AVL Analysis

Prof. Wade Fagen-Ulmschneider

Balanced BSTs that are kept in balance through tree rotations on insert and remove are called **AVL** trees, named after **Adelson-Velsky** and **Landis**.
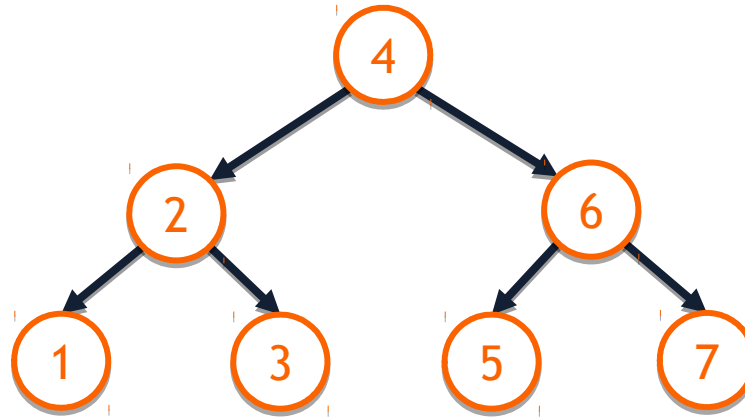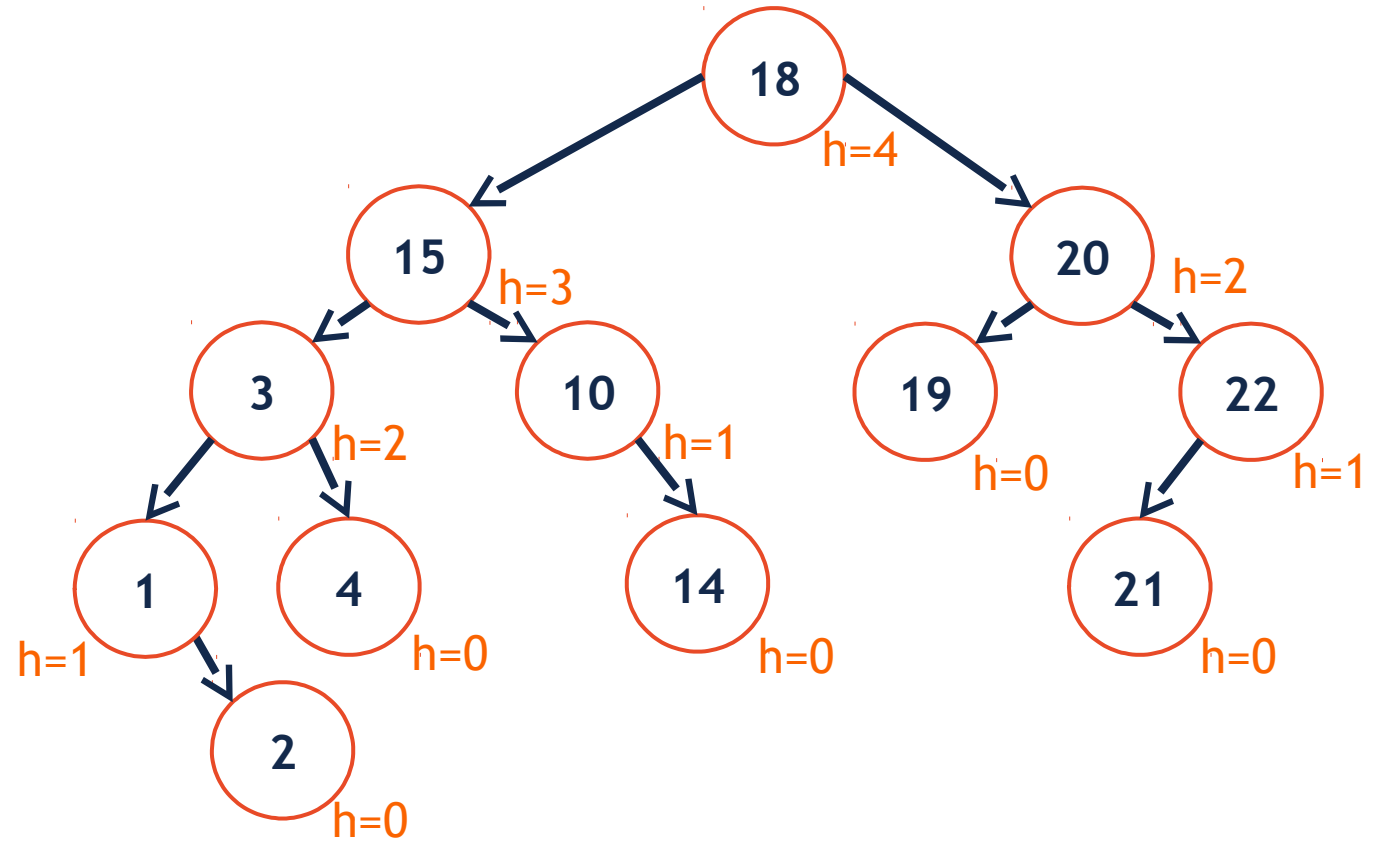
AVL Tree

Everything about finding, inserting, and removing from a BST remains true.

In an AVL tree, we add steps to ensure we maintain balance.

- Extra work on insert/remove
- To quickly compute the balance factor, AVL trees store the height of every node as part of the node itself.
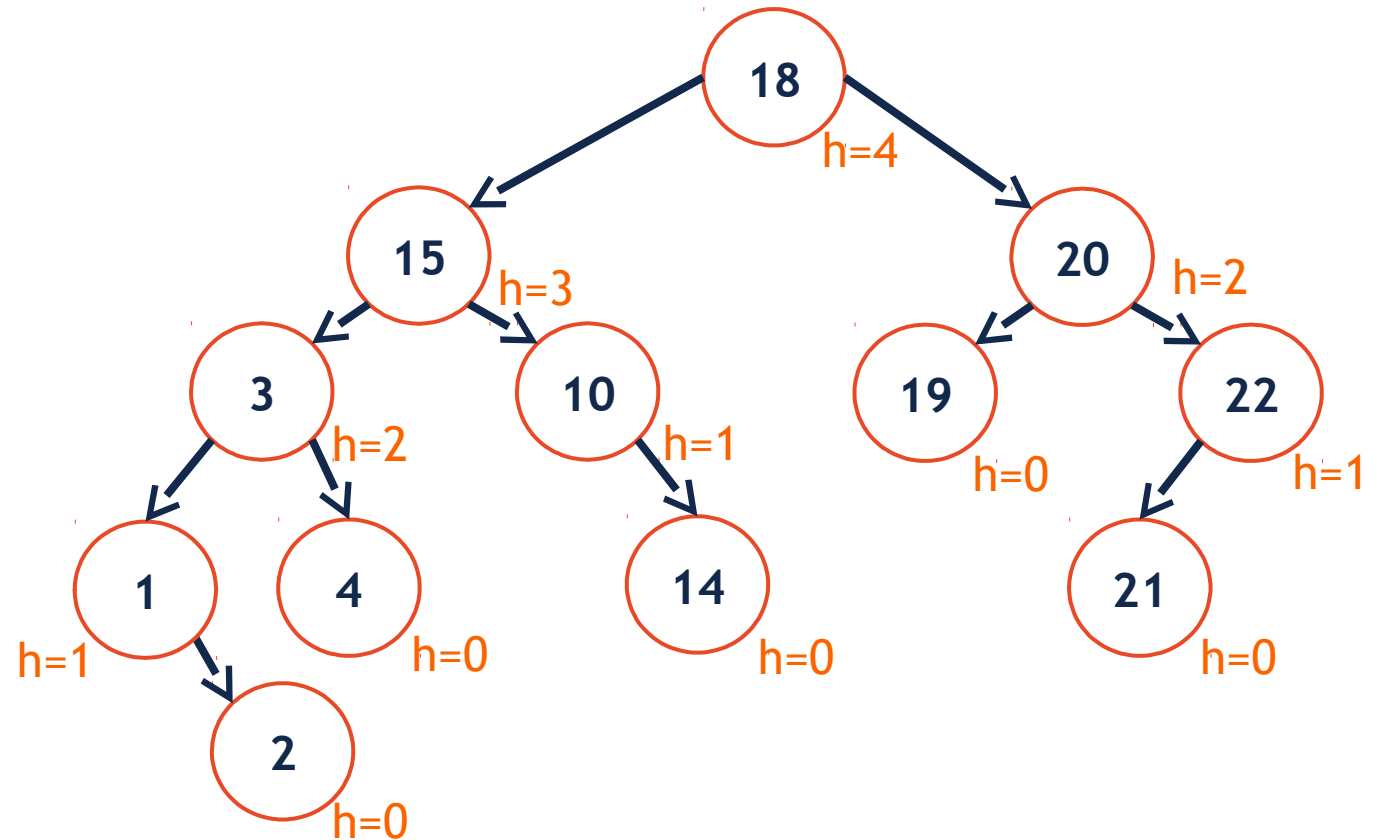
AVL Trees

insert(11)

AVL Insert

insert(11)

1: Insert at proper place.
2: Check for imbalance.
3: Rotate, if necessary.
4: Update height.

```cpp
template <typename K, typename D>
void AVL<K, D>::_ensureBalance(TreeNode *& cur) {
  // Calculate the balance factor:
  int balance = height(cur->right) - height(cur->left);

  // Check if the node is current not in balance:
  if ( balance == -2 ) {
    int l_balance = height(cur->left->right) - height(cur->left->left);
    if ( l_balance == -1 ) { _rotateRight( cur ); }
    else                   { _rotateLeftRight( cur ); }
  } else if ( balance == 2 ) {
    int r_balance = height(cur->right->right) - height(cur->right->left);
    if( r_balance == 1 ) { _rotateLeft( cur ); }
    else                 { _rotateRightLeft( cur ); }
  }

  _updateHeight(cur);
}
```
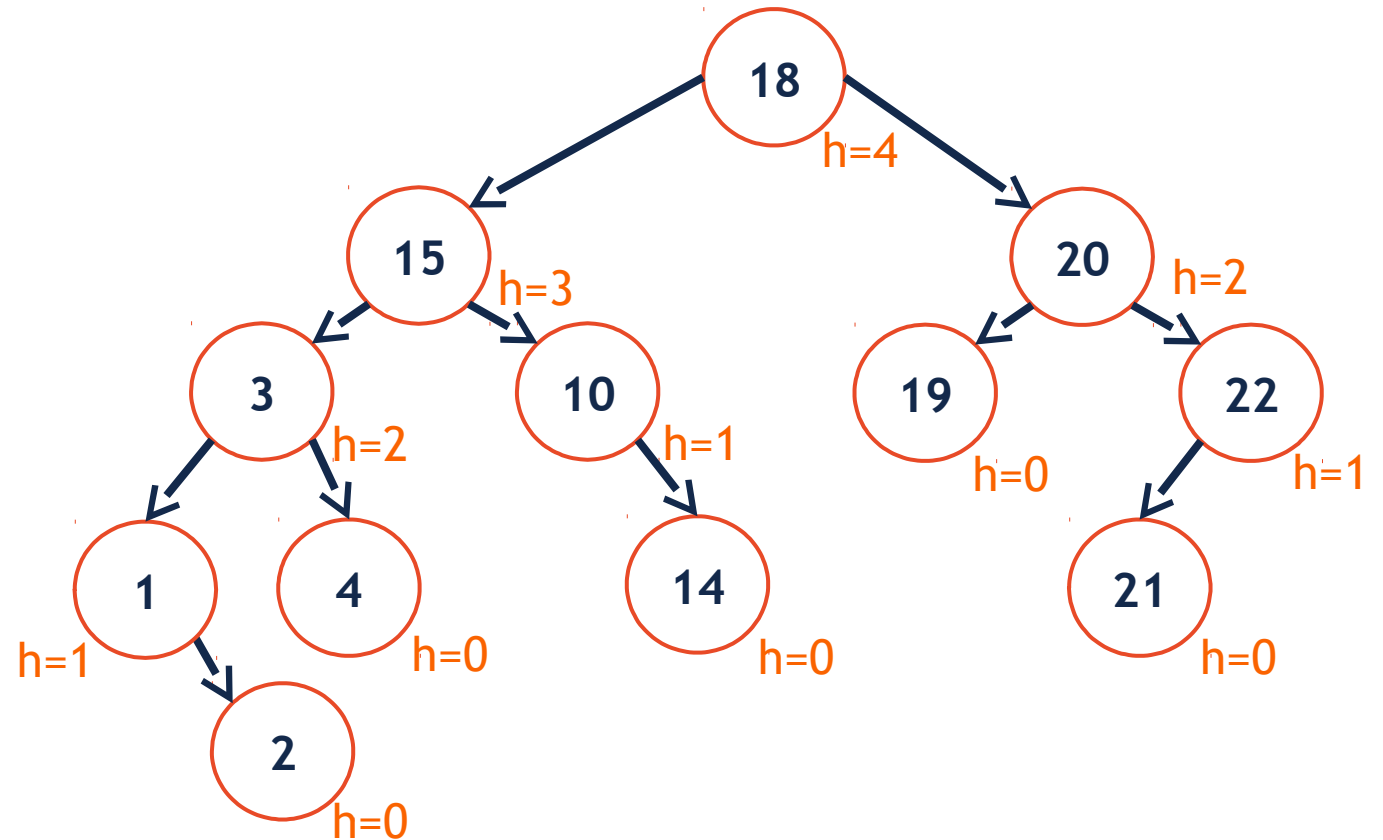
```cpp
142  template <typename K, typename D>
143  void AVL<K, D>::_updateHeight(TreeNode *& cur) {
144      cur->height = 1 + max(height(cur->left), height(cur->right));
145  }
```

```cpp
81  template <typename K, typename D>
82  void AVL<K, D>::_rotateLeft(TreeNode *& cur) {
83      TreeNode *x = cur;
84      TreeNode *y = cur->right;
85
86      x->right = y->left;
87      y->left = x;
88      cur = y;
89
90      _updateHeight(x);
91      _updateHeight(y);
92  }
```

AVL::remove

remove(15)

```cpp
/**
* Recursive IoP remove.
*/
template <typename K, typename D>
const D & AVL<K, D>::_iopRemove(TreeNode *& node, TreeNode *& iop) {
    if (iop->right != nullptr) {
        // IoP not found, keep going deeper:
        const D & d = _iopRemove(node, iop->right);
        if (iop) { _ensureBalance(iop); }
        return d;
    } else {
        // Found IoP, swap the location:
        _swap( node, iop );
        std::swap( node, iop );

        // Remove the swapped node (at iop's position):
        return _remove(iop);
    }
}
```

AVL Trees

- An AVL Tree is an implementation of a **balanced** Binary Search Tree (BST).

- An implementation of an AVL tree starts with a BST implementation and adds two key ideas:
  – Maintains the height at each node
  – Maintains the balance factor after insert and remove