# Data Structures
# Inserting Nodes Into an AVL Tree

Cain Susko

Queen's University
School of Computing

February 19, 2022

# Operations

For all AVL operations it is assumed that the tree was balanced *before* the operation began.

# Insert

The insertion process begins the same as with a normal BST, but adding a new node to the AVL tree, it may unbalance the tree by 1. This would cause the AVL tree to become invalid. We rectify this by using an operation called Rotation.

**Rotation** If a node has becore out of balance in a given direction, we *rotate* it in the **opposite** direction. This maintains the inorder ordering of keys. The height of each touched node must be updated as well.

**Unbalanced Cases** Consider the lowest node $k$ that has now become unbalanced. The new 'offending' node could be in one of the four following *grandchild* subtrees relative to $k$:

**Left / Right** There was an insert into a left child's left subtree or insert into right childs right subtree. All that is required is to rotate once away from the direction of the offending node.

**L-R / R-L** There was an insert in the left subtree of a right child or a right subtree of a left child. There first must be a rotation on the offending new node (opposide of its direction), and then a rotation on the offending, existing child node (again, opposide to its direction).

## Algorithm

thus, the insertion algorithm for AVL trees is the following:

1. Find a Location within a given tree for the new value that satisfies the existing BST requirements.
2. Insert the node.
3. Search from the insert up the tree, looking for imbalances.
4. if an imbalance is found:
   (a) if it is **Left / Right**, perform a single rotation.
   (b) if it is **L-R / R-L**, perform a double rotation