

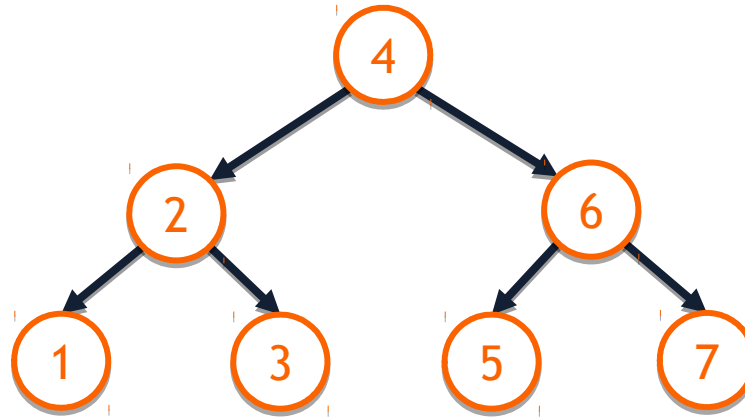
AVL Analysis

Prof. Wade Fagen-Ulmschneider

I ILLINOIS

ALMA MATER
TO BRIGHT CHILDREN
OF THE FUTURE

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

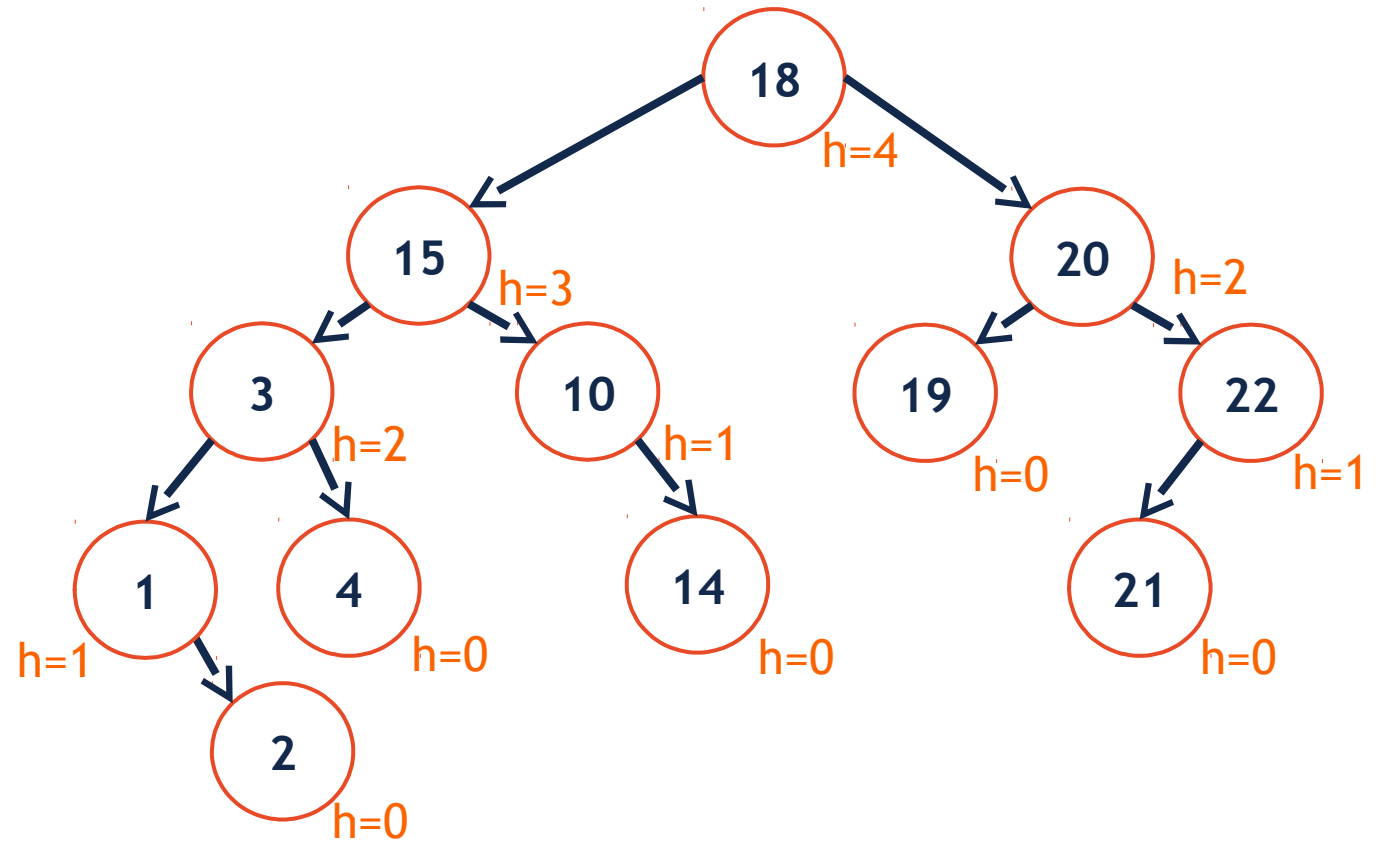
Everything about finding, inserting, and removing from a BST remains true.

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

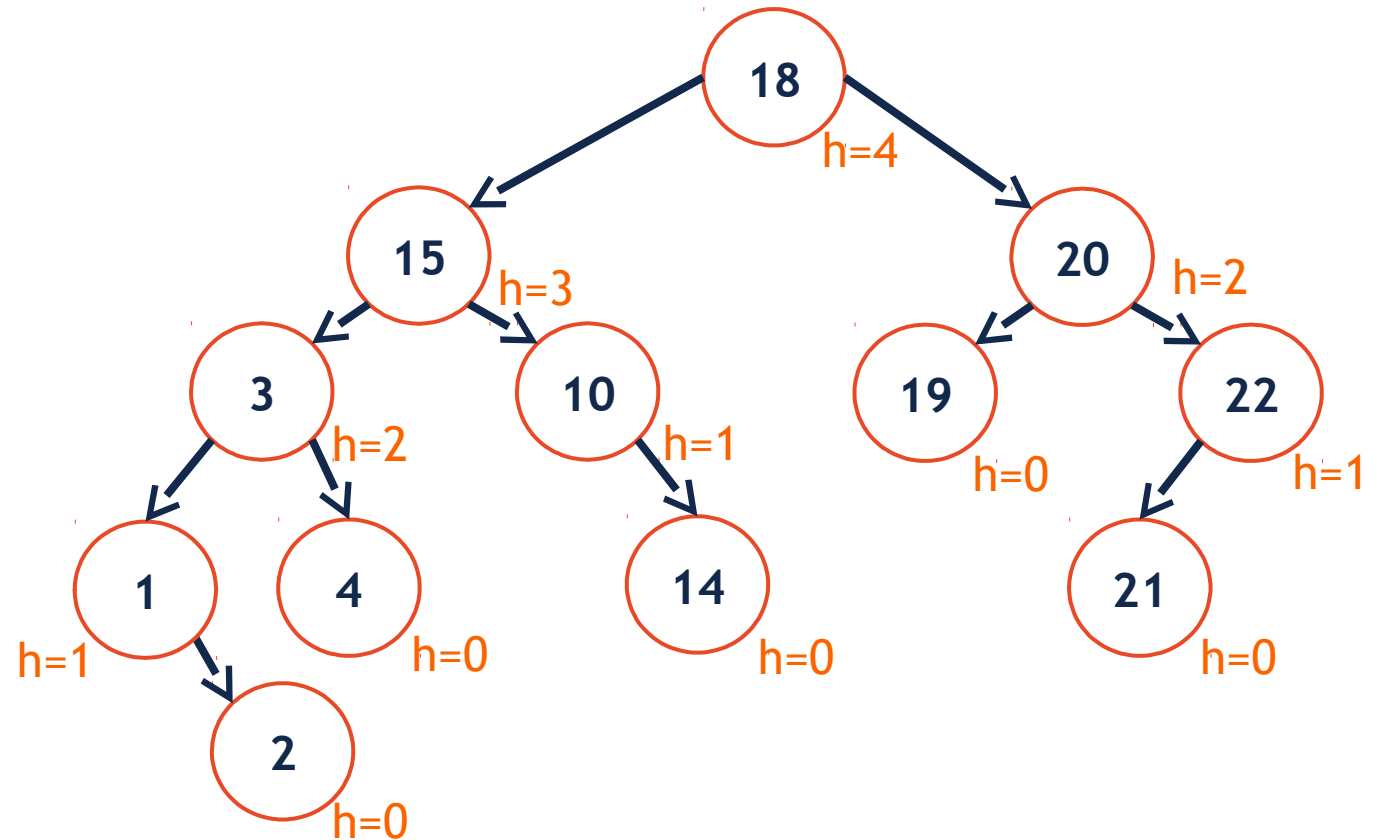
insert(11)



AVL Insert

insert(11)

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



avl/AVL.cpp

```
119 template <typename K, typename D>
120 void AVL<K, D>::_ensureBalance(TreeNode *& cur) {
121     // Calculate the balance factor:
122     int balance = height(cur->right) - height(cur->left);
123
124     // Check if the node is current not in balance:
125     if ( balance == -2 ) {
126         int l_balance = height(cur->left->right) - height(cur->left->left);
127         if ( l_balance == -1 ) { _rotateRight( cur ); }
128         else { _rotateLeftRight( cur ); }
129     } else if ( balance == 2 ) {
130         int r_balance = height(cur->right->right) - height(cur->right->left);
131         if( r_balance == 1 ) { _rotateLeft( cur ); }
132         else { _rotateRightLeft( cur ); }
133     }
134
135     _updateHeight(cur);
136 }
```

avl/AVL.cpp

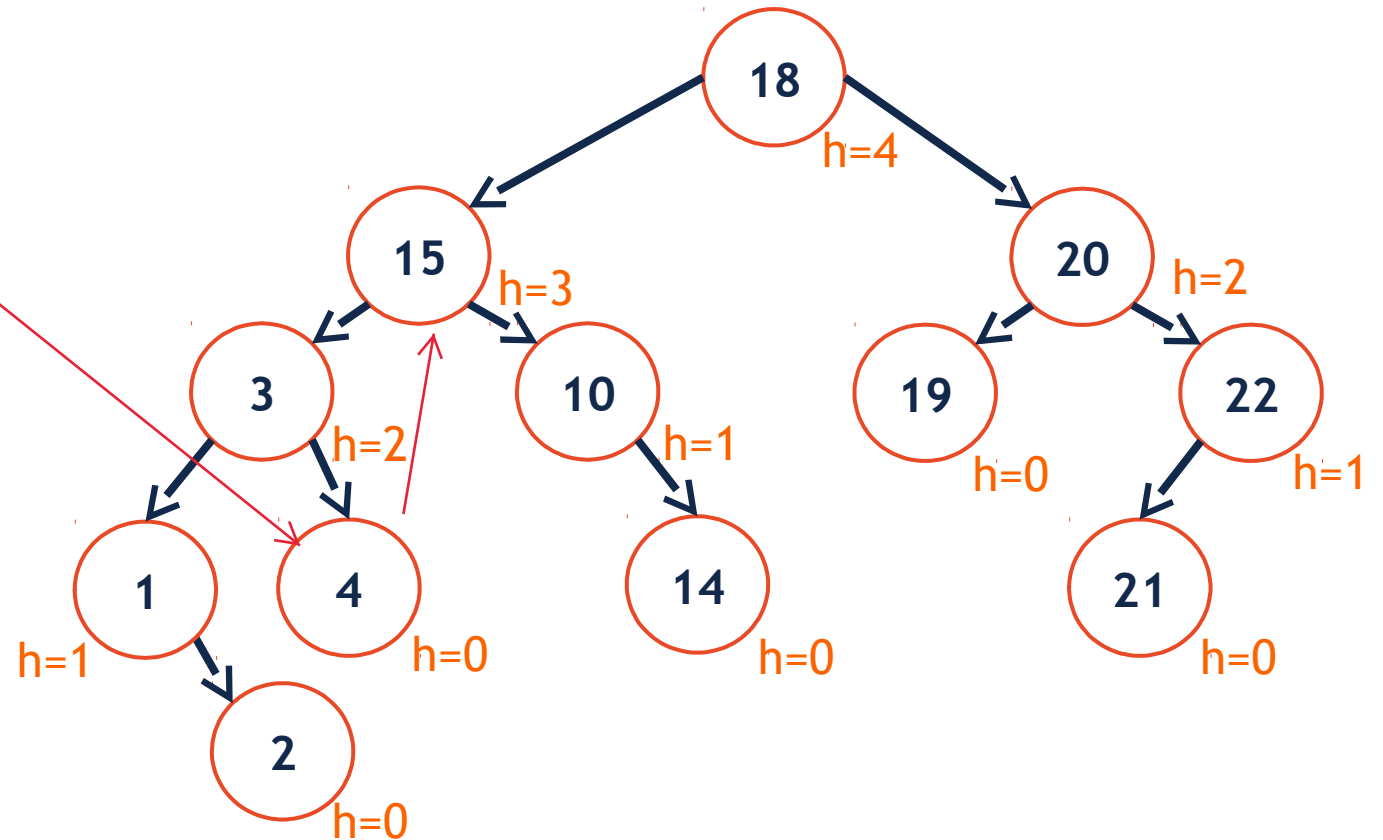
```
142 template <typename K, typename D>
143 void AVL<K, D>::_updateHeight(TreeNode *& cur) {
144     cur->height = 1 + max(height(cur->left), height(cur->right));
145 }
```

```
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)

* IOP: most right node in
the left branch
* So 4 to replace removed 15



avl/AVL.cpp

```
221  /**
222  * Recursive IoP remove.
223  */
224  template <typename K, typename D>
225  const D & AVL<K, D>::_iopRemove(TreeNode *& node, TreeNode *& iop) {
226      if (iop->right != nullptr) {
227          // IoP not found, keep going deeper:
228          const D & d = _iopRemove(node, iop->right);
229          if (iop) { _ensureBalance(iop); }
230          return d;
231      } else {
232          // Found IoP, swap the location:
233          _swap( node, iop );
234          std::swap( node, iop );
235
236          // Remove the swapped node (at iop's position):
237          return _remove(iop);
238      }
239  }
```

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

