COMP20007 Design of Algorithms

Binary Search Trees - Part 1

Daniel Beck

Lecture 17

Semester 1, 2023

Dictionaries

- Abstract Data Structure
- Collection of (key, value) pairs
  - Values are usually records, such as my videogames or the Unimelb students.
  - Keys are (unique) identifiers, such as the name of a game or the student ID.
- Required operations:
  - Search for a value (given a key)
  - Insert a new pair
  - Delete an existent pair (given a key)

Dictionaries - Implementations

Unsorted array / Linked list

- Search: $\Theta(n)$ comparisons;

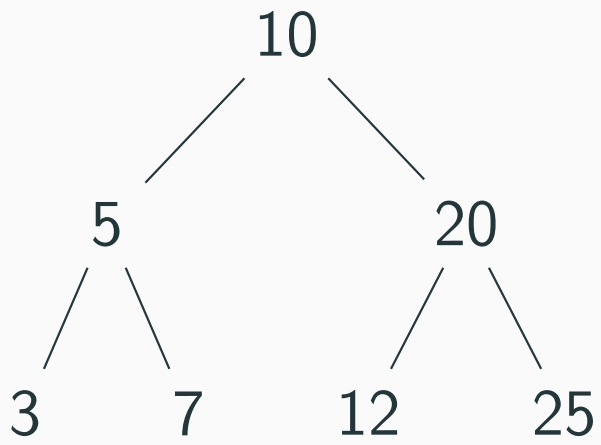
Sorted array

- Search: $\Theta(\log n)$ comparisons;
- Insert/Delete: $\Theta(n)$ record swaps;

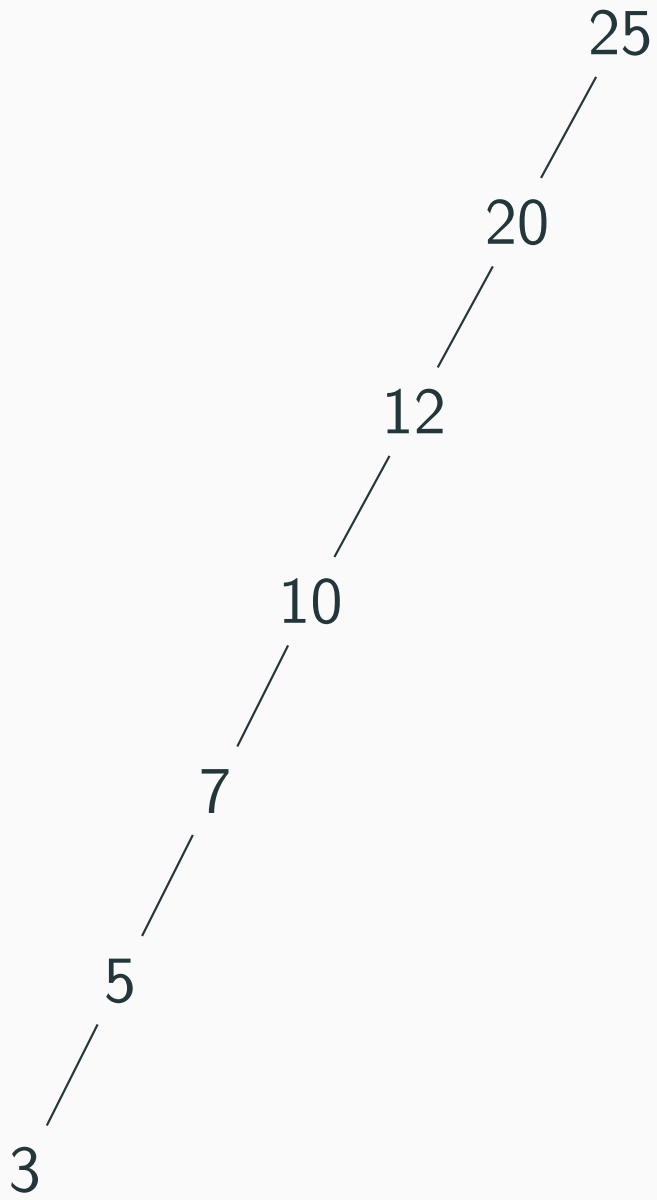
This lecture: a better data structure

- Search: $\Theta(\log n)$ comparisons;
- Insert/Delete: $\Theta(\log n)$ record swaps;

Binary Search Tree



Binary Search Tree - Worst Case



BST - How to avoid degeneracy?

Two options:

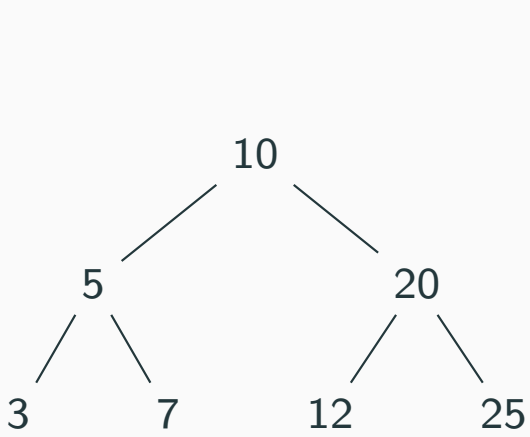
- Self-balancing
  - **AVL trees**
  - Red-black trees
  - Splay trees
- Change the representation
  - **2-3 trees**
  - 2-3-4 trees
  - B-trees

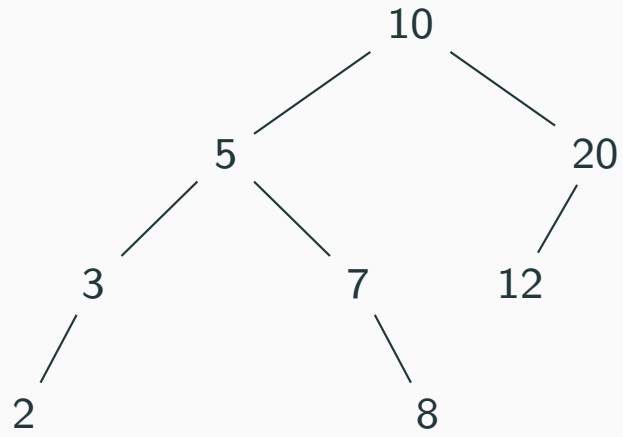
AVL trees

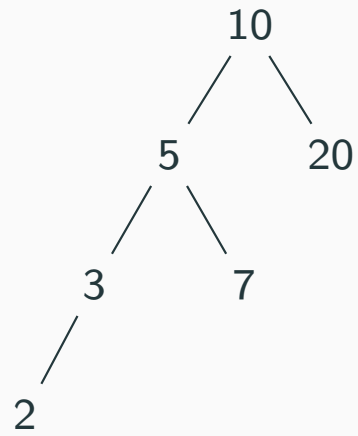
- Named after Adelson-Velsky and Landis.
- A BST where each node has a *balance factor*: the difference in height between the left and right subtrees.
- When the balance factor becomes 2 or -2, *rotate* the tree to adjust them.

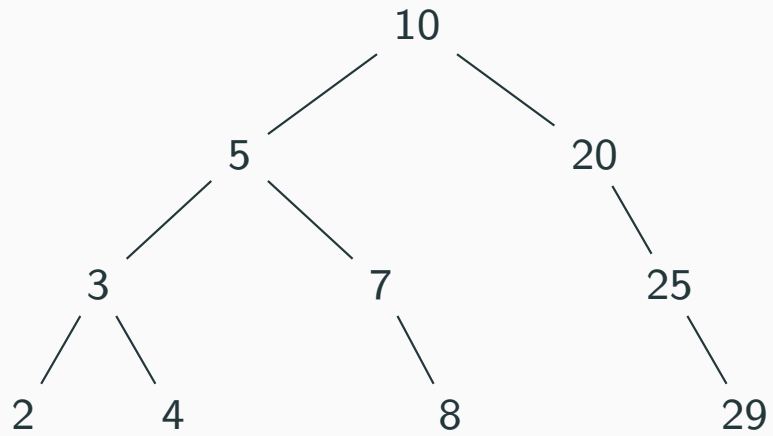
AVL Trees: Examples and Counter-Examples

Which of these are AVL trees?





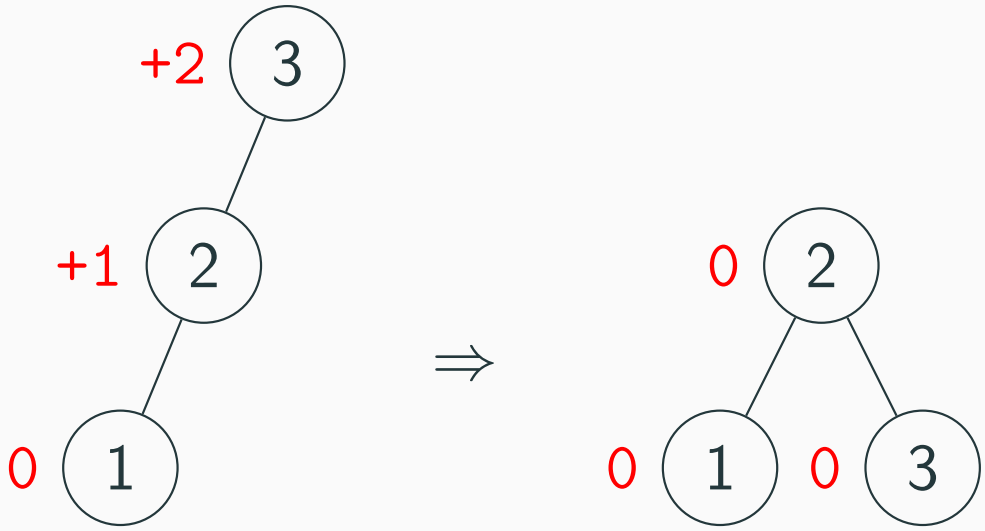




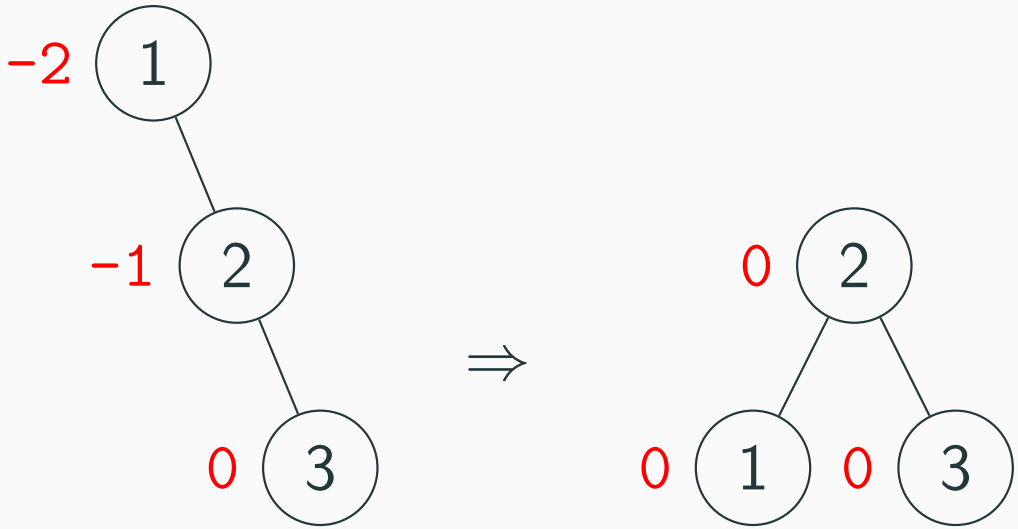
AVL Trees - Rotations

- Search is done as in BSTs.
- Insertion and Deletion also done as in BSTs, with additional steps at the end.
  - Update balance factors.
  - If the tree becomes unbalanced, perform *rotations* to rebalance it.

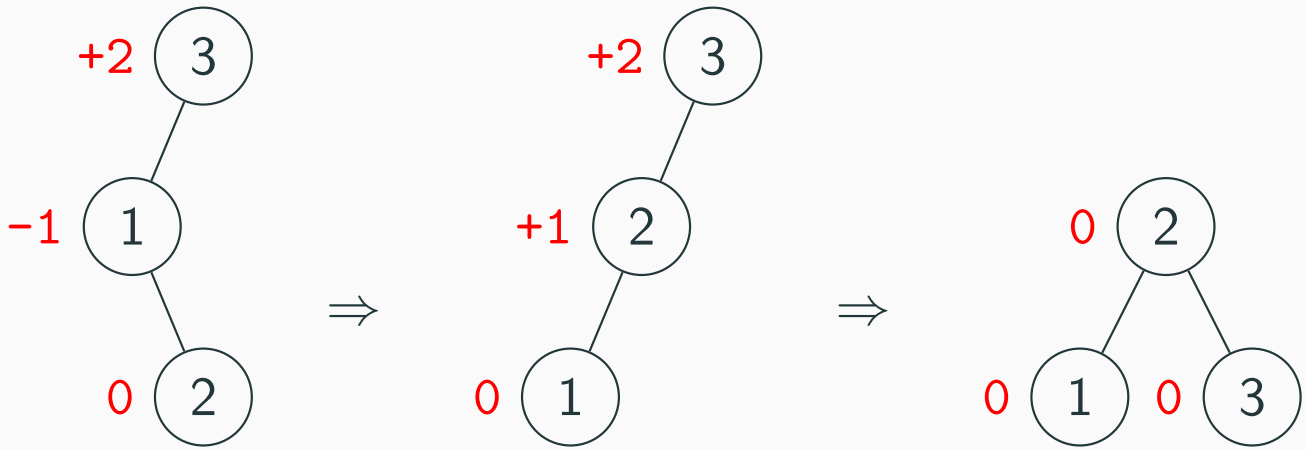
AVL Trees: R-Rotation



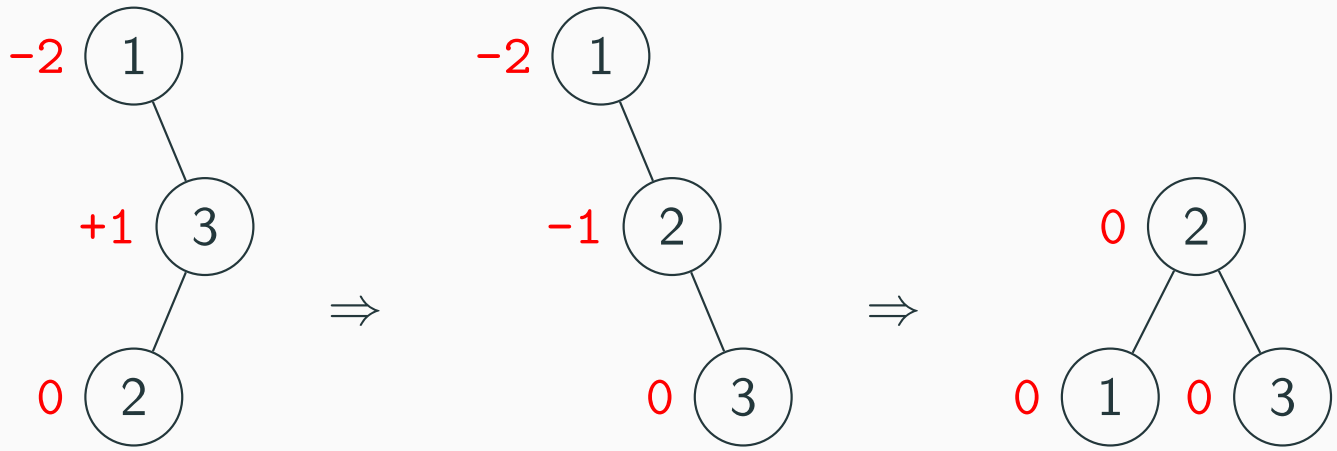
AVL Trees: L-Rotation



AVL Trees: LR-Rotation

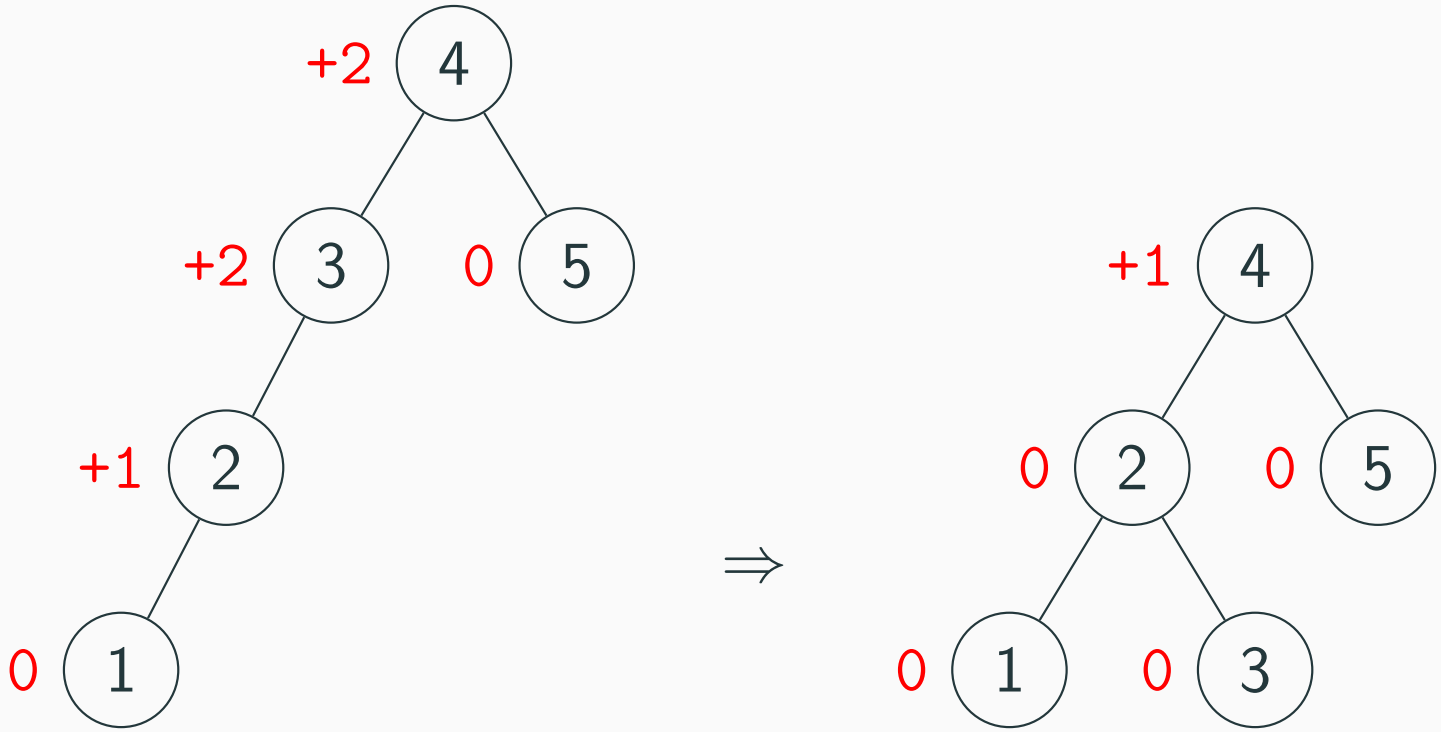


AVL Trees: RL-Rotation



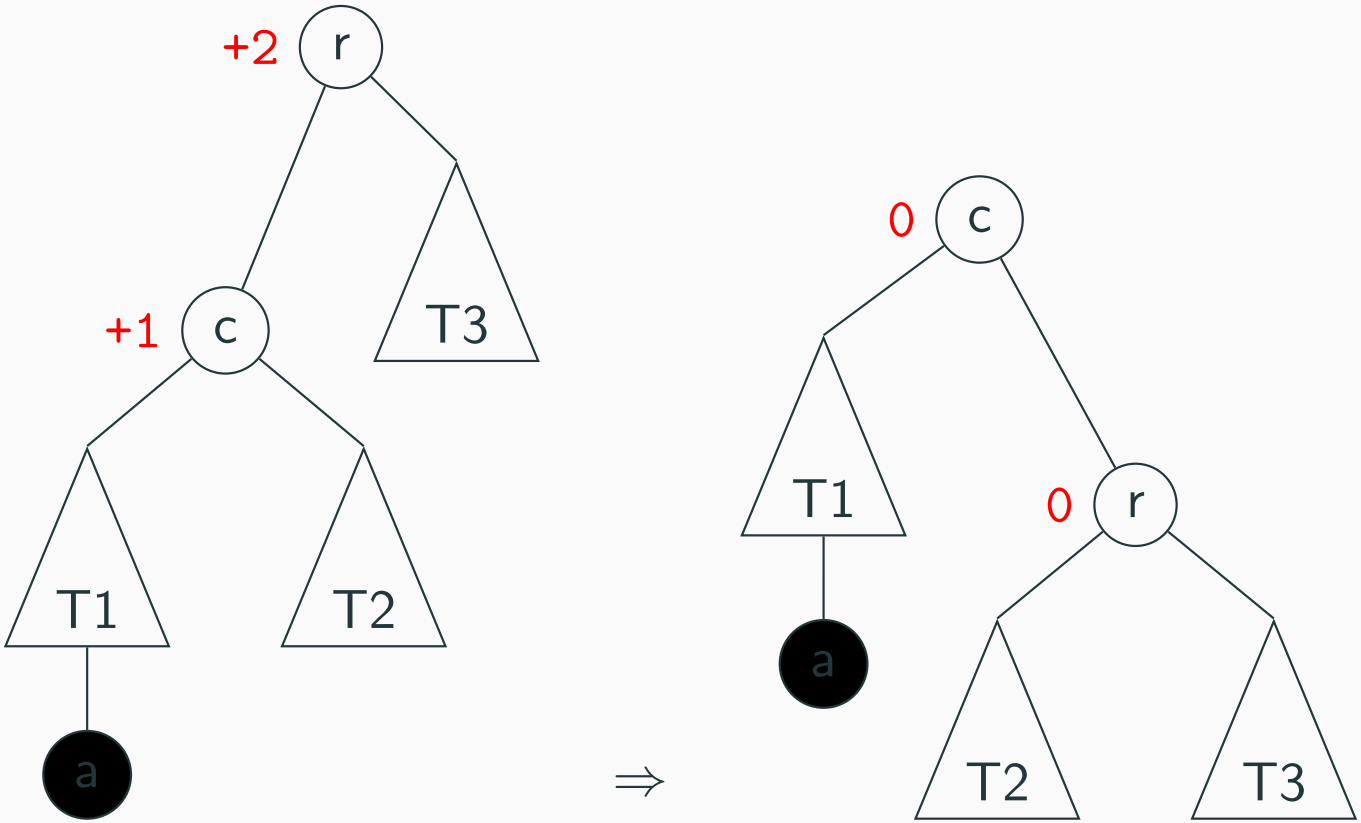
AVL Trees: Where to Perform the Rotation

Along an unbalanced path, we may have several nodes with balance factor 2 (or -2):



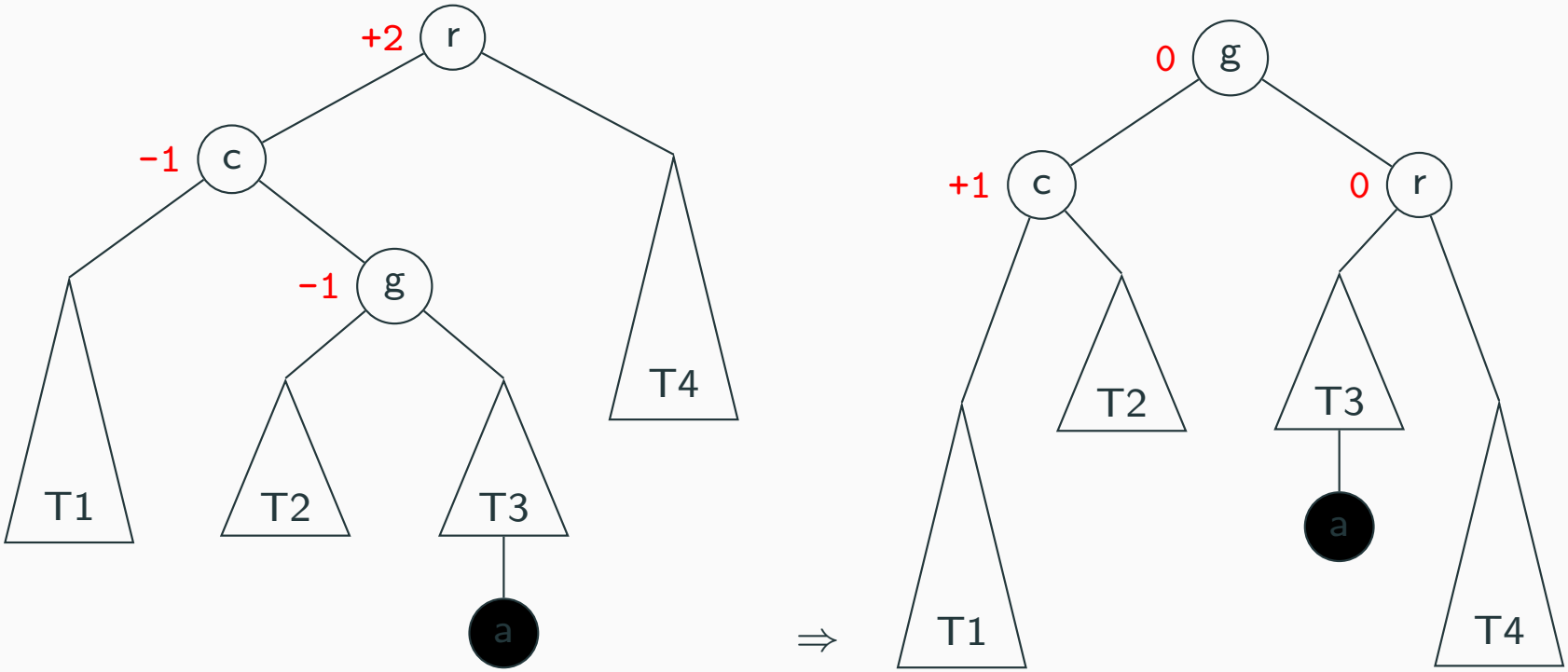
It is always the lowest unbalanced subtree that is re-balanced.

AVL Trees: The Single Rotation, Generally



This shows an R-rotation; an L-rotation is similar.

AVL Trees: The Double Rotation, Generally



This shows an LR-rotation; an RL-rotation is similar.

Properties of AVL Trees

- Rotations ensure that an AVL tree is always balanced.
- An AVL tree with n nodes has depth $\Theta(\log n)$.
- This ensures all three operations are $\Theta(\log n)$.

Self-balancing trees - In practice

- AVL Trees are just one example of self-balancing trees.
- An alternative that is widely used is Red-Black trees (not covered).
    - Main idea: longest path is twice as long as the shortest path in the worst case (as opposed to +1 or -1 cases in AVL trees).
    - Real-world example: C++ *maps* (which implement dictionaries) use red-black trees.

Self-balancing trees - In practice

- AVL trees are "more balanced" but require more frequent rotations.
    - Better if searches are more frequent than insertions/deletions.
- Red-black trees are "less balanced" but require less rotations.
    - Better if insertions/deletions are more frequent than searches.

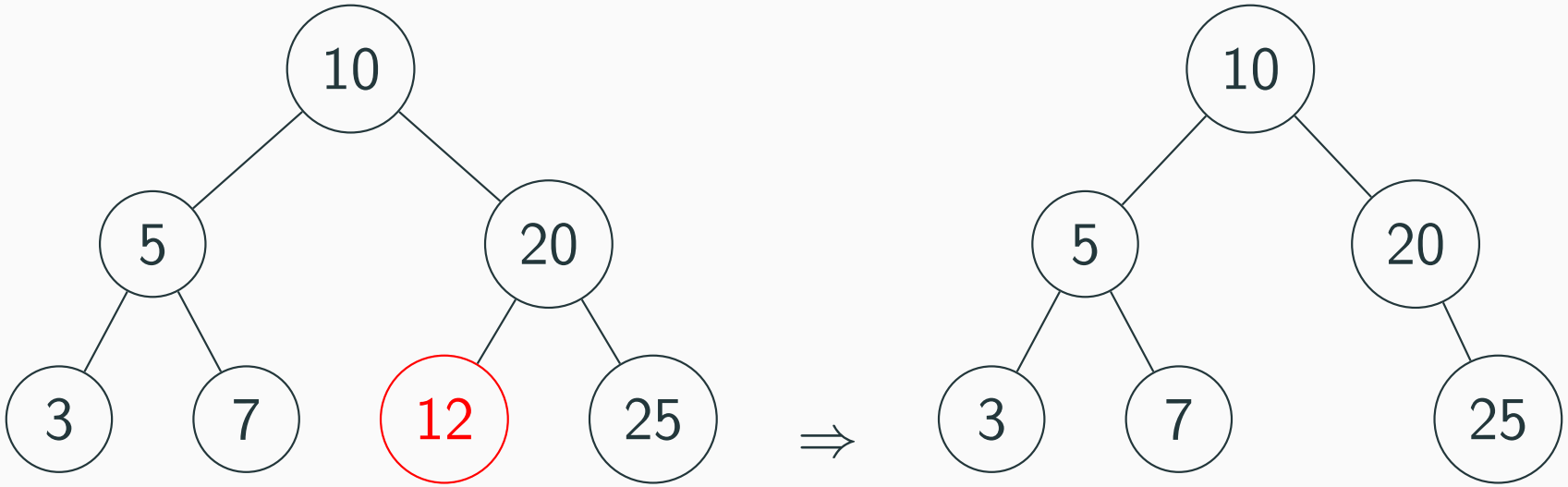
Summary

- Dictionaries are a collection of (key, value) pairs.
- Three main operations: search, insert and delete.
- Standard BSTs can get unbalanced: performance degradation.
- Self-balancing trees guarantee $\Theta(\log n)$ performance.

Next lecture: balancing BSTs through representation changes.

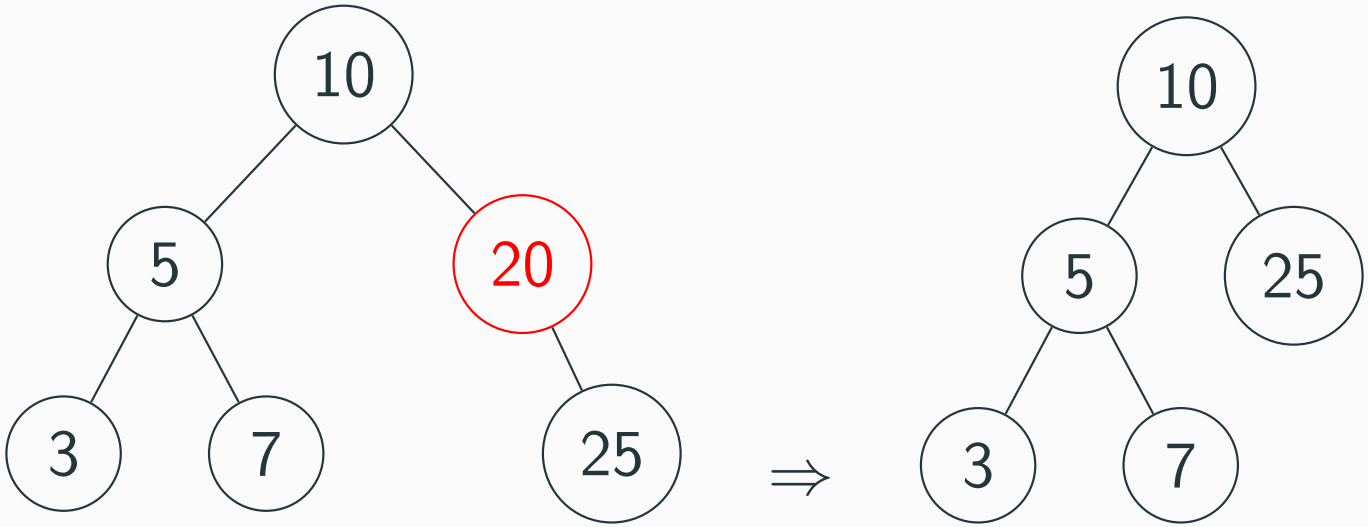
Deletions in a BST

Case 1: node is a leaf. Simply remove the node.



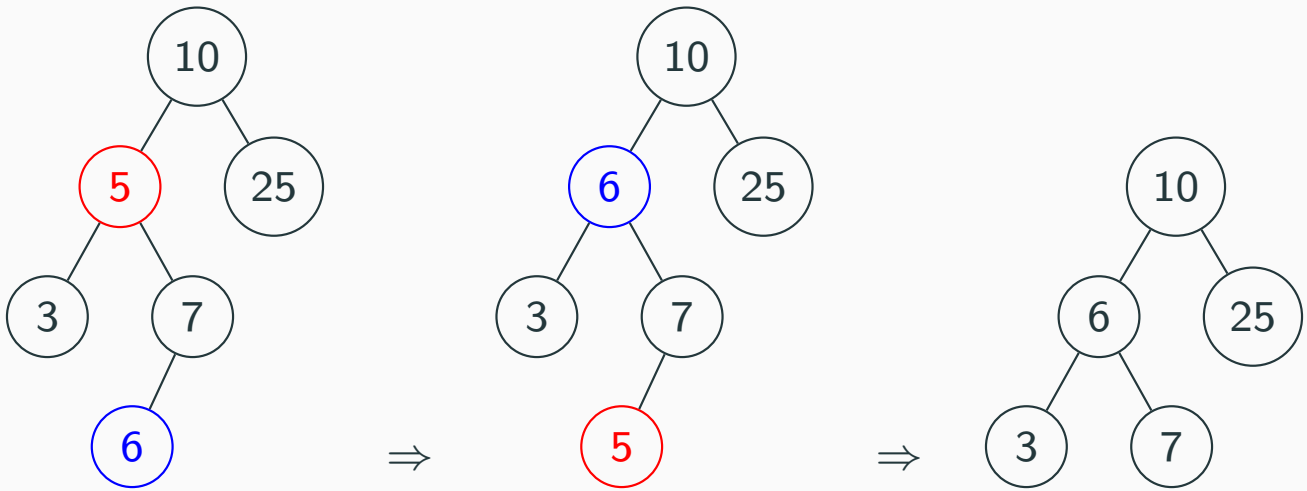
Deletions in a BST

Case 2: node has 1 child. Replace node with child.



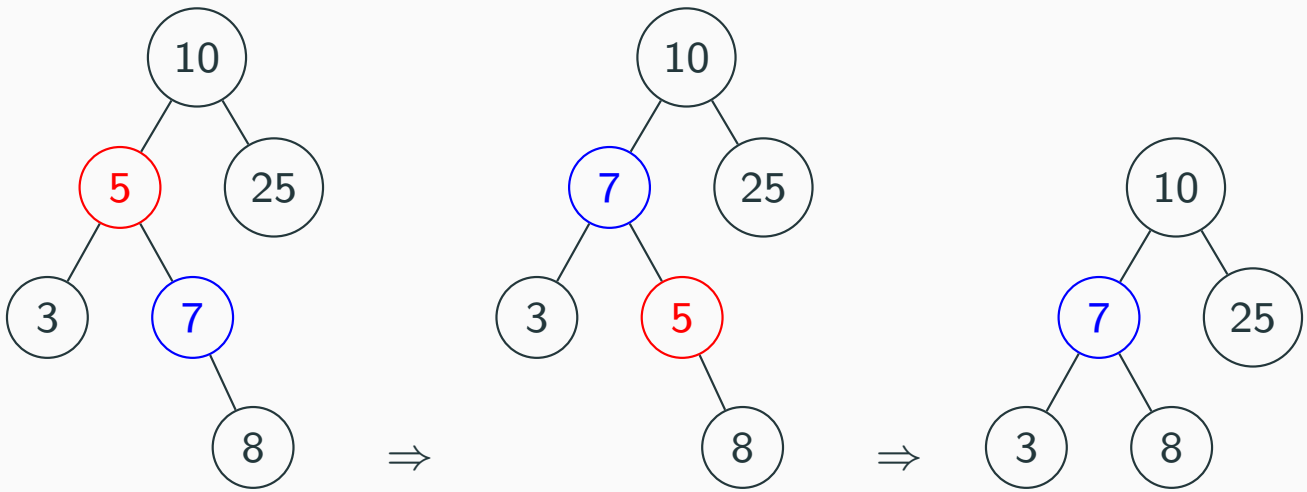
Deletions in a BST

Case 3: node has 2 children. Swap node with immediate successor (or predecessor). Then delete the node, triggering Case 1 or 2.



Deletions in a BST

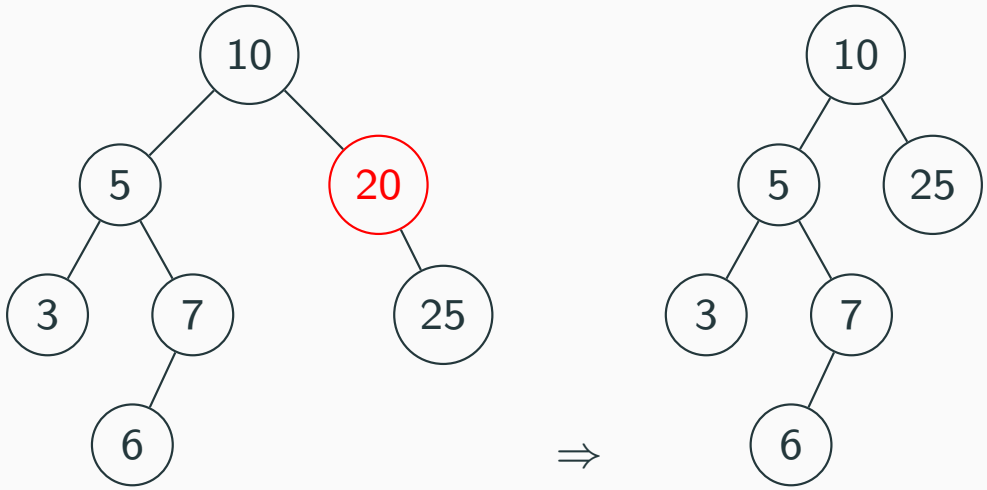
Case 3: another example



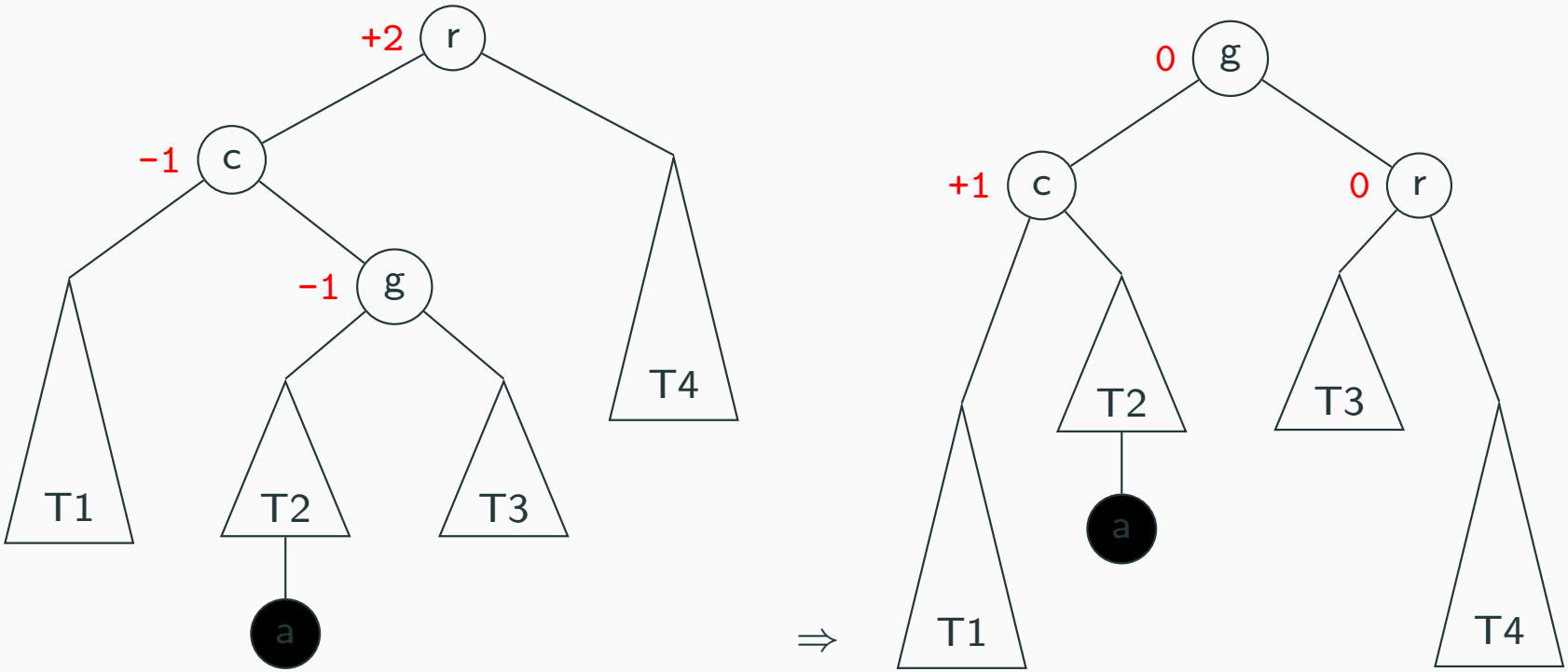
Deletions in an AVL-tree

Same as BST, but two extra steps:

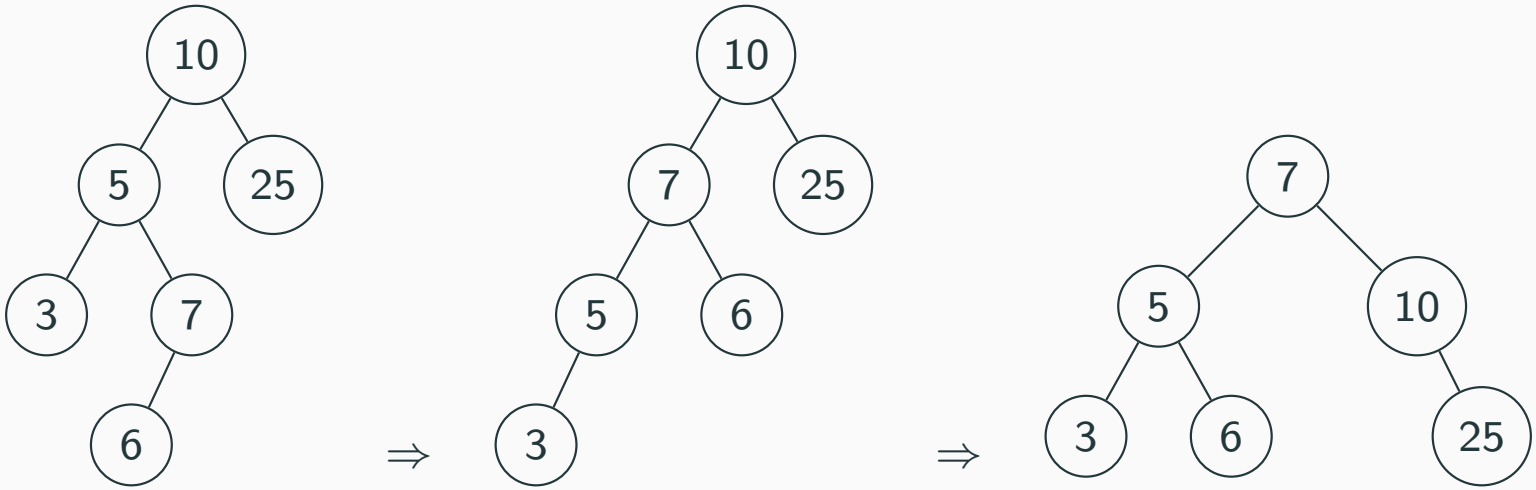
- Update heights starting from deleted node and upwards.
- Rotate at each node when required.



AVL Trees: The Double Rotation, Generally



Deletions in an AVL-tree



COMP20007 Design of Algorithms

Binary Search Trees - Part 2

Daniel Beck

Lecture 18

Semester 1, 2023

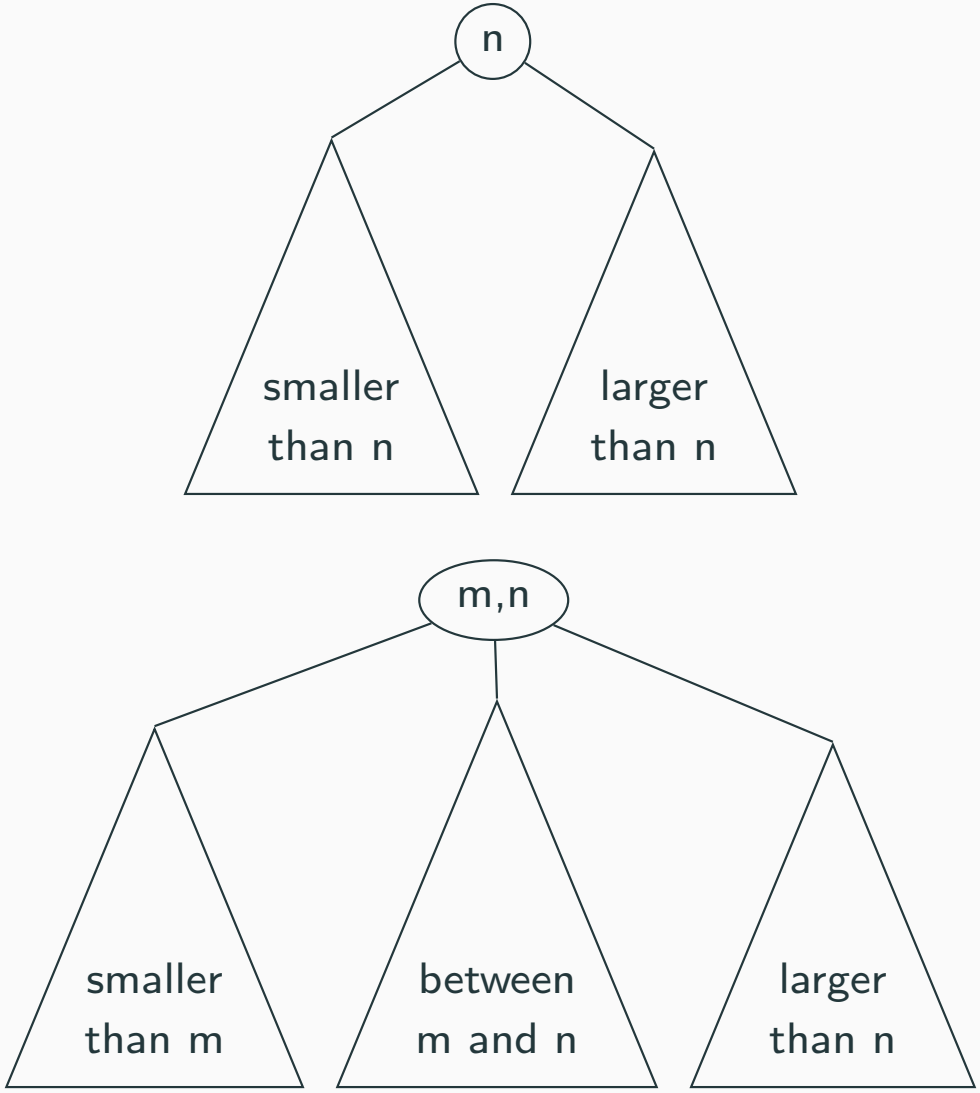
Dictionaries - Summary

- Dictionaries are a collection of (key, value) pairs.
- Three main operations: search, insert and delete.
- Standard BSTs can get unbalanced: performance degradation.
- Self-balancing trees guarantee $\Theta(\log n)$ performance.

Representational Changes

- Same goal: keeping the tree balanced.
- But instead of relying on rotations, we will allow multiple elements per node and multiple children per node.
- 2–3 trees: contains only 2-nodes and 3-nodes.
  - A 2-node contains one element and at most two children (as in BSTs)
  - A 3-node contains two elements and at most three children.
- Easy way to keep the tree balanced.
- Can be extended in many ways: 2–3–4 trees, B-trees, etc.

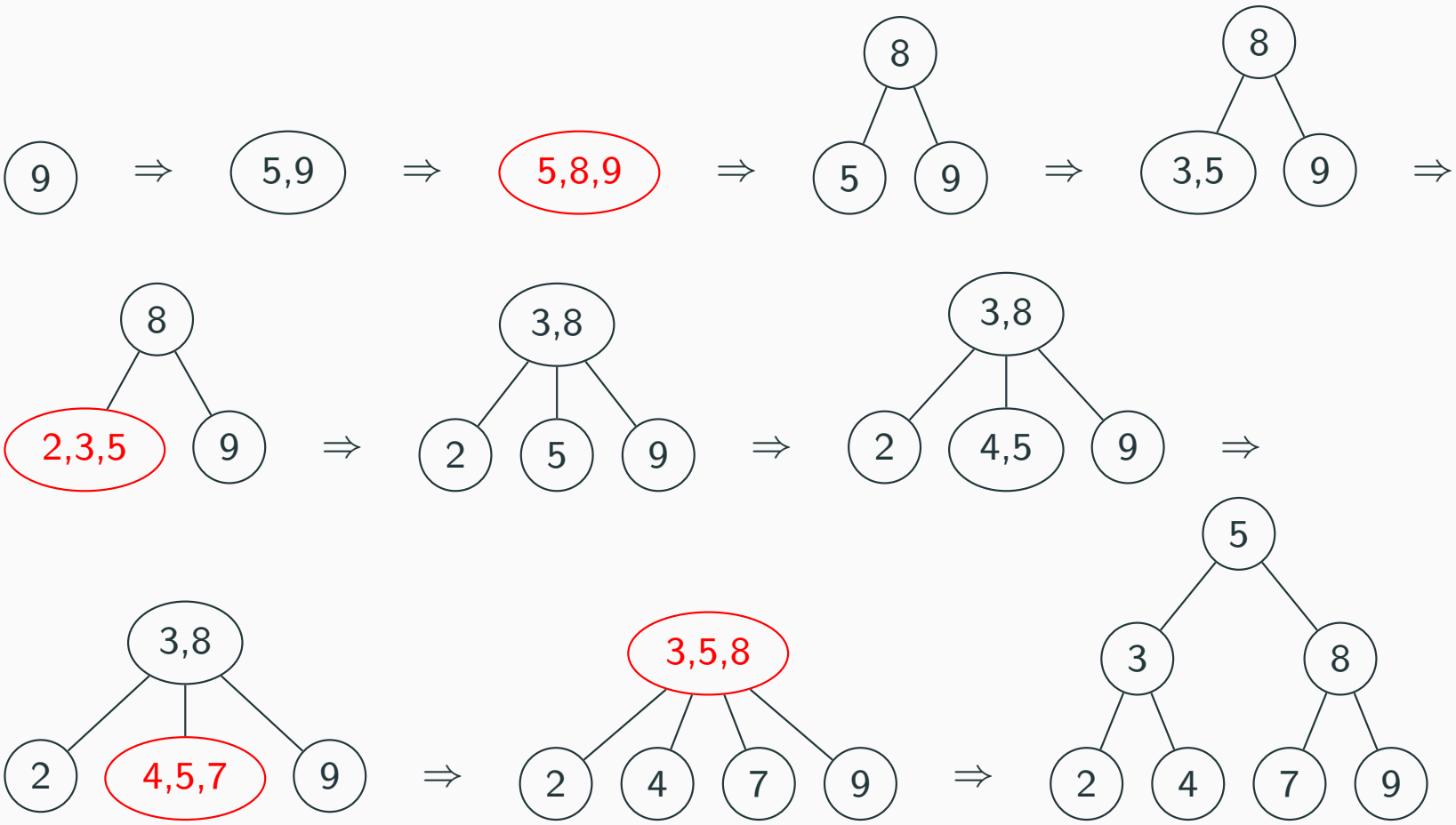
2-Nodes and 3-Nodes



Insertion in a 2–3 Tree

- As in a BST, assume that we are searching for k.
- If the leaf node is a 2-node, insert k, becoming a 3-node.
- Otherwise, momentarily form a node with three elements:
    - In sorted order, call them k_1, k_2, and k_3.
    - Split the node, so that k_1 and k_3 form their own individual 2-nodes, and k_2 is promoted to the parent node.
    - If the parent node was a 3-node, repeat.

Example: Build a 2–3 Tree from 9, 5, 8, 3, 2, 4, 7



Exercise: 2–3 Tree Construction

Build the 2–3 tree that results from inserting these keys, in the given order, into an initially empty tree:

C, O, M, P, U, T, I, N, G

Exercise: 2–3 Tree Construction

Extensions

- 2–3–4 trees: includes 4-nodes.
- B-trees: a generalisation. (2–3 trees are B-trees of order 3)
- B^+-trees: internal nodes *only contain keys*, values are all in the leaves (plus a bunch of optimisations)

Key property: balance is achieved by allowing multiple elements per node.

BSTs - Summary

- Dictionaries store *(key, value)* pairs.
- BSTs provide $\Theta(\log n)$ search, insert and delete operations.
- Standard BSTs can degrade to linked lists and $\Theta(n)$ worst case performance.
  - Self-balancing: AVL trees
  - Change of representation: 2–3 trees

BSTs - In practice

- Self-balancing trees tend to be better when the dictionary is fully in memory.
  - C++ *maps*: implemented via Red-black trees.
- Multiple elements per node are a better choice when secondary memory is involved.
  - B-trees (and its invariants) are widely used in SQL databases (PostgreSQL, SQLite) and filesystems (Ext4, Reiser4).

Next week: C++ maps use BSTs. Python dicts use hash tables.