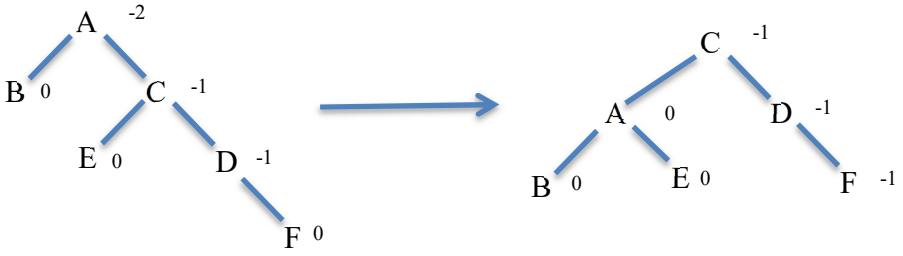


Inserting a node into an AVL tree may cause it to be come unbalanced.



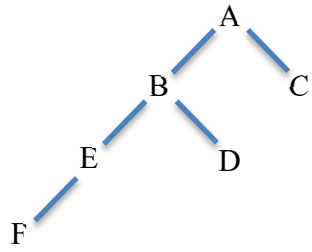
The number beside each node is its balancing factor, i.e. the height of the left subtree minus the height of the right subtree.

When this happens the tree can be restored to its "AVL state" by one or two rotations.

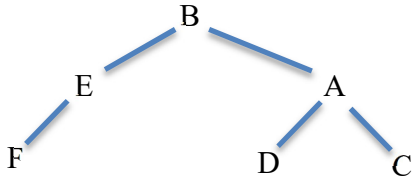


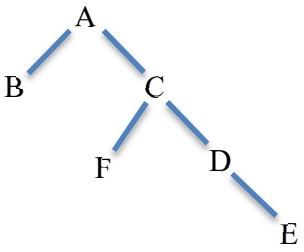
Here we pivoted left about node C.

When a subtree becomes “non-AVL” there are just 4 cases that we need to deal with. These cases can be characterised by path to the newly inserted node and the point at which the imbalance occurs on the way back to the root.

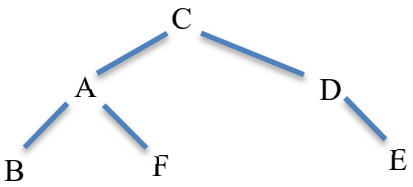


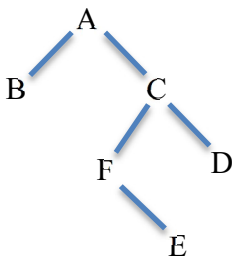
Inserted F
Left-Left case:
Pivot right
about B



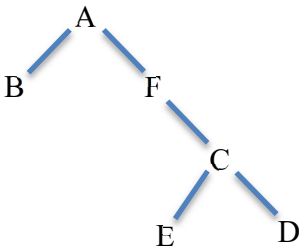


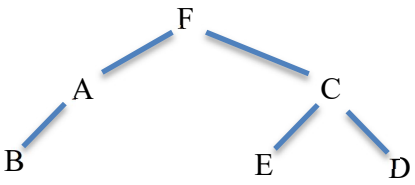
Inserted E
Right-Right case:
Pivot right about
C

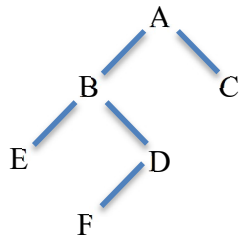




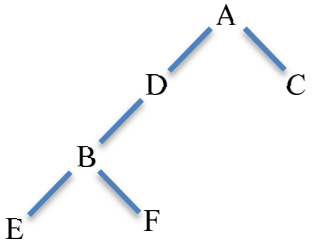
Inserted E
Right-Left case:
Pivot right about
C and then pivot
left about F

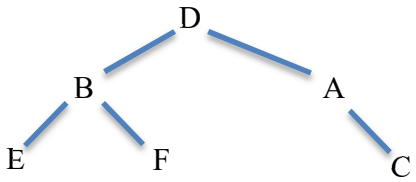




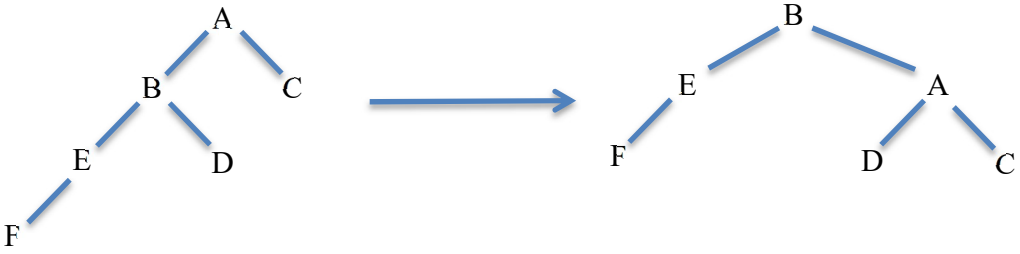


Inserted F
Left-Right case:
Pivot left about
B and then pivot
right about D





Haskell's pattern matching feature makes the above rotations easy to implement.
Consider the left-left case.



This is represented as

Root n (Root n1 lst1 rst1) rst ⟶ Root n1 lst1 (Root n rst1 rst)

which is the following Haskell function.

```
balanceLL Root n (Root n1 lst1 rst1) rst = Root n1 lst1 (Root n rst1 rst)
```

Implement AVL trees in Haskell and in particular two functions:

- insert a values into an AVL tree (maintaining its AVL properties); and
- display an AVL tree.