Lecturer: Andrew Parkes
http://www.cs.nott.ac.uk/~pszajp/

COMP2054 ADE

Binary Search Trees: Balance and Rotations

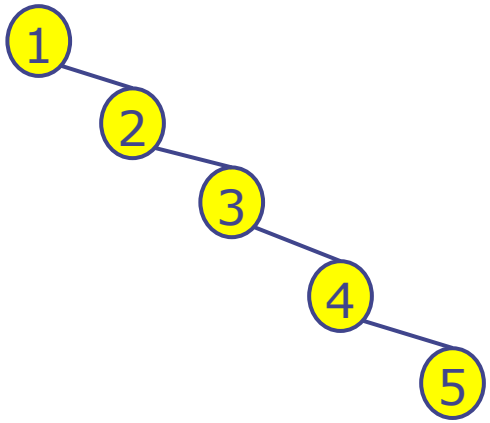
Recap: Binary Search Trees

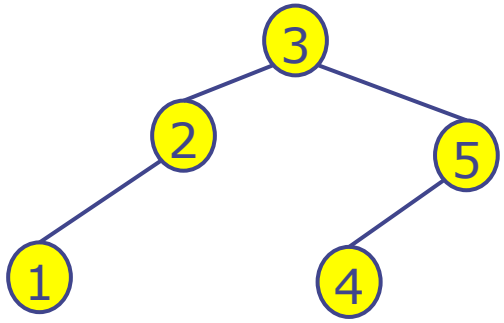
- A binary search tree (BST) is a binary tree storing key-value entries at its nodes and satisfying the "search tree" property:
  - For any node (not only the root) then
  - Nodes in the **left subtree** have (strictly) **smaller** keys
  - Nodes in the **right subtree** have (strictly) **larger** keys

Hence:

- An in-order traversal of a BST visits the keys in increasing order

The BST Imbalance problem



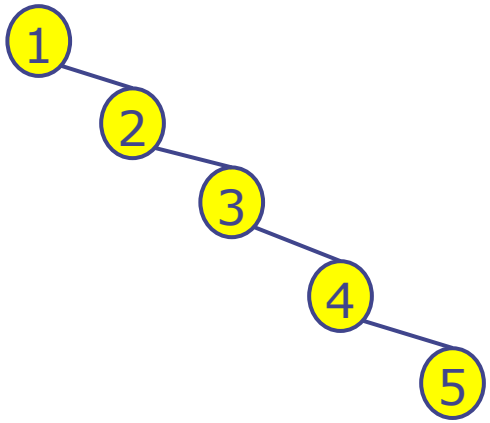


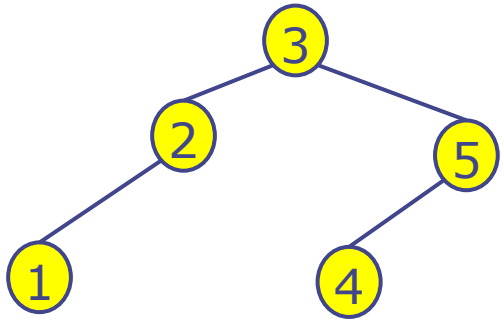
- Both trees are valid BSTs with the same content
- Both have in-order traversal: 1,2,3,4,5. But …
  - The left has height 4
  - The right has height 2
- This matters because the vital BST algorithms are O(height)
- The left tree has a larger height because it is imbalanced: the heights of the left and right subtrees are very different
- A tree is said to be "balanced" if the heights of left and right subtrees of any node are (close to) equal, and so the height is O(log n)

Recall: Issues in Self-Balancing

- Suppose you have a very imbalanced search tree, there are always corresponding balanced search trees
- Could make trees balanced using a "total rebuild"
  - But would require O(n), and so very inefficient compared to the desired O(log n)
- Re-balancing needs to be O(log n) or O(height)
- Suggests re-balancing needs to just look at the path to some recently changed node, not the entire tree
- A priori, it is not at all obvious that this is possible!

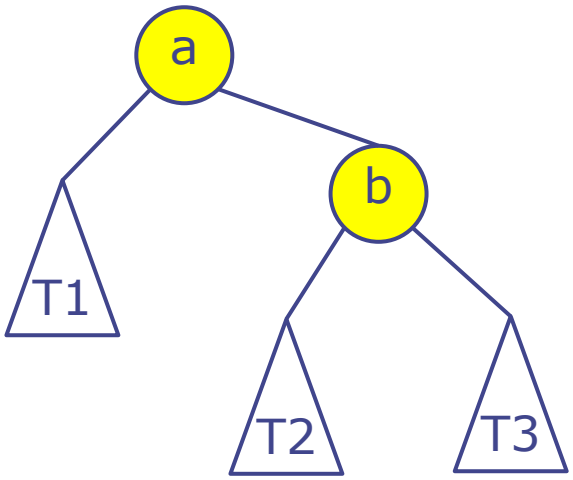
The BST Imbalance problem





- For any number of nodes, there is (obviously?) a balanced tree
  - Could balance by a "total rebuild"
  - just placing the keys into a new tree in the correct order
  - But total rebuilds would require O(n) time – doing them would destroy the O(log n) efficiency advantages of a BST.
- AIM: Do "small local rebuilds" during insert/delete operations, to maintain the balance, and so retain the O(log n) cost
- This lecture: What "small local rebuilds can we do"?

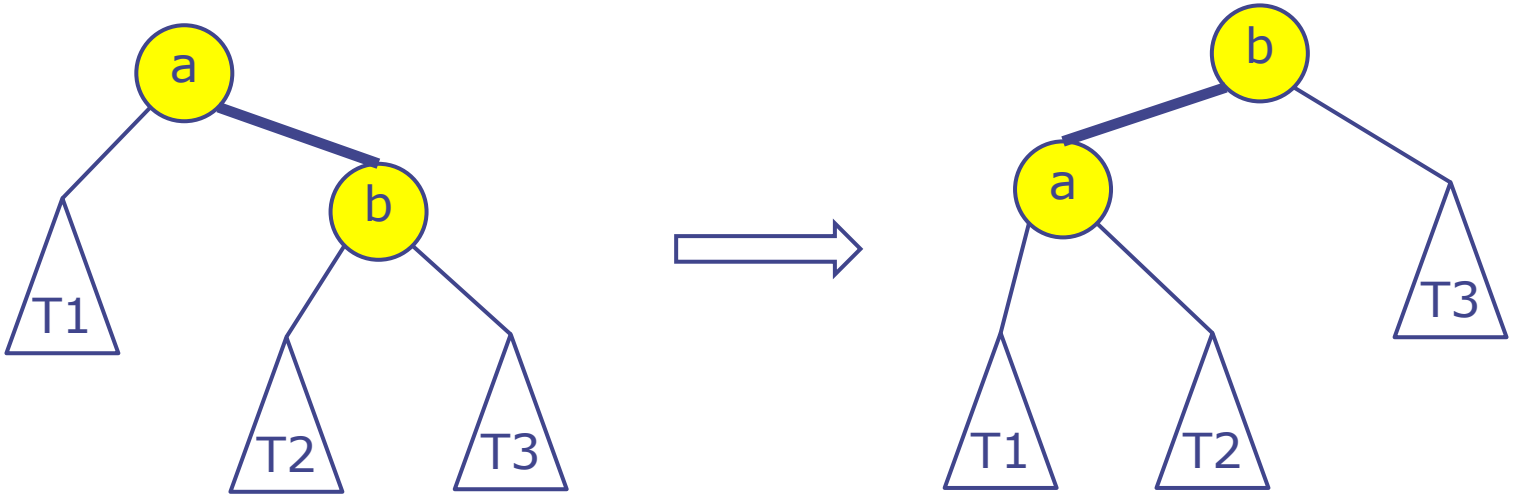
Example of a BST, T



- Where subtrees T1, T2, T3 are BSTs themselves
- Height h(T) = max(1 + h(T1), 2 + h(T2), 2 + h(T3))
- In-order traversal, "In" :

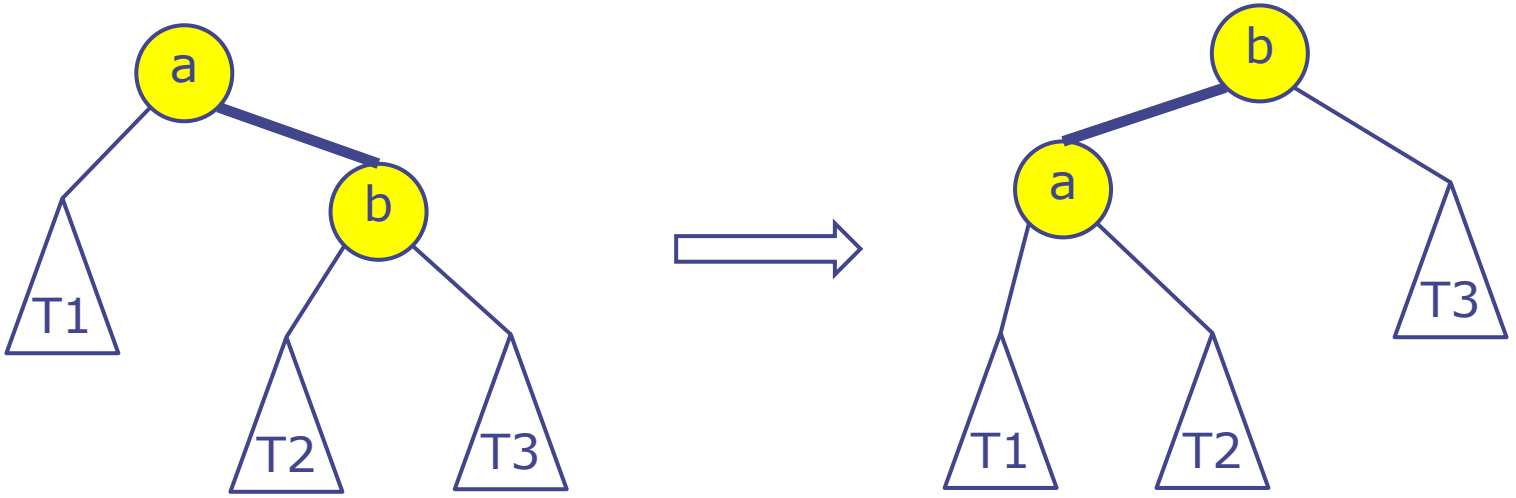
  In(T) = In(T1), a, In(T2), b, In(T3)
- Can we use the same nodes and subtrees to build a different BST?
  - But must have the same In-order Traversal !
  - Too costly to do anything internally to T1,T2,T3 themselves !

Example of a "rotation"



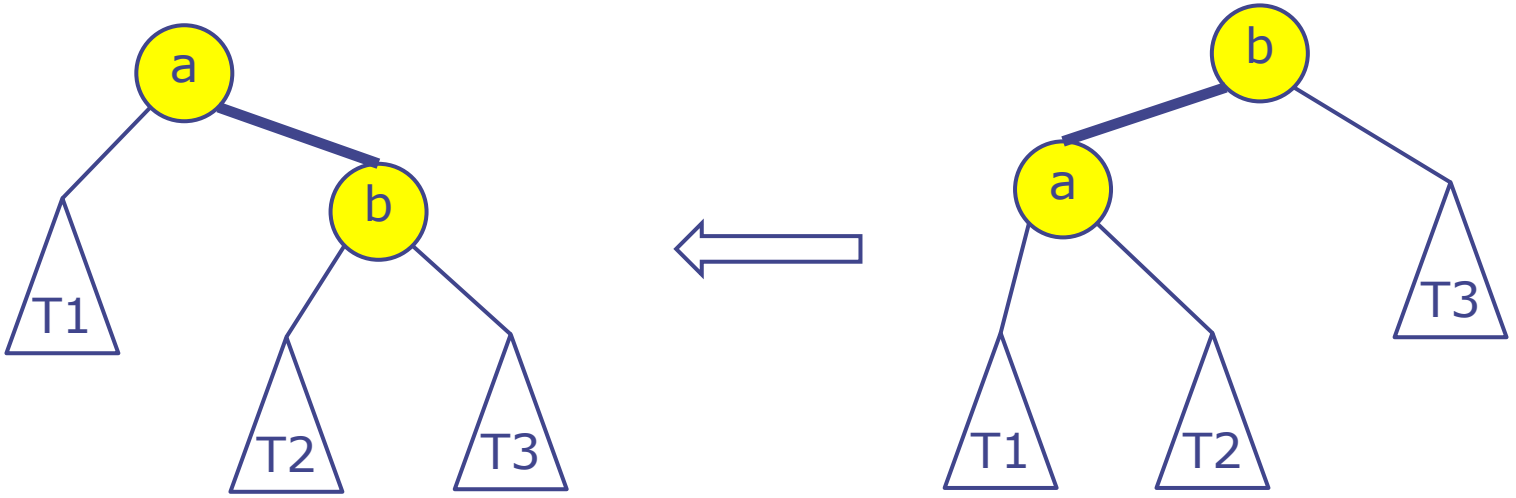
- Both trees have in-order traversal:
  In(T1), a, In(T2), b, In(T3)
- Depths:
  - Node a moved down by one
  - Node b moved up by one
  - Sub-tree T1 moved down by one
  - Sub-tree T3 moved up by one

Example of a "rotation" 2



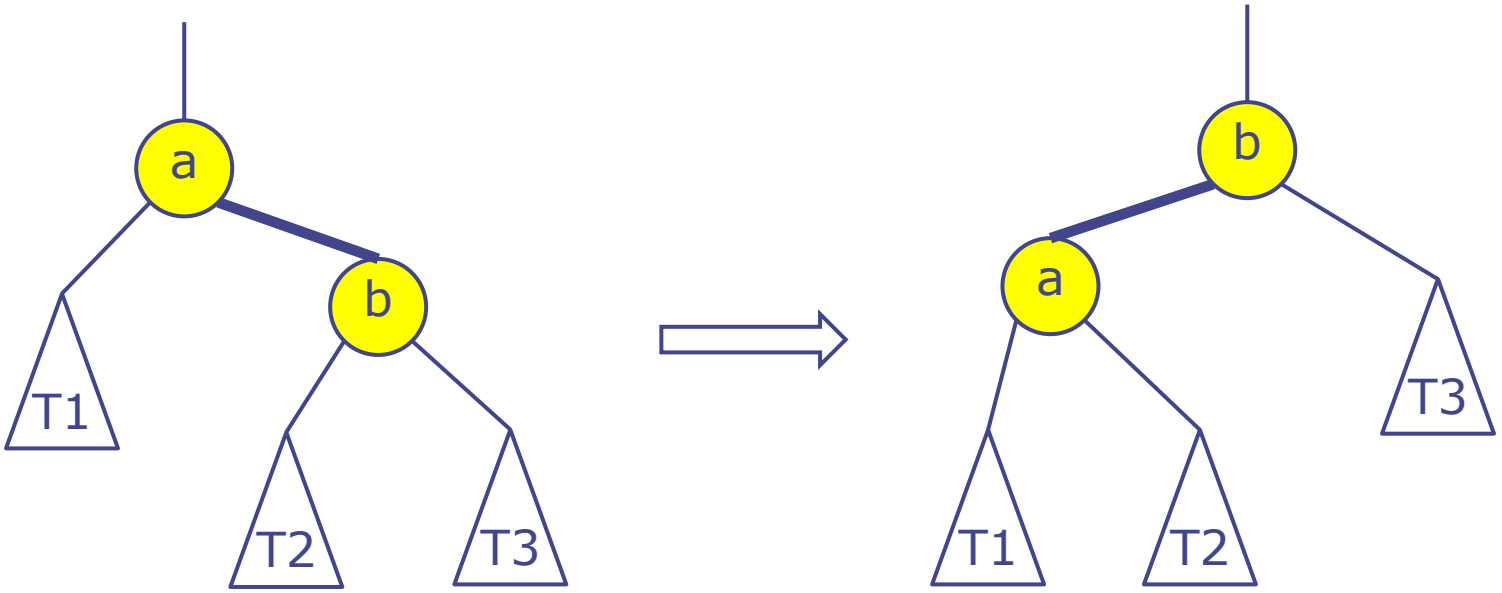
- Called a (single) "rotation" as the edge a-b rotated
- No clear convention as to whether it is a "left rotation" or a "right rotation"
  - Wiki calls it a "left rotation" around node 'a'
  - Clearer to just identify the edge, as the rotation is then fixed
- The root of the tree changed from 'a' to 'b'

Example of a "rotation" 3



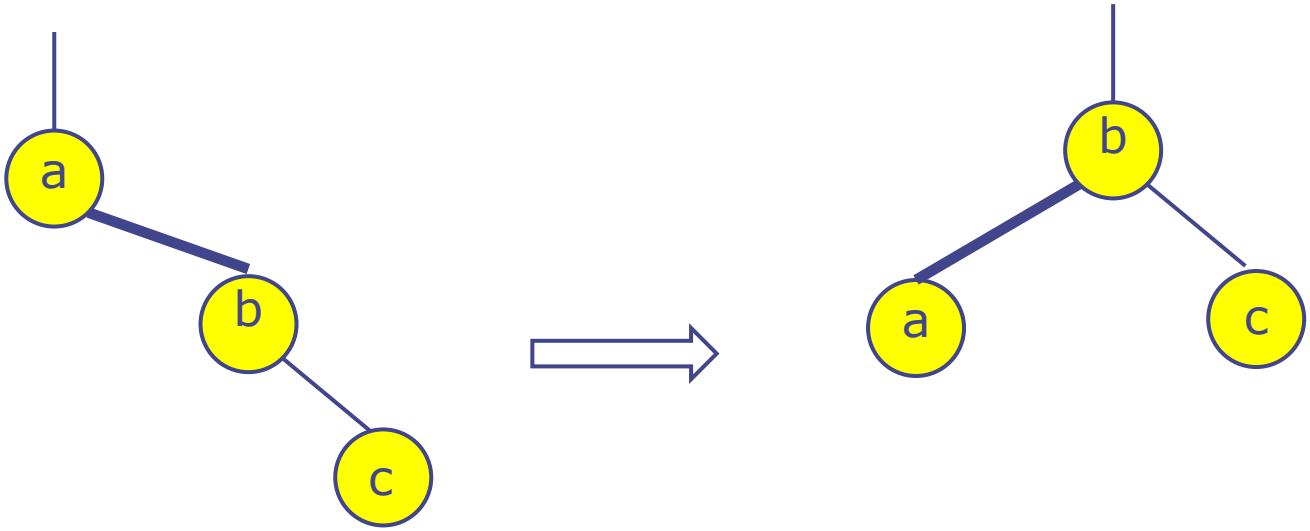
- Unsurprisingly, we can also do the reverse process
  - With the reverse effects on the depths on the sub-trees:
  - 'a' and T1 rise up,
  - 'b' and T3 sink down

Example of a "rotation" 4



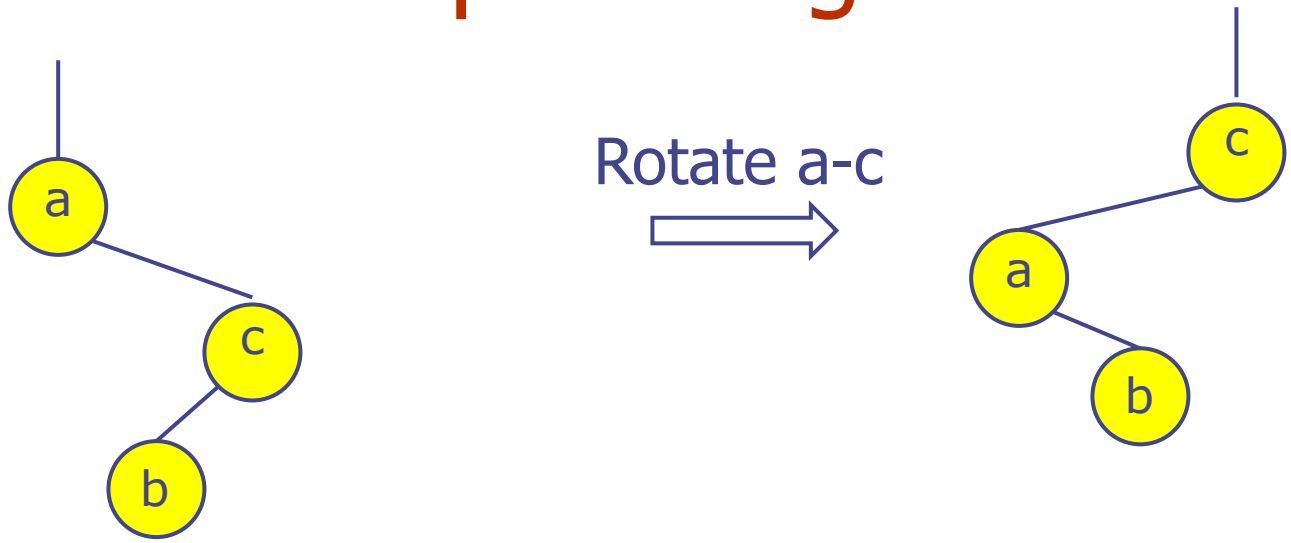
- The tree can also be a sub-tree of a BST
- The parent of a is moved to be the parent of b
- Once we have decided to rotate a specific edge then we only need to look at a fixed number of nodes and edges.
  - Hence, **a single rotation only has O(1) in cost**.

Simple Example – single rotation



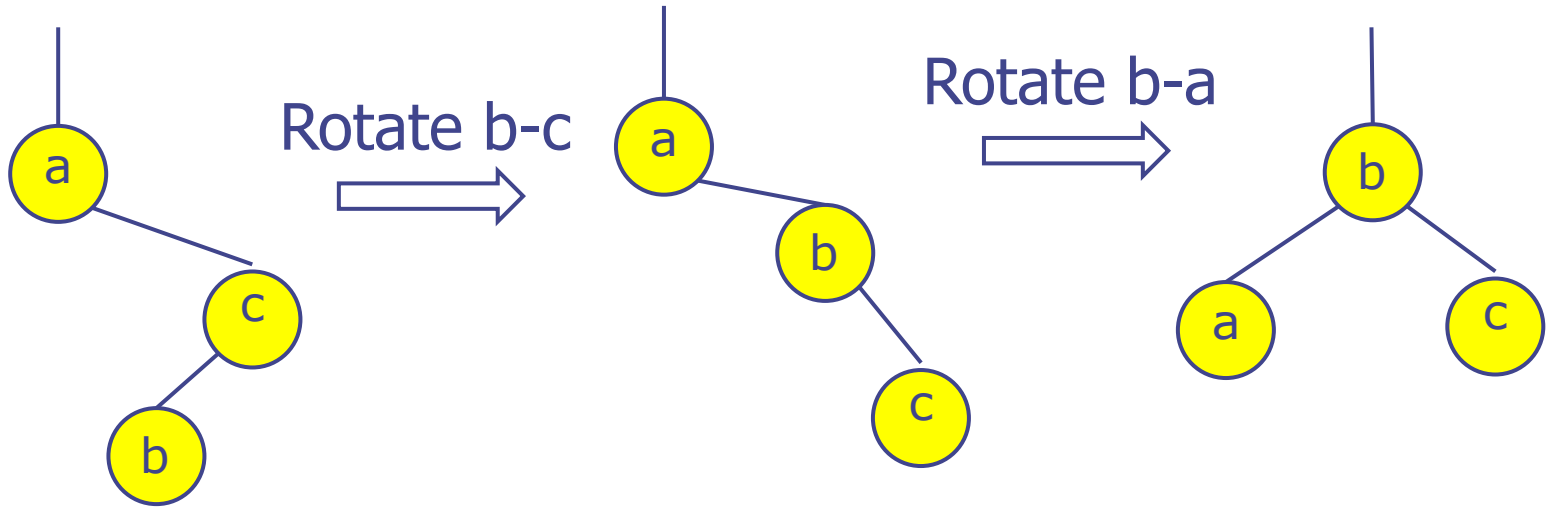
- Worst case: a chain, with a < b < c
- Rotation on the edge a-b
  - Pushes node a down, **pulls node b up** to make it the new root
  - Balances the tree
  - reduces the height from 2 to 1
- "Raising b" was the good choice because it is the median value and so should be the root

Harder Example: Single rotation fails



- "Oscillating Chain", with a < b < c
- Rotation on the edge a-c
  - Pushes node a down, pulls node c up to make it the new root
  - No effect on the height
  - Still have "oscillating chain"
  - NO USEFUL IMPROVEMENT

Specific Example: Double works



- Rotation on the edge b-c
  - Pushes node c down, pulls node b.
  - No immediate effect on the height, but now have a "straight chain"
- Then can rotate on edge a-b
  - Pulls b up again, to make it the root
  - Height is now reduced
- The median node is 'b' and so it should be at the root – so do rotations to "pull it up" (like 'up-heap') – depth of b needs to drop from 2 to 0
  - Each rotation can only move 'b' up by 1, hence need two rotations
- Coordinated "Double rotation" is sometimes needed !

Using rotations

- The goal is then to control the usage of rotations to reduce the overall height of the tree
  - Single rotations sometimes help
  - Sometimes need "double rotations" – a "coordinated pair of single rotations"
- In small examples, we can do this "by inspection"
- In general, need to be able to select rotations algorithmically

Solution: Self-Balancing

Goal of "Self-Balancing":

- Constantly do a "local restructuring" of the trees:
- Keep the trees height balanced so that the height is logarithmic in the size
- Only do rotations along the paths encountered during the BST operations
- Performance always logarithmic.

AVL Trees [[Not assessed]]

- AVL (Adelson-Velskii & Landis) trees are binary search trees where nodes also have additional information: the difference in depth between their right and left subtrees (*balance factor*). First example of balanced trees.
- For each node, the "balance factor" of that node is
  height(right subtree) – height(left subtree)
- In an AVL tree the balance of **every** node is allowed to be **only** 0,1 or -1.
- AVL trees do dynamic self-balancing in O(log n) time

Top down and bottom up insertion [[Not assessed]]

- Top down insertion algorithms make changes to the tree (necessary to keep the tree balanced) as they search for the place to insert the item. They make one pass through the tree.
- Bottom up: first insert the item, and then work back through the tree making changes. Less efficient because make two passes through the tree.

Examples [Not assessed]

- Bottom up: AVL trees
  - Details not required this session!
- Top down: 2-4 trees & red-black trees
  - Details not required this session!

But should know that they are designed to help keep the trees balanced in order that the height stays small.
They guarantee O(log n) – unlike Hashmaps

Minimal Expectations

- The definitions of (single) rotations
- Examples of their usage to reduce the height, in some small(ish) trees
- Manually selecting the rotations