AVL Trees

- Better Balanced Binary Search Trees
- AVL Trees
- AVL Tree Examples
- AVL Insertion Algorithm
- Maintaining Balance/Height
- Searching AVL Trees
- Performance of AVL Trees

❖ Better Balanced Binary Search Trees

So far, we have seen ...

- randomised trees ... make poor performance unlikely
- occasional rebalance ... fix balance periodically
- splay trees ... reasonable amortized performance
- but all types still have *O(n)* worst case

Ideally, we want both average/worst case to be *O(log n)*

- AVL trees ... fix imbalances as soon as they occur
- 2-3-4 trees ... use varying-sized nodes to assist balance
- red-black trees ... isomorphic to 2-3-4, but binary nodes

❖ AVL Trees

Invented by Georgy Adelson-Velsky and Evgenii Landis (1962)

Goal:

- tree remains reasonably well-balanced *O(log n)*
- cost of fixing imbalance is relatively cheap

Approach:

- insertion (at leaves) may cause imbalance
- repair balance as soon as we notice imbalance
- repairs done locally, not by overall tree restructure

❖ ... AVL Trees

A tree is unbalanced when abs(height(left)-height(right)) > 1

left ≠ right

This can be repaired by rotation:

- if left subtree too deep, rotate right
- if right subtree too deep, rotate left

Problem: determining height/depth of subtrees is expensive

- need to traverse whole subtree to find longest path

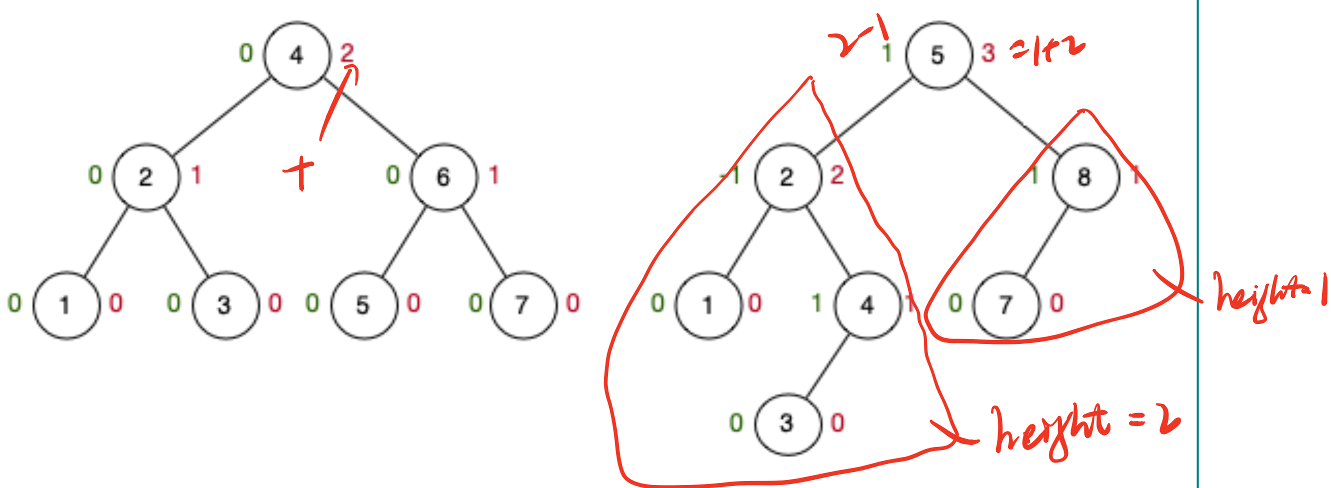
Solution: store balance data in each node (either height or balance)

- but extra effort needed to maintain this data on insertion

❖ AVL Tree Examples

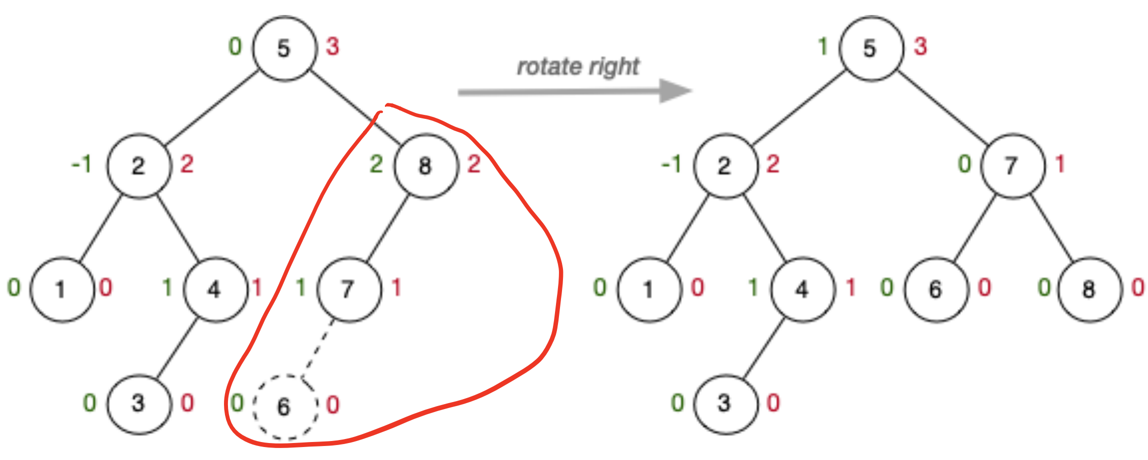
Red numbers are height; green numbers are balance

difference in height



❖ ... AVL Tree Examples

How an unbalanced tree can be rebalanced



Not AVL once 6 inserted

Rotation restores balance

① 6>5 5 t ↘ AVL(8, 6)	LH=2 RH=2 balance
6<8 8 t ↙ AVL(7, 6)	LH=2 RH=0 LH−RH=2 >1 rotate right
6<7 7 t ↙ AVL(Null, 6)	balance LH=1 RH=0. LH−RH=1
t=Null insert 6	balance LH=RH=0.

❖ AVL Insertion Algorithm

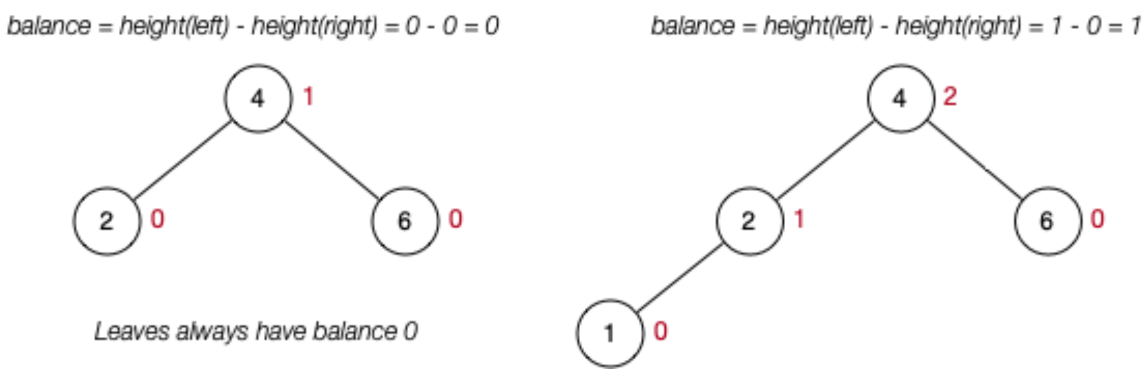
Implementation of AVL insertion

```
insertAVL(tree,item):
|  Input  tree, item
|  Output tree with item AVL-inserted
|
|  if tree is empty then
|     return new node containing item
|  else if item = data(tree) then   duplicate
|     return tree
|  else
|  |  if item < data(tree) then
|  |     left(tree) = insertAVL(left(tree),item)
|  |  else if item > data(tree) then
|  |     right(tree) = insertAVL(right(tree),item)
|  |  end if
|  |  LHeight = height(left(tree))
|  |  RHeight = height(right(tree))
|  |  if (LHeight - RHeight) > 1 then
|  |     if item > data(left(tree)) then
|  |        left(tree) = rotateLeft(left(tree))
|  |     end if
|  |     tree=rotateRight(tree)
|  |  else if (RHeight - LHeight) > 1 then
|  |     if item < data(right(tree)) then
|  |        right(tree) = rotateRight(right(tree))
|  |     end if
|  |     tree=rotateLeft(tree)
|  |  end if
|  |  return tree
|  end if
```

check balance or not

❖ Maintaining Balance/Height

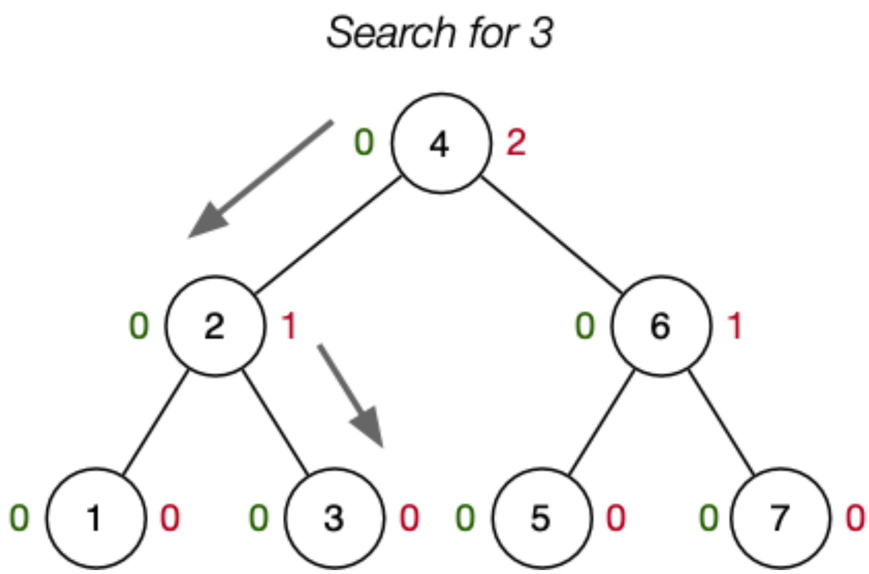
Store height in nodes; update on insertion; compute balance



If abs(balance) > 1 after updating, rebalance via rotation

❖ Searching AVL Trees

Exactly the same as for regular BSTs.



Height/balance measures are ignored

❖ Performance of AVL Trees

Analysis of AVL trees:

- trees are height-balanced; subtree depths differ by +/-1
- average/worst-case search performance of *O(log n)*
- *require* extra data to be stored in each node (efficiency)
- *require* extra data to be maintained during insertion
- may not be weight-balanced; subtree sizes may differ