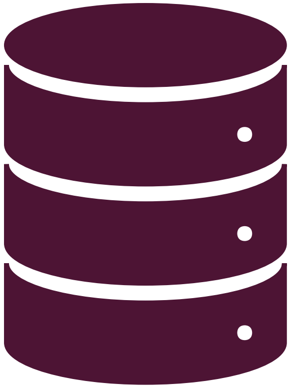


DATA STRUCTURES

WENYE LI
CUHK-SZ

AVL (ADELSON-VELSKII AND LANDIS) TREE

- A balanced binary search tree where the height of the two subtrees (children) of a node differs by at most one. Look-up, insertion, and deletion are O(log n), where n is the number of nodes in the tree.

- Note: The materials in this slides are NOT in the scope of final exam.

DEFINITION OF HEIGHT

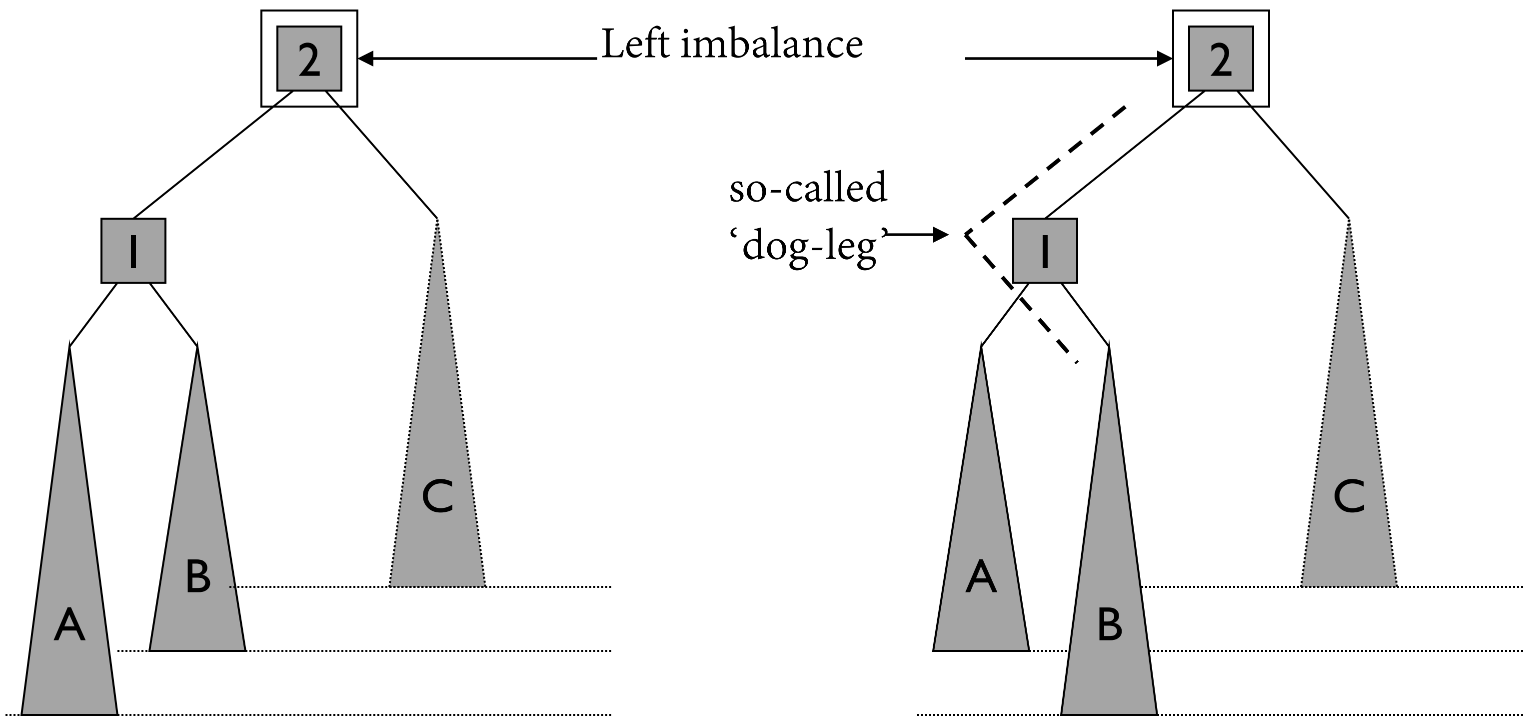
- Height: the length of the longest path from a node to a leaf.
  - All leaves have a height of 0
  - An empty tree has a height of –1

THE INSERTION PROBLEM

- Unless keys appear in just the right order, imbalance will occur
- It can be shown that there are only two possible types of imbalance (see next slide):
  - Left-left (or right-right) imbalance
  - Left-right (or right-left) imbalance
  - The right-hand imbalances are the same, by symmetry

THE TWO TYPES OF IMBALANCE

- Left-left (right-right)
- Left-right (right-left)



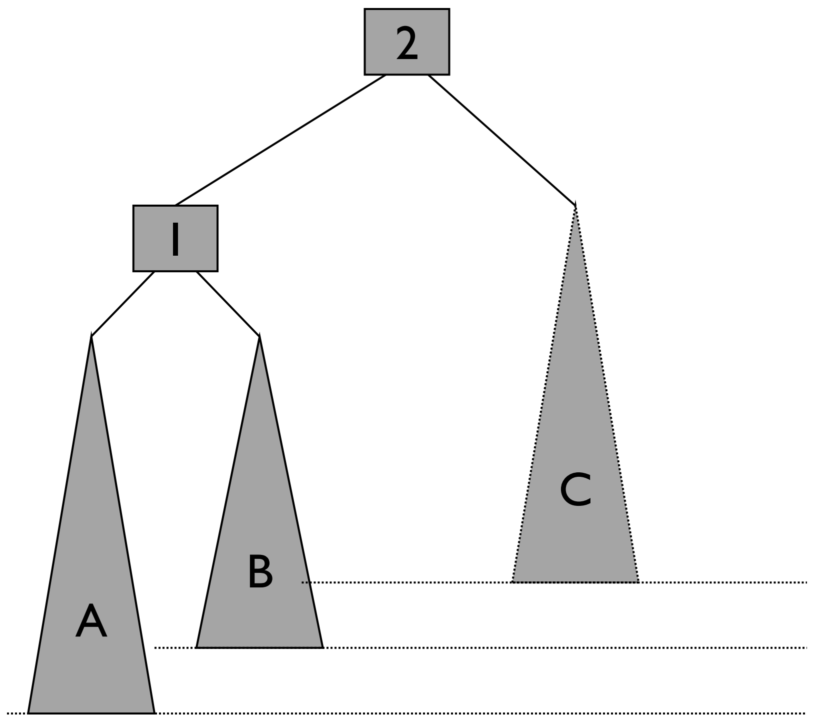
There are no other possibilities for the left (or right) subtree

LOCALISING THE PROBLEM

- Two principles:
  - Imbalance will only occur on the path from the inserted node to the root (only these nodes have had their subtrees altered - local problem)
  - Rebalancing should occur at the deepest unbalanced node (local solution too)

LEFT(LEFT) IMBALANCE (1)
[AND RIGHT(RIGHT) IMBALANCE, BY SYMMETRY]

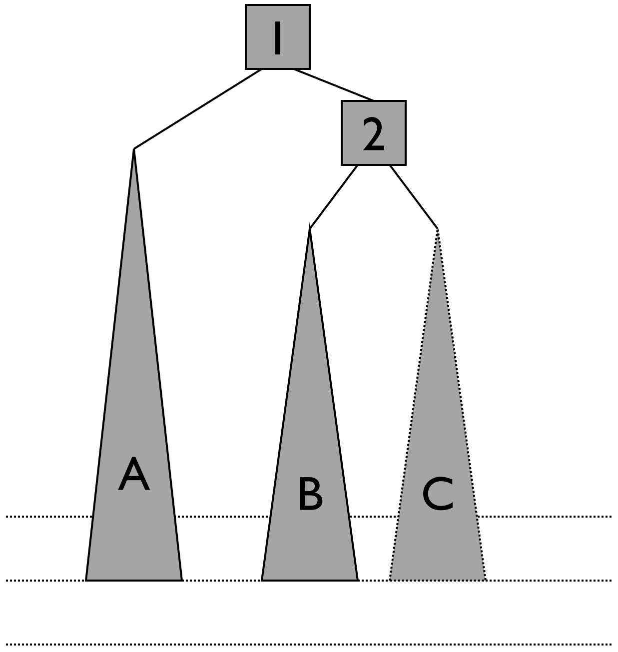
- Note the levels



- B and C have the same height
- A is one level higher
- Therefore make 1 the new root, 2 its right child and B and C the subtrees of 2

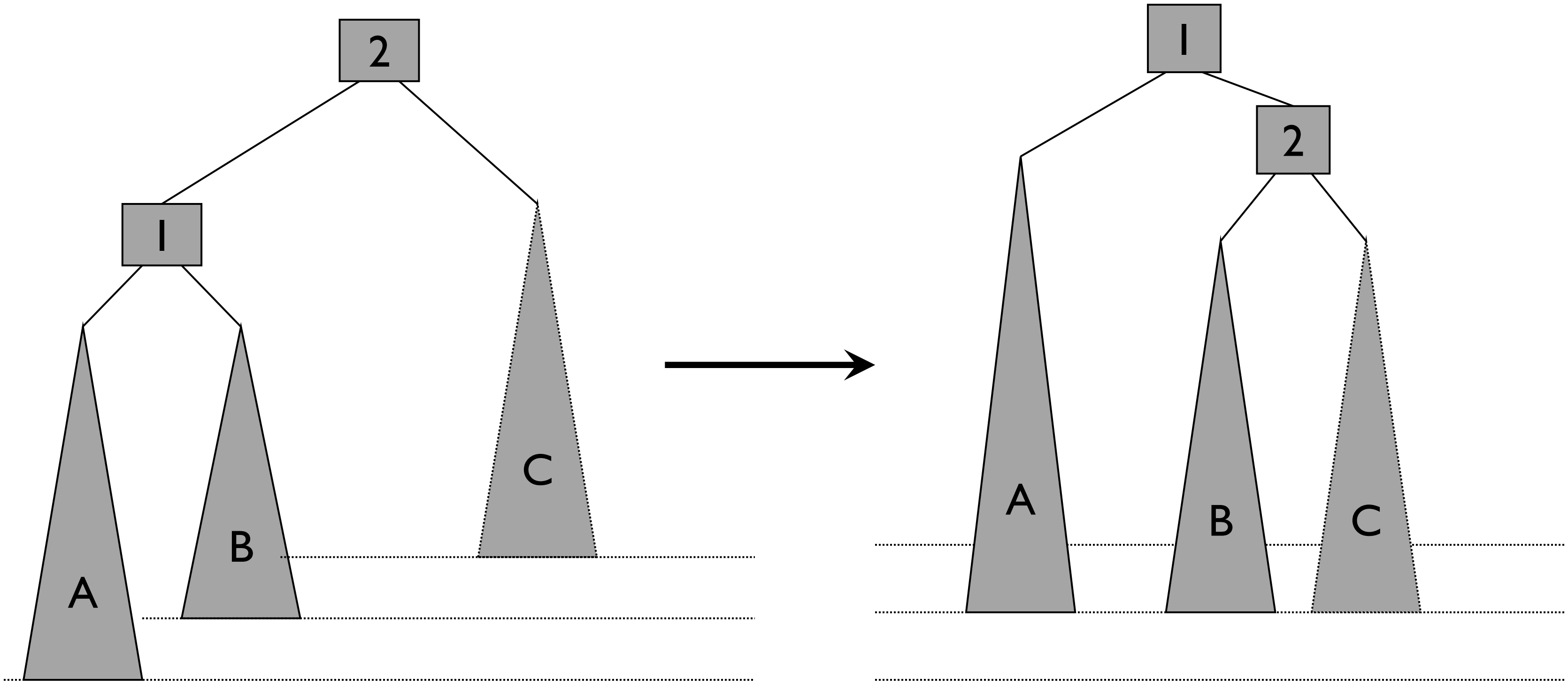
LEFT(LEFT) IMBALANCE (2) [AND RIGHT(RIGHT) IMBALANCE, BY SYMMETRY]

- Note the levels



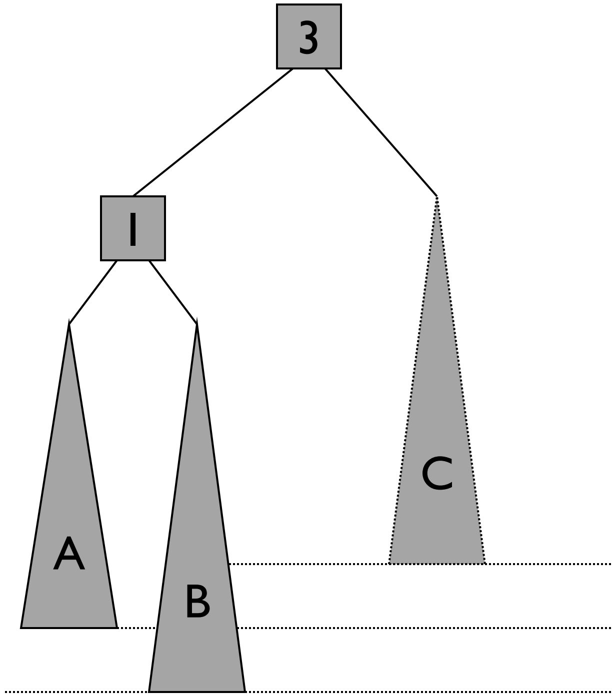
- B and C have the same height
- A is one level higher
- Therefore make 1 the new root, 2 its right child and B and C the subtrees of 2
- Result: a more balanced and legal AVL tree

SINGLE ROTATION



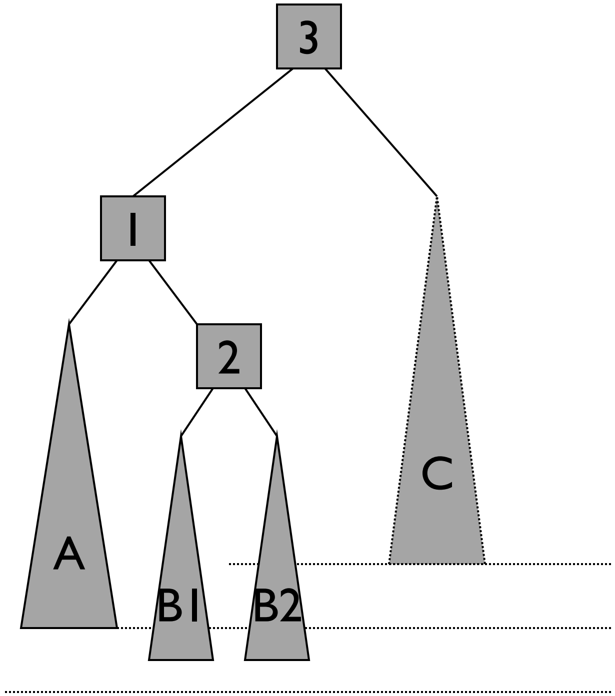
LEFT(RIGHT) IMBALANCE (1)
[AND RIGHT(LEFT) IMBALANCE BY SYMMETRY]

- Can't use the left-left balance trick - because now it's the middle subtree, i.e. B, that's too deep.
- Instead consider what's inside B...



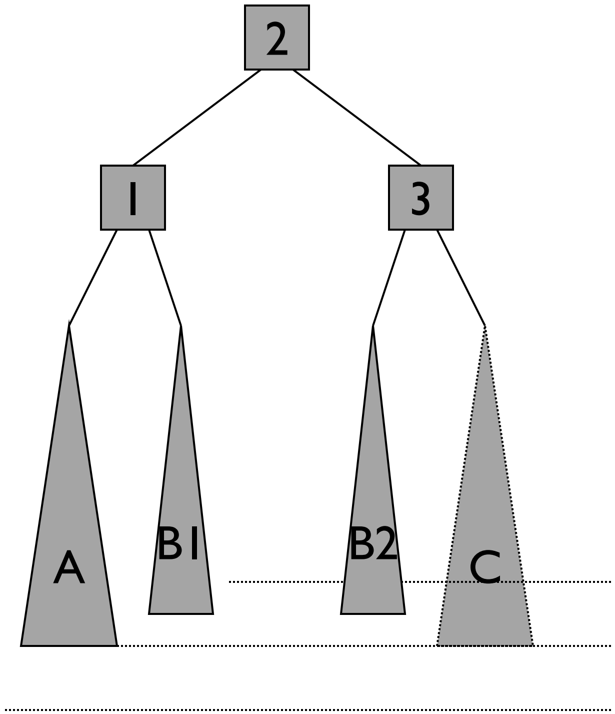
LEFT(RIGHT) IMBALANCE (2) [AND RIGHT(LEFT) IMBALANCE BY SYMMETRY]

- B will have two subtrees containing at least one item
- We do not know which is too deep - set them both to 0.5 levels below subtree A

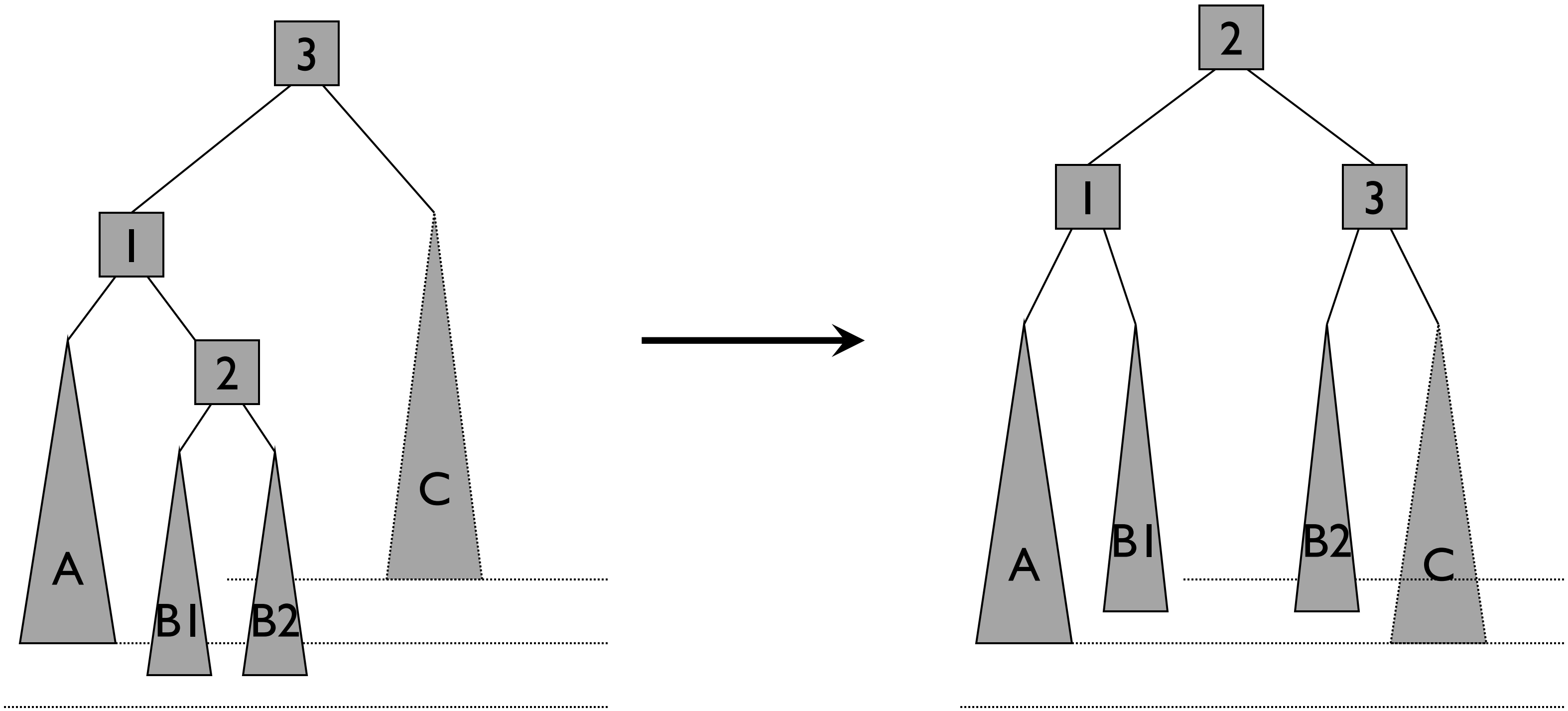


LEFT(RIGHT) IMBALANCE (3) [AND RIGHT(LEFT) IMBALANCE BY SYMMETRY]

- Neither 1 nor 3 worked as root node so make 2 the root
- Rearrange the subtrees in the correct order
- No matter how deep B1 or B2 (+/- 0.5 levels) we get a legal AVL tree again



DOUBLE ROTATION



INSERT METHOD

```
private AvlNode<Anytype> insert(Anytype x, AvlNode<Anytype> t ) {
/*1*/     if( t == null )
                t = new AvlNode<Anytype>( x, null, null );
/*2*/     else if( x.compareTo( t.element ) < 0 )
          {
                t.left = insert( x, t.left );
                if( height( t.left ) - height( t.right ) == 2 )
                      if( x.compareTo( t.left.element ) < 0 )
                            t = rotateWithLeftChild( t );
                      else
                            t = doubleWithLeftChild( t );
          }
/*3*/     else if( x.compareTo( t.element ) > 0 )
          {
                t.right = insert( x, t.right );
                if( height( t.right ) - height( t.left ) == 2 )
                      if( x.compareTo( t.right.element ) > 0 )
                            t = rotateWithRightChild( t );
                      else
                            t = doubleWithRightChild( t );
          }
/*4*/     else
                ;  // Duplicate; do nothing
          t.height = max( height( t.left ), height( t.right ) ) + 1;
          return t;
}
```

ROTATE WITH LEFT CHILD METHOD

```
private static AvlNode<Anytype> rotateWithLeftChild(
    AvlNode<Anytype> k2 )
{
        AvlNode<Anytype> k1 = k2.left;
        k2.left = k1.right;
        k1.right = k2;
        k2.height = max(height(k2.left), height(k2.right)) + 1;
        k1.height = max(height( k1.left), k2.height) + 1;
        return k1;
}
```

ROTATE WITH RIGHT CHILD METHOD

```
private static AvlNode<Anytype> rotateWithRightChild(
    AvlNode<Anytype> k1 )
{
    AvlNode<Anytype> k2 = k1.right;
    k1.right = k2.left;
    k2.left = k1;
    k1.height = max( height( k1.left ), height( k1.right ) ) + 1;
    k2.height = max( height( k2.right ), k1.height ) + 1;
    return k2;
}
```

DOUBLE WITH LEFT CHILD METHOD

```
private static AvlNode<Anytype> doubleWithLeftChild(
    AvlNode<Anytype> k3 )
{
    k3.left = rotateWithRightChild( k3.left );
    return rotateWithLeftChild( k3 );
}
```

DOUBLE WITH RIGHT CHILD METHOD

```
private static AvlNode<Anytype> doubleWithRightChild(
    AvlNode<Anytype> k1 )
{
    k1.right = rotateWithLeftChild( k1.right );
    return rotateWithRightChild( k1 );
}
```

THANKS