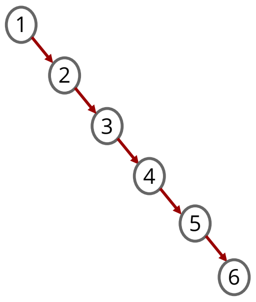
COMP2321 – DATA STRUCTURES

AVL Trees

Dr. Radi Jarrar
Department of Computer Science
Birzeit University

AVL

- AVL (Adelson-Velskii and Landis) tree is a binary search tree with additional balance property.
- It actually ensures that the depth of the tree is O(log n).
- Why we need balance?



AVL (2)

- Height of the left subtree and height of the right subtree differ by at most 1
- For a tree to be balanced, the value has to only be -1, 0, 1 at each node.
- Balance(tree) = height of (tree.left) - height of (tree.right) (height of the root's left branch and the height of the root's right branch).

AVL (3)

- This means that all elements in the tree can be ordered in some consistent manner.

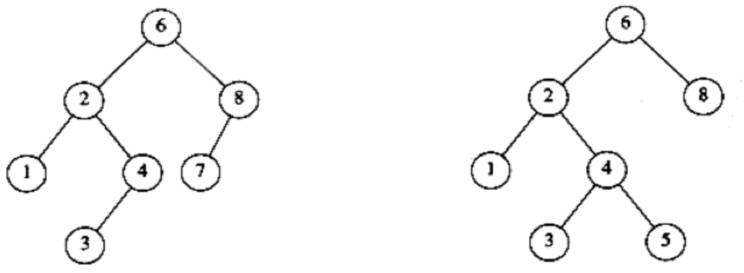


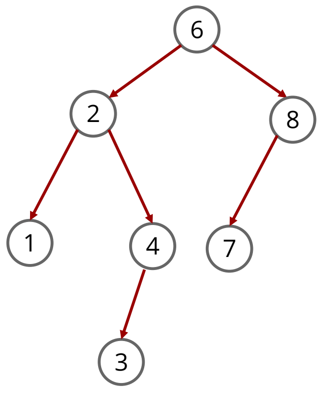
Figure 4.29 Two binary search trees. Only the left tree is **AVL.**

Rotations

- The balance of an AVL-tree is made via rotations.
- **The algorithm:** the basic algorithm is to start at the node inserted and travel up the tree, updating the balance information at every node on the path.
- If we get to the root without having found any badly balanced nodes, then nothing to update.
- Otherwise, we do rotation at the first bad node found, adjust the balance, and we are done.

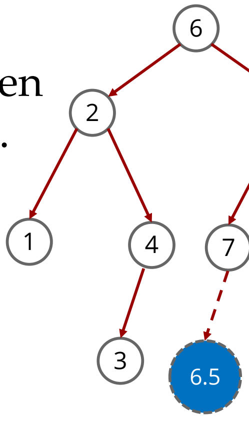
Rotations (2)

- If the insertion causes some node in the AVL tree to lose the balance, do a rotation at that node. In most cases, this is sufficient to rebalance the tree.
- There are two types of rotations
  - Single Rotation
  - Double Rotation
- Example: Insert the node 6.5 to the following tree



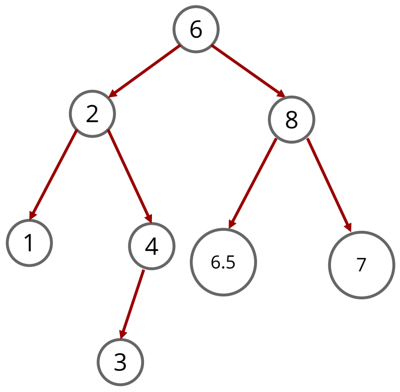
Rotations (3)

- After inserting 6.5, the original AVL tree on the left, node 8 becomes unbalanced.
- Thus, we do a single rotation between 7 & 8 obtaining the tree on the right.



Rotations (4)

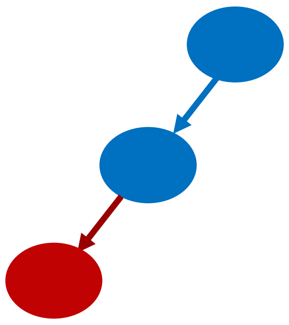
- After rotation the tree becomes



Rotations (5)

- To re-balance a tree there are 4 different situations:

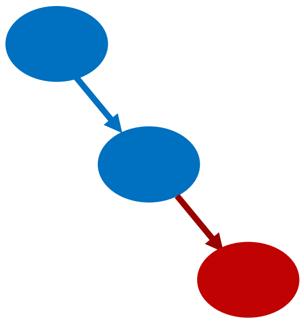
1. insert into the left sub-tree of the left child (Right single rotation)



Rotations (5)

- To re-balance a tree there are 4 different situations:

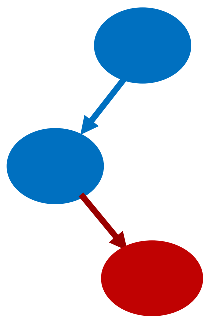
2. insert into the right sub-tree of the right child (Left single rotation)



Rotations (5)

- To re-balance a tree there are 4 different situations:

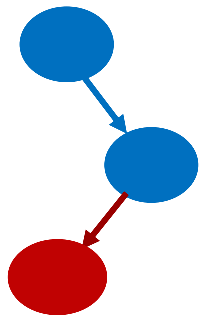
3. insert into the right of the left sub-tree (LR - double rotation)



Rotations (5)

- To re-balance a tree there are 4 different situations:

4. insert into the left of the right sub-tree (RL – double rotation)

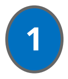


Example

- Create an AVL tree and insert the values from 1 to 7.
- Insert(1)

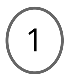
Example

- Create an AVL tree and insert the values from 1 to 7.
- Insert(1)



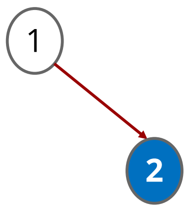
Example

- Create an AVL tree and insert the values from 1 to 7.
- Insert(**2**)



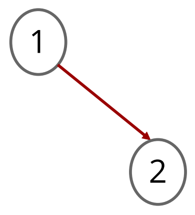
Example

- Create an AVL tree and insert the values from 1 to 7.
- Insert(**2**)



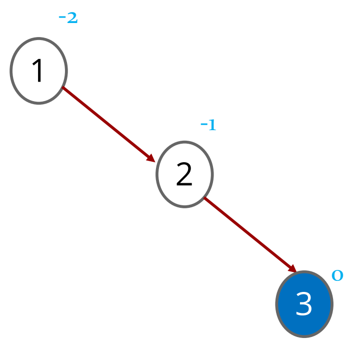
Example

- Create an AVL tree and insert the values from 1 to 7.
- Insert(**3**)



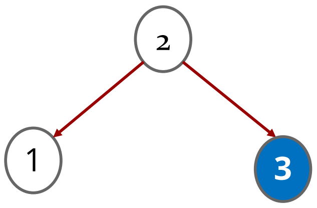
Example

- Create an AVL tree and insert the values from 1 to 7.
- Insert(**3**)



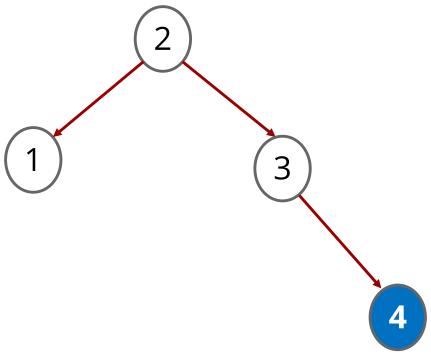
Example

- Insert(4)



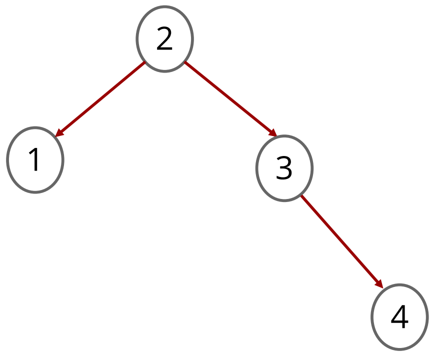
Example

- Insert(4)



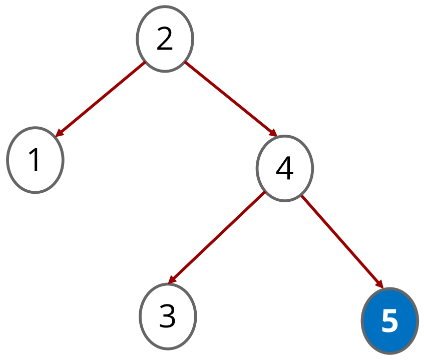
Example

- Insert(**5**)



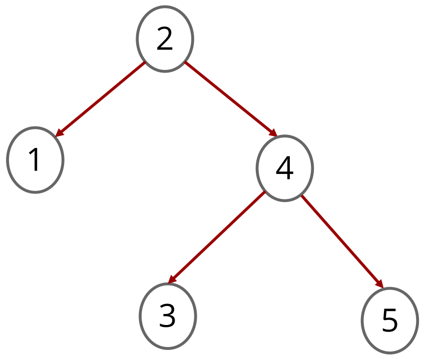
Example

- Insert(**5**)



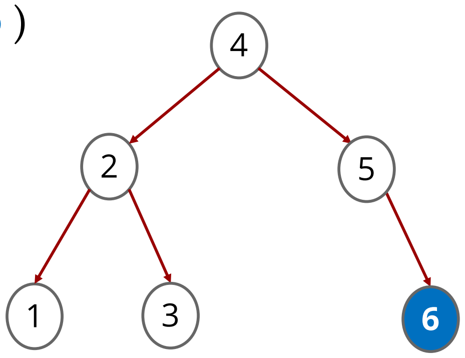
Example

- Insert(**6**)



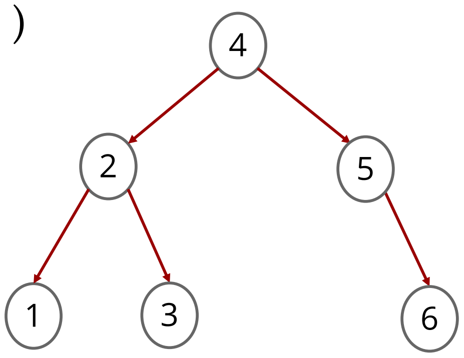
Example

- Insert(**6**)



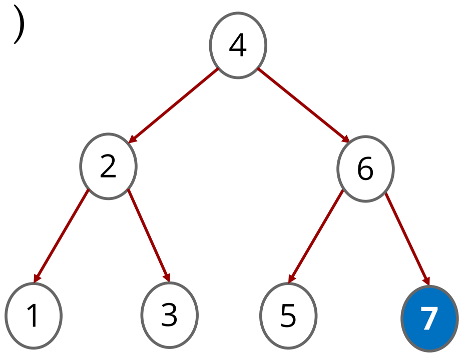
Example

- Insert(**7**)



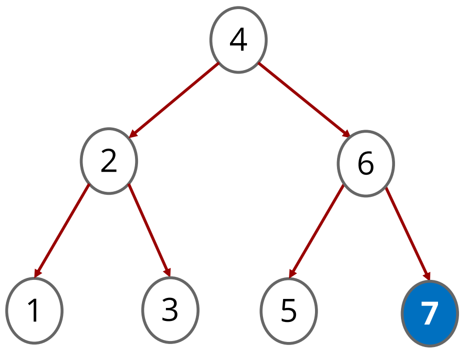
Example

- Insert(**7**)



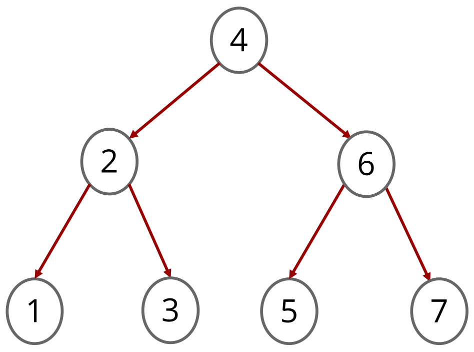
Example

- Now insert the numbers 10 – 16 backward to the tree



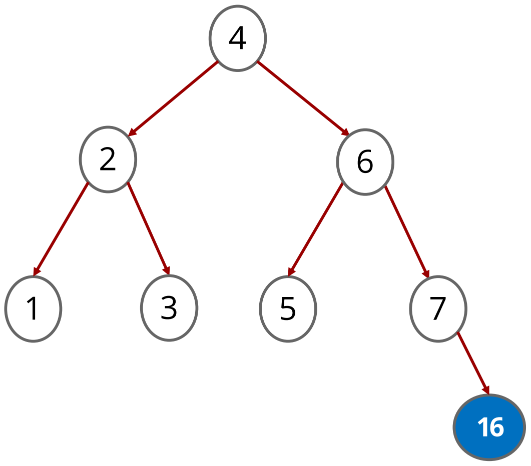
Example

- Insert(**16**)



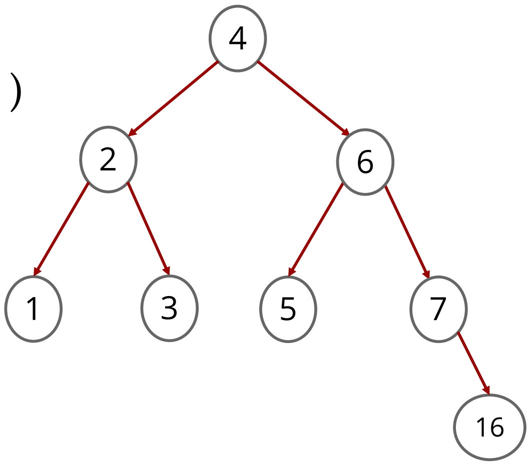
Example

- Insert(**16**)



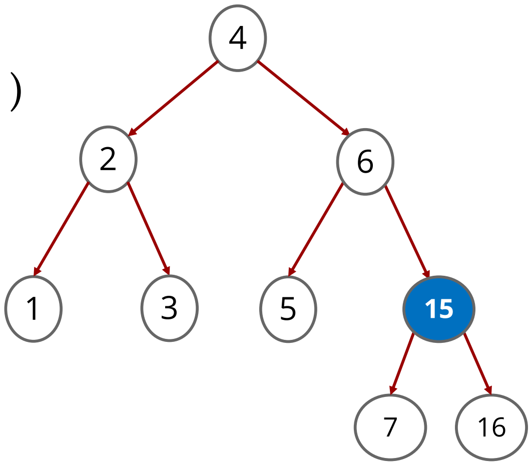
Example

- Insert(**15**)



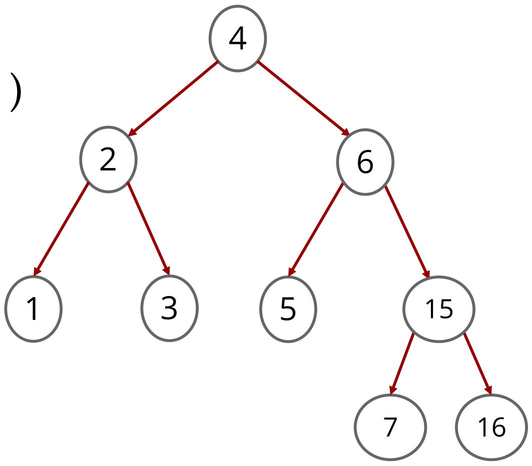
Example

- Insert(**15**)



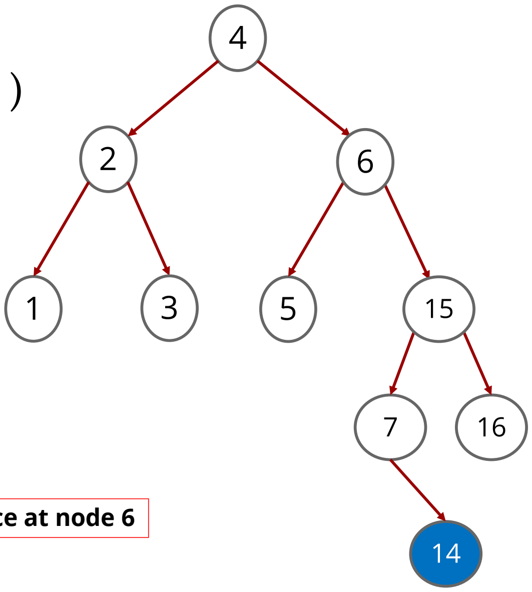
Example

- Insert(**14**)



Example

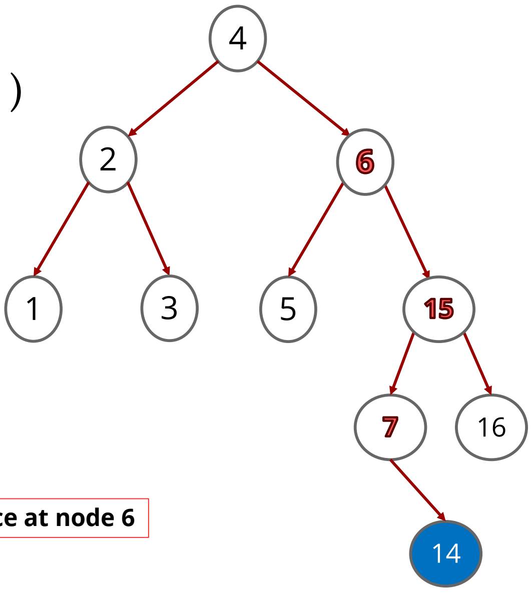
- Insert(**14**)



14 causes imbalance at node 6

Example

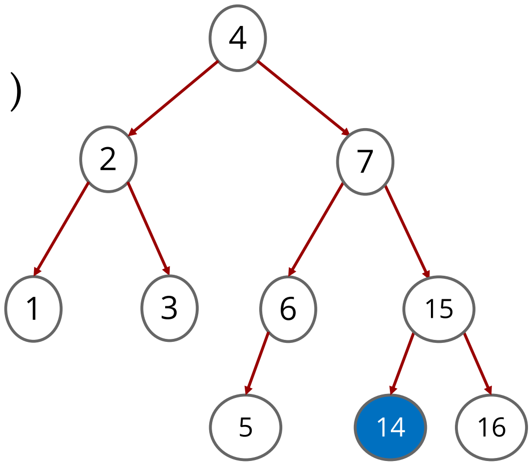
- Insert(**14**)



14 causes imbalance at node 6

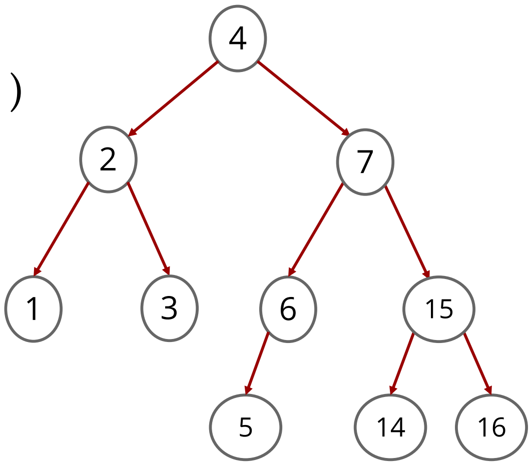
Example

- Insert(**14**)



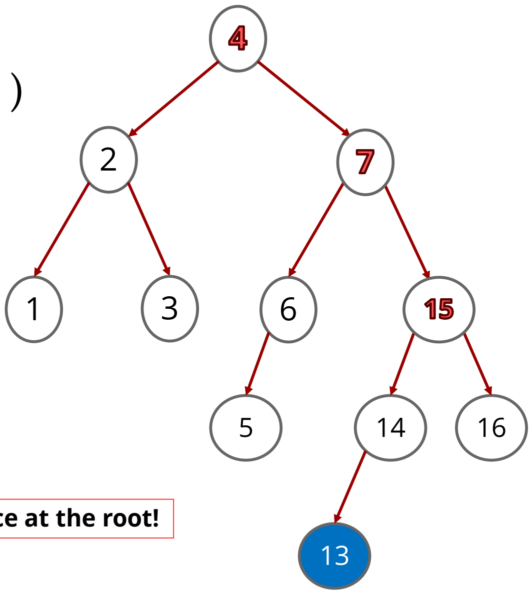
Example

- Insert(**13**)



Example

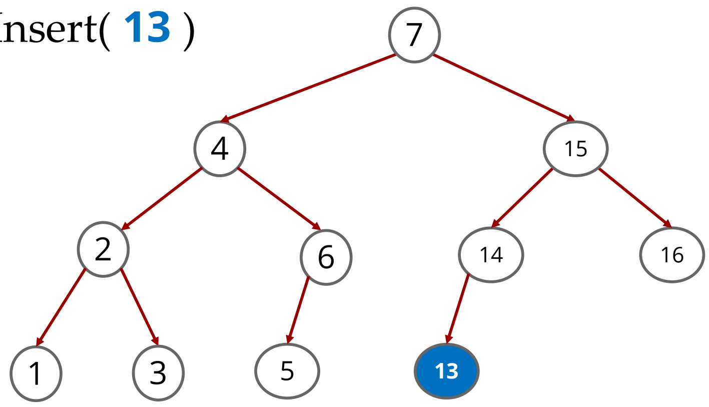
- Insert(**13**)



13 causes imbalance at the root!

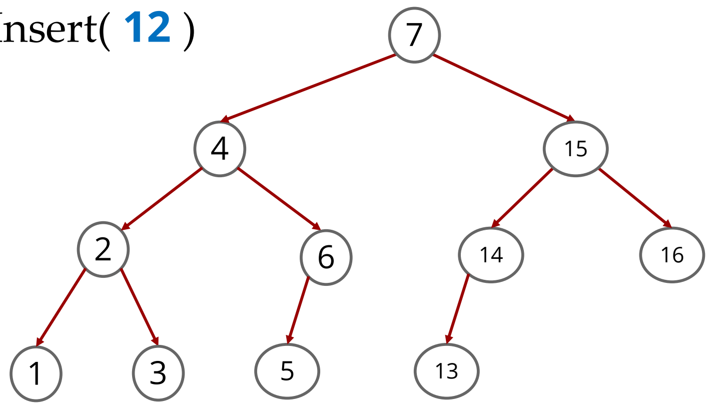
Example

- Insert(**13**)



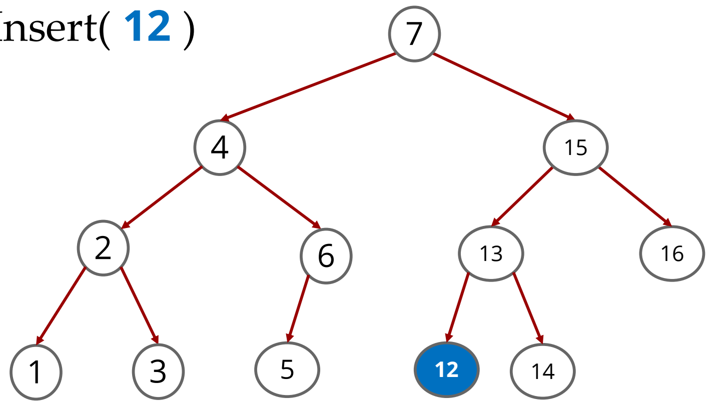
Example

- Insert(**12**)



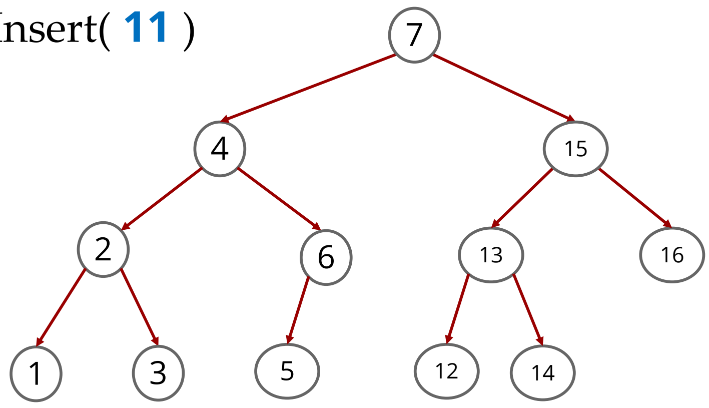
Example

- Insert(**12**)



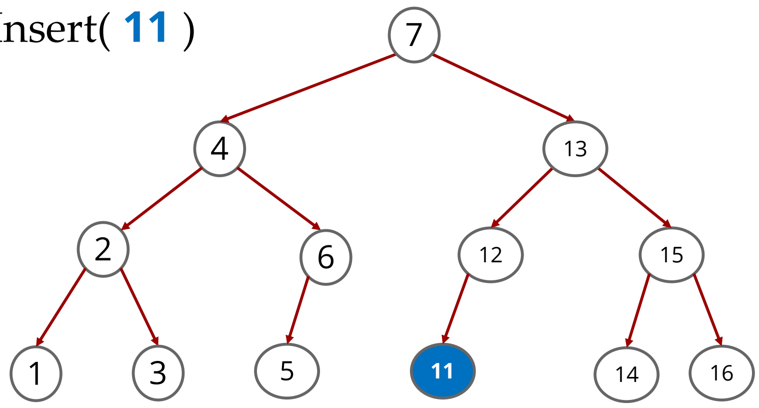
Example

- Insert(**11**)



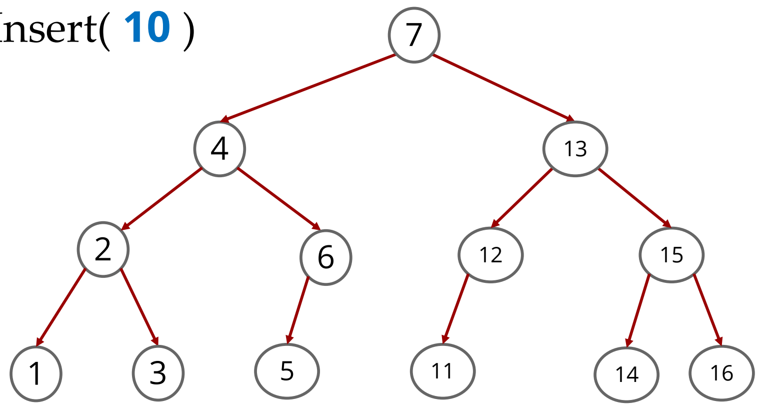
Example

- Insert(**11**)



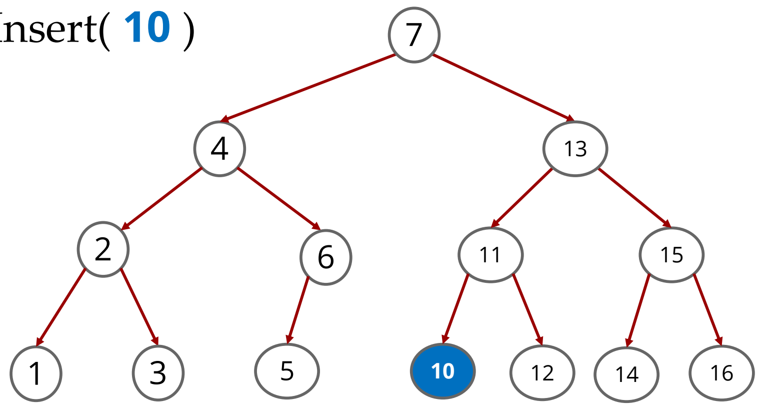
Example

- Insert(**10**)



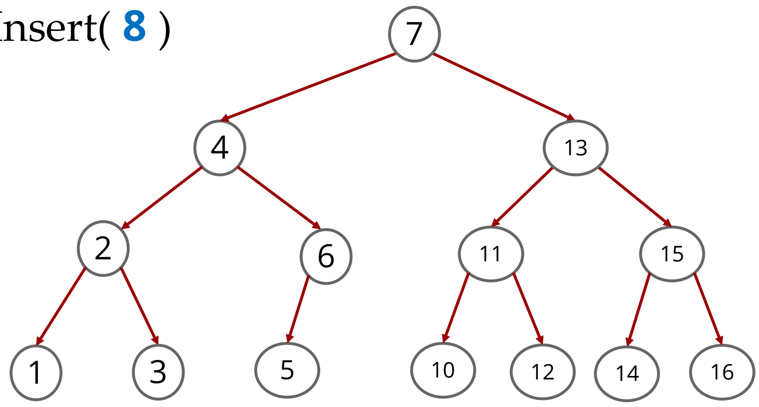
Example

- Insert(**10**)



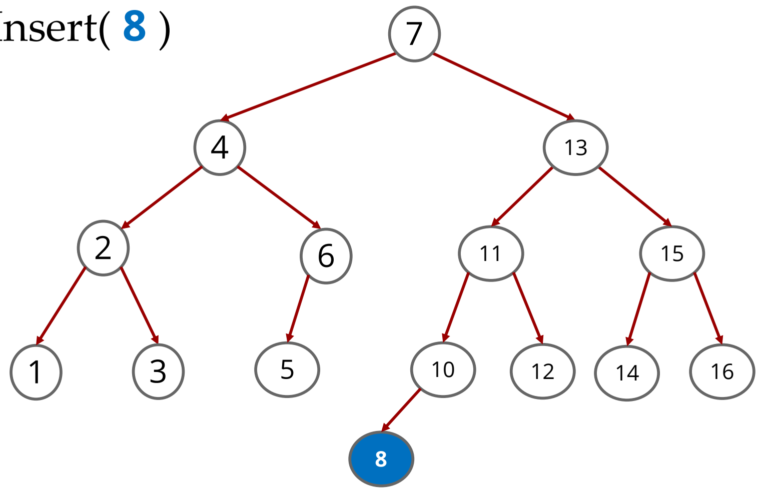
Example

- Insert(**8**)



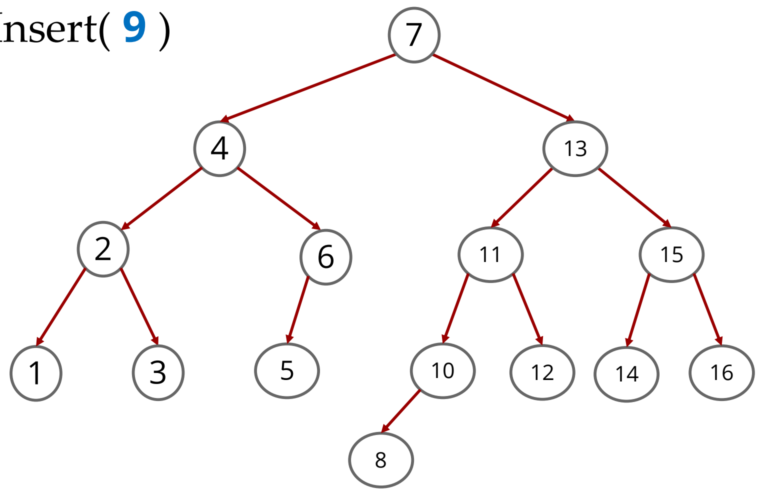
Example

- Insert(**8**)



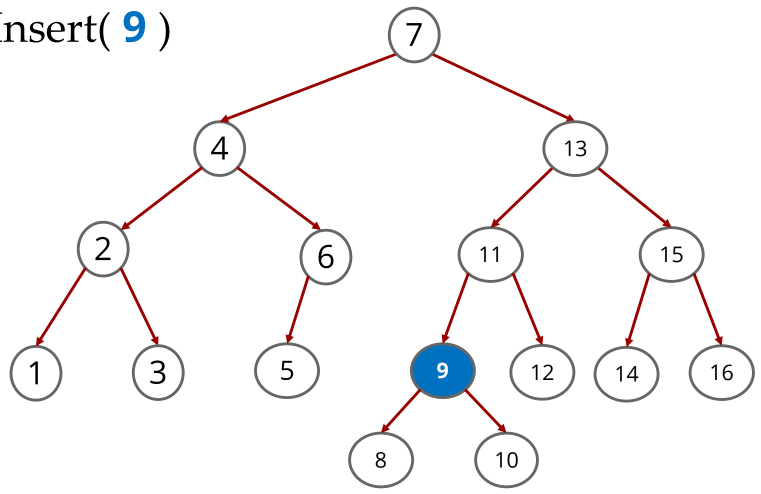
Example

- Insert(9)



Example

- Insert(9)



AVL Tree Implementation

- The main structure is the same as the BST
- Differs in adding the Rotations methods
  - `SingleRotateWithLeft`
  - `SingleRotateWithRight`
  - `DoubleRotateWithRight`
  - `DoubleRotateWithLeft`
- Code will be sent (the textbook author’s implementation)

Applications of AVL Trees

- Used frequently for quick searching as it takes O(log n) because the tree is balanced
- Used in Databases
  - Intensive look-up applications where insertion & deletion are not that frequent but search for items is performed frequently
  - The large cost of rebalancing is a limitation!
- Used for in-memory collections such as sets and dictionaries