CSC263 Tutorial #4

AVL Trees

February 3, 2023

Things covered in this tutorial

- ⋆ What's an AVL tree?
- ⋆ How do I insert into or delete from an AVL tree?
- ⋆ How do I rotate?
- ⋆ How can I make the AVL tree very slow?[1]

[1] Still $\mathcal{O}(\log n)$ though...

First of all... BST review

To insert into a BST:

First of all... BST review

To insert into a BST:

- ⋆ Do a BST search until you find an empty spot.
- ⋆ Insert the node into the empty spot.

First of all... BST review

To insert into a BST:

- ⋆ Do a BST search until you find an empty spot.
- ⋆ Insert the node into the empty spot.

To delete a node from a BST, there are three cases:

First of all... BST review

To insert into a BST:

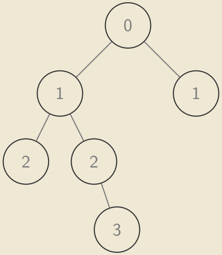
- ⋆ Do a BST search until you find an empty spot.
- ⋆ Insert the node into the empty spot.

To delete a node from a BST, there are three cases:

- ⋆ The node has no children: delete the node.
- ⋆ The node has one child: delete the node, promote the child.
- ⋆ The node has two children: swap the node with its successor, then delete the node (which has 0 or 1 children).

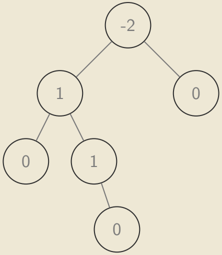
AVL trees

Height of a binary tree: starts from 0.



AVL trees

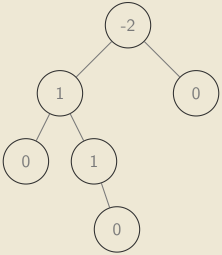
Balance factor of a node: right subtree height - left subtree height



Not an AVL tree!

AVL trees

Balance factor of a node: right subtree height - left subtree height

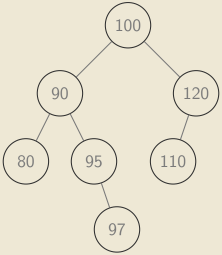


Not an AVL tree!

AVL trees are BSTs that satisfy the **AVL invariant**: every node has a balance factor of -1, 0, or 1.

AVL trees

AVL trees are BSTs that satisfy the **AVL invariant**: every node has a balance factor of -1, 0, or 1.



An AVL tree

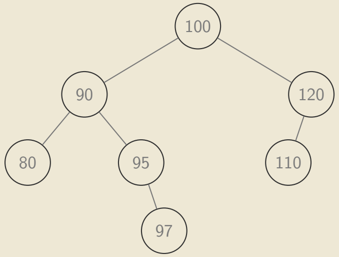
AVL trees: insertion

```
Insert(root, key, value):
  # BST insert as usual
  BST_insert(root, key, value)

  # Fix balance, update height
  root.balance_factor = root.right.height - root.left.height
  if root.balance_factor < -1 or root.balance_factor > 1:
    fix_imbalance(D) # Rotations!
  D.height = max(D.left.height, D.right.height) + 1
```

Perform the following:

- ⋆ Insert(85).
- ⋆ Insert(115).



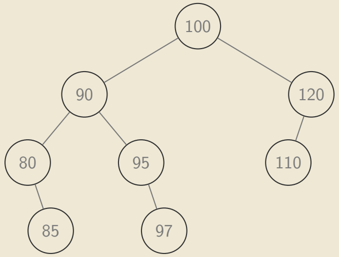
AVL trees: insertion

```
Insert(root, key, value):
  # BST insert as usual
  BST_insert(root, key, value)

  # Fix balance, update height
  root.balance_factor = root.right.height - root.left.height
  if root.balance_factor < -1 or root.balance_factor > 1:
    fix_imbalance(D) # Rotations!
  D.height = max(D.left.height, D.right.height) + 1
```

Perform the following:

- ⋆ Insert(85).
- ⋆ Insert(115).



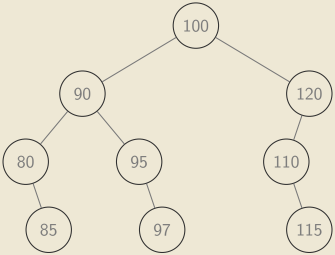
AVL trees: insertion

```
Insert(root, key, value):
  # BST insert as usual
  BST_insert(root, key, value)

  # Fix balance, update height
  root.balance_factor = root.right.height - root.left.height
  if root.balance_factor < -1 or root.balance_factor > 1:
    fix_imbalance(D) # Rotations!
  D.height = max(D.left.height, D.right.height) + 1
```

Perform the following:

- ⋆ Insert(85).
- ⋆ Insert(115).



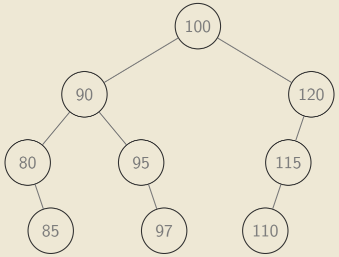
AVL trees: insertion

```
Insert(root, key, value):
  # BST insert as usual
  BST_insert(root, key, value)

  # Fix balance, update height
  root.balance_factor = root.right.height - root.left.height
  if root.balance_factor < -1 or root.balance_factor > 1:
    fix_imbalance(D) # Rotations!
  D.height = max(D.left.height, D.right.height) + 1
```

Perform the following:

- ⋆ Insert(85).
- ⋆ Insert(115).



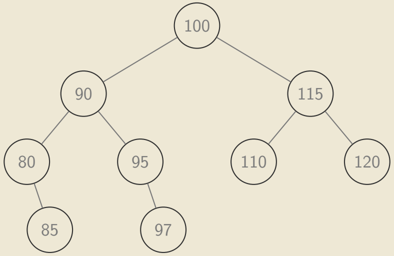
AVL trees: insertion

```
Insert(root, key, value):
  # BST insert as usual
  BST_insert(root, key, value)

  # Fix balance, update height
  root.balance_factor = root.right.height - root.left.height
  if root.balance_factor < -1 or root.balance_factor > 1:
    fix_imbalance(D) # Rotations!
  D.height = max(D.left.height, D.right.height) + 1
```

Perform the following:

- ⋆ Insert(85).
- ⋆ Insert(115).



AVL trees: insertion

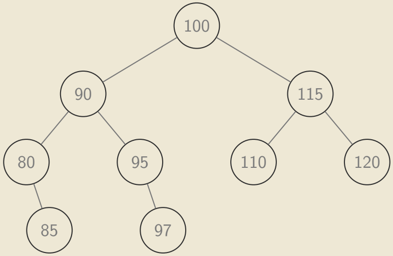
```
Insert(root, key, value):
  # BST insert as usual
  BST_insert(root, key, value)

  # Fix balance, update height
  root.balance_factor = root.right.height - root.left.height
  if root.balance_factor < -1 or root.balance_factor > 1:
    fix_imbalance(D) # Rotations!
  D.height = max(D.left.height, D.right.height) + 1
```

Perform the following:

- ⋆ Insert(85).
- ⋆ Insert(115).

For Insert(115), we needed to do a left-right rotation.

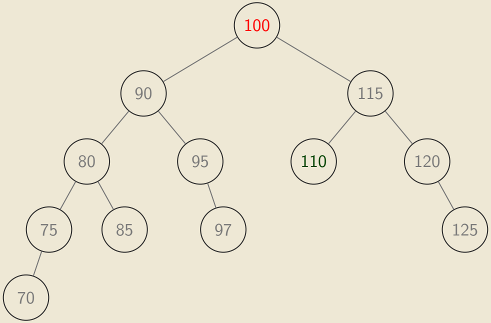


AVL trees: deletion

Similar to insertion: delete as in a BST, and then rebalance and update attributes. However, multiple rebalances may be needed.

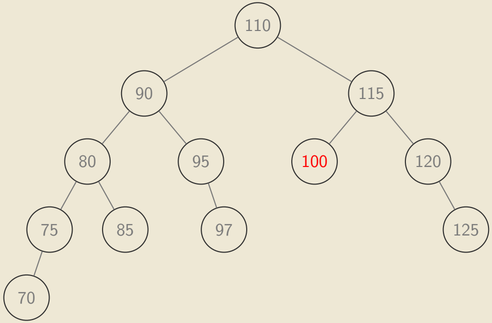
AVL trees: deletion

Similar to insertion: delete as in a BST, and then rebalance and update attributes. However, multiple rebalances may be needed.



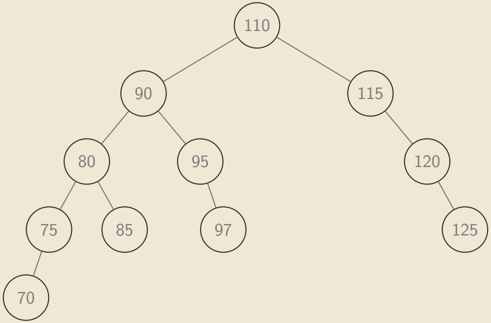
AVL trees: deletion

Similar to insertion: delete as in a BST, and then rebalance and update attributes. However, multiple rebalances may be needed.



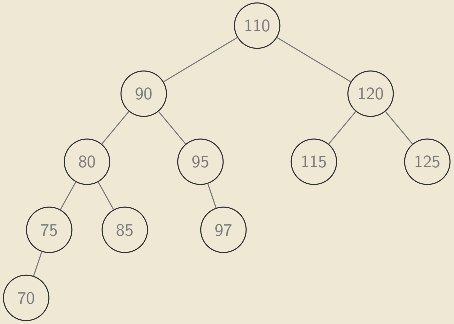
AVL trees: deletion

Similar to insertion: delete as in a BST, and then rebalance and update attributes. However, multiple rebalances may be needed.



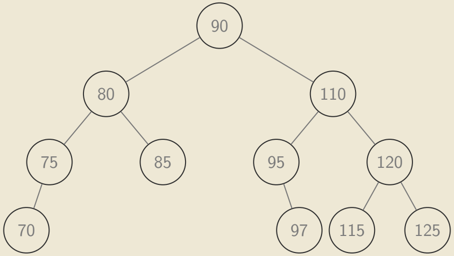
AVL trees: deletion

Similar to insertion: delete as in a BST, and then rebalance and update attributes. However, multiple rebalances may be needed.



AVL trees: deletion

Similar to insertion: delete as in a BST, and then rebalance and update attributes. However, multiple rebalances may be needed.



AVL trees: deletion

Worst case: $\mathcal{O}(\log n)$ rotations! (Triangles are subtrees with labelled heights)

AVL trees: deletion

Worst case: $\mathcal{O}(\log n)$ rotations! (Triangles are subtrees with labelled heights)