Data Structures and Algorithms

Session 10. February 23, 2009

Instructor: Bert Huang

http://www.cs.columbia.edu/~bert/courses/3137

Announcements

- Homework 2 due now.
- Homework 3 to be posted after class. Due 3/9
- Midterm review March 9th
- Midterm Exam March 11th
  - closed book, closed notes

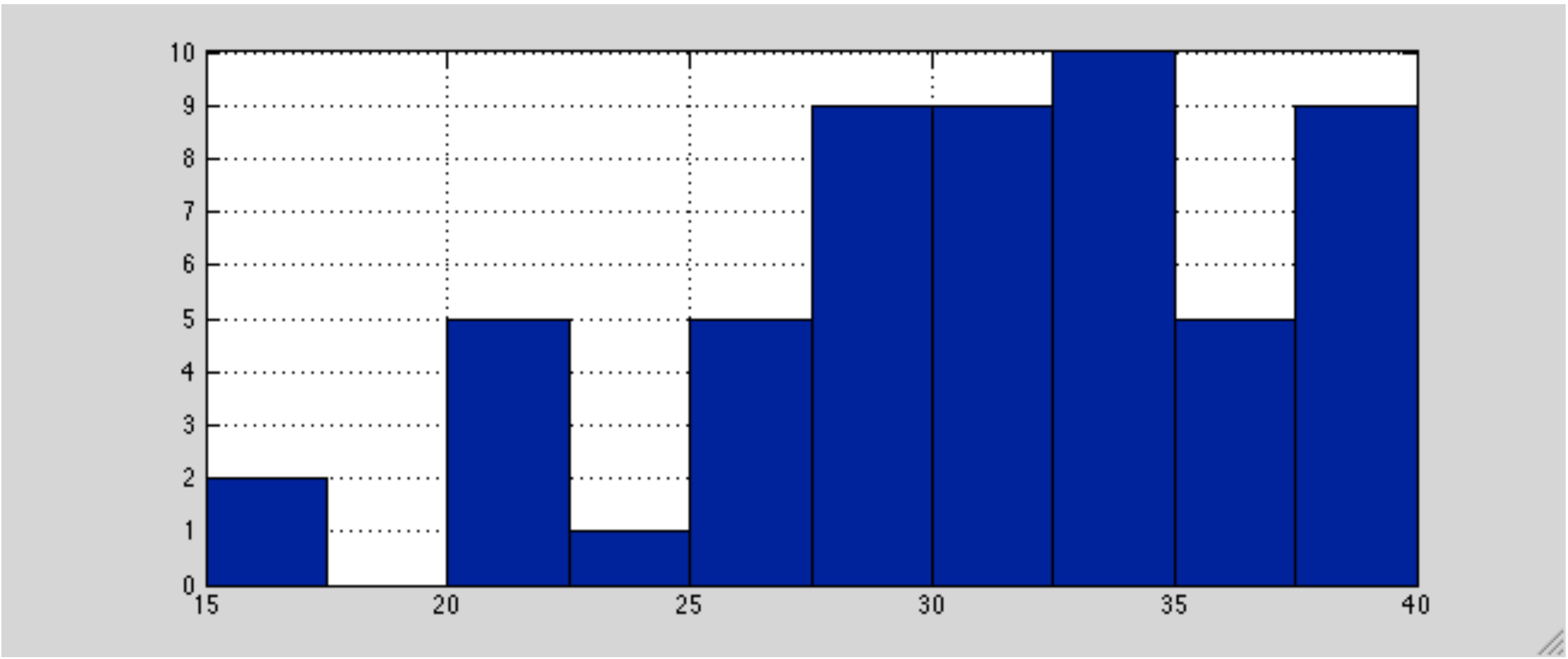
Review

- Brief look at tradeoffs
- Balanced (AVL) Binary Search Trees
    - AVL Tree property
    - Tree Rotations
    - Worst case depth analysis

Today's Plan

- HW1 solutions (long overdue)
- Splay Trees
- Prefix Trees (tries)

HW1 Histogram



- Average was 31.25, std-deviation 6

Amortized Running Time

- Don't guarantee *each* operation is **O(log N)**
- Instead, prove that **M** operations take **O(M log N)**
- Then each operation has an **amortized** running time of **O(log N)**

Splay Trees

- Like AVL trees, use the standard binary search tree property
- After any operation on a node, make that node the new root of the tree
  - Make the node the root by repeating one of two moves that make the tree more spread out

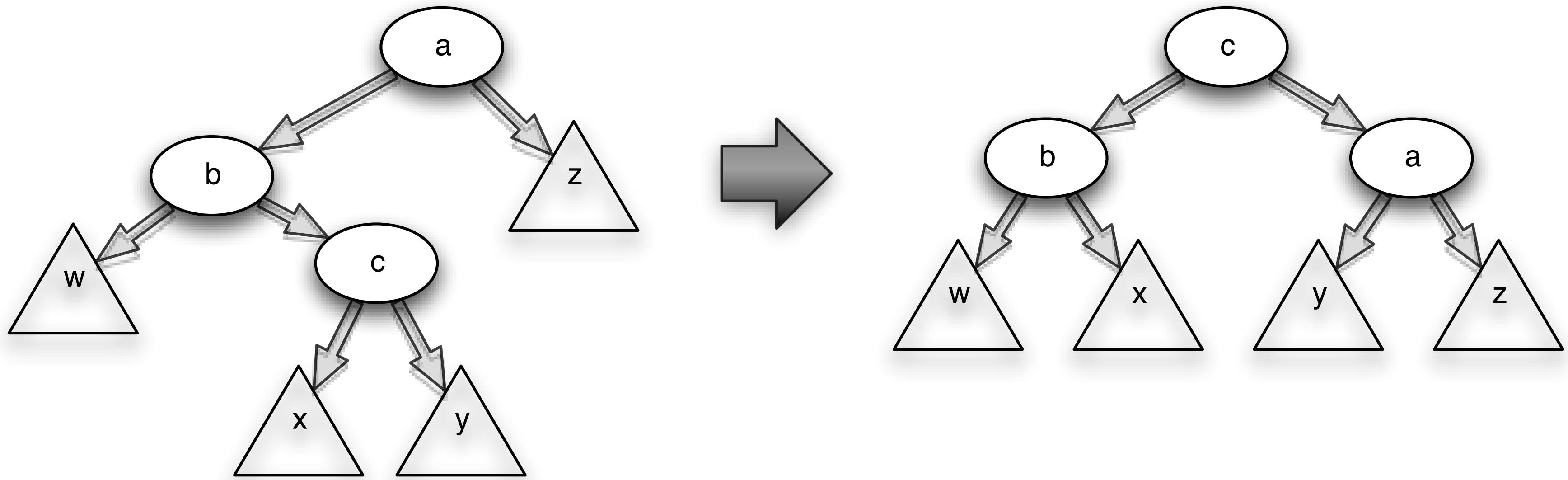
Informal Justification

- Similar to *caching*.
  - Heuristically, data that is accessed tends to be accessed often.
- Easier to implement than AVL trees
  - No height info

Easy cases

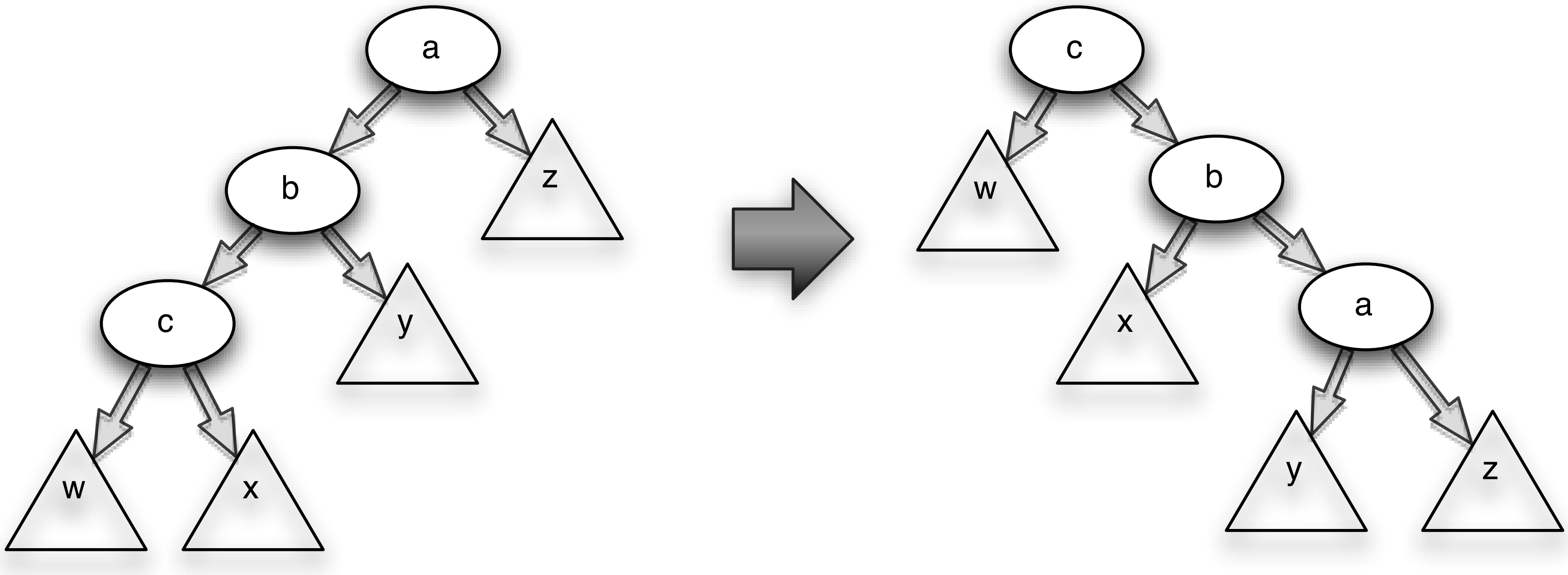
- If node is root, do nothing
- If node is child of root, do single AVL rotation
- Otherwise, node has a grandparent, and there are two cases

Case 1: zig-zag



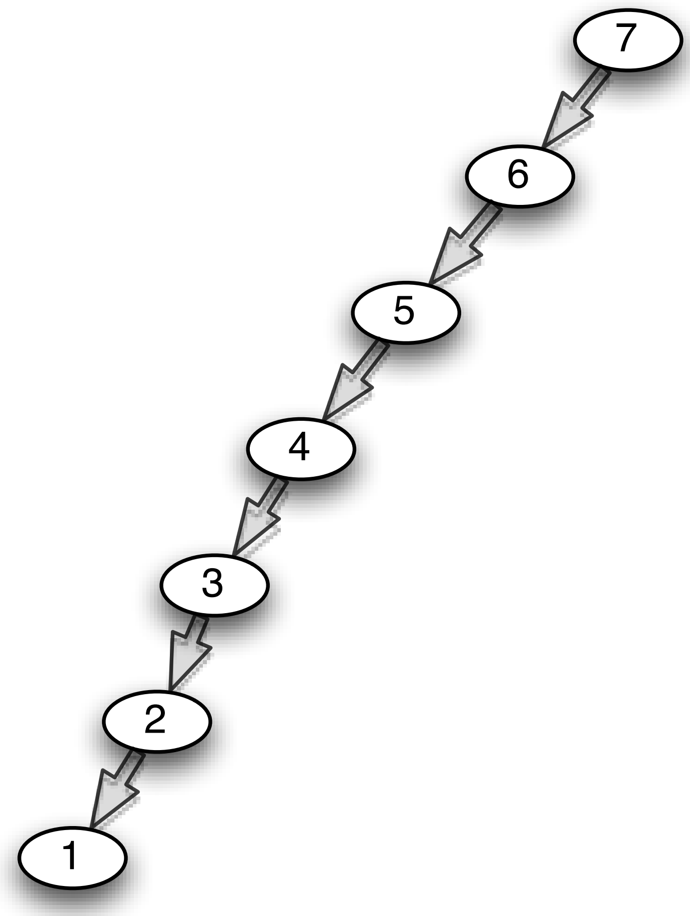
- Use when the node is the right child of a left child (or left-right)
- Double rotate, just like AVL tree

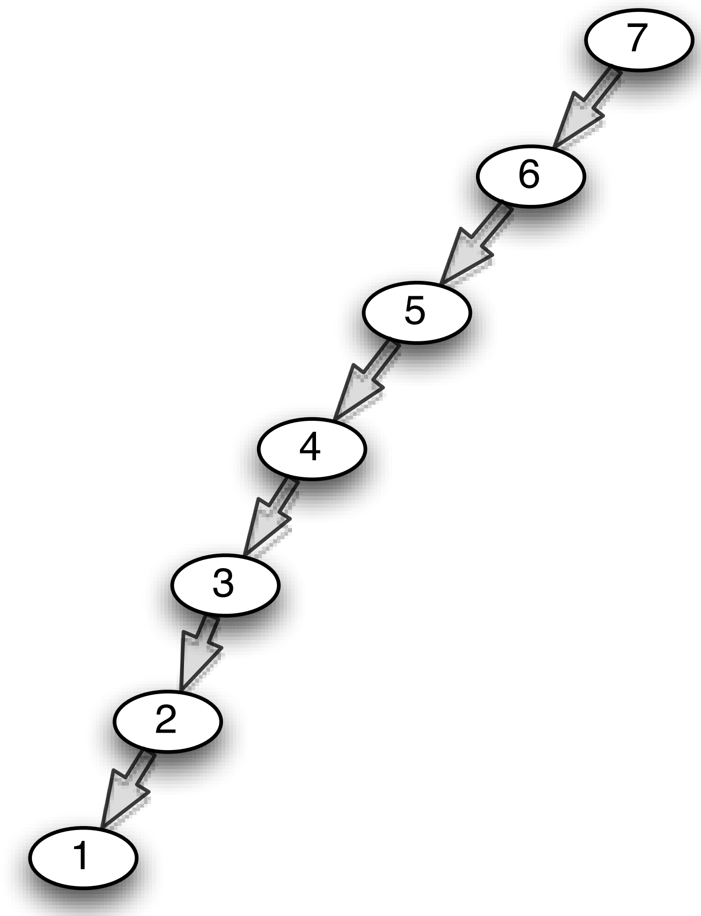
Case 2: zig-zig



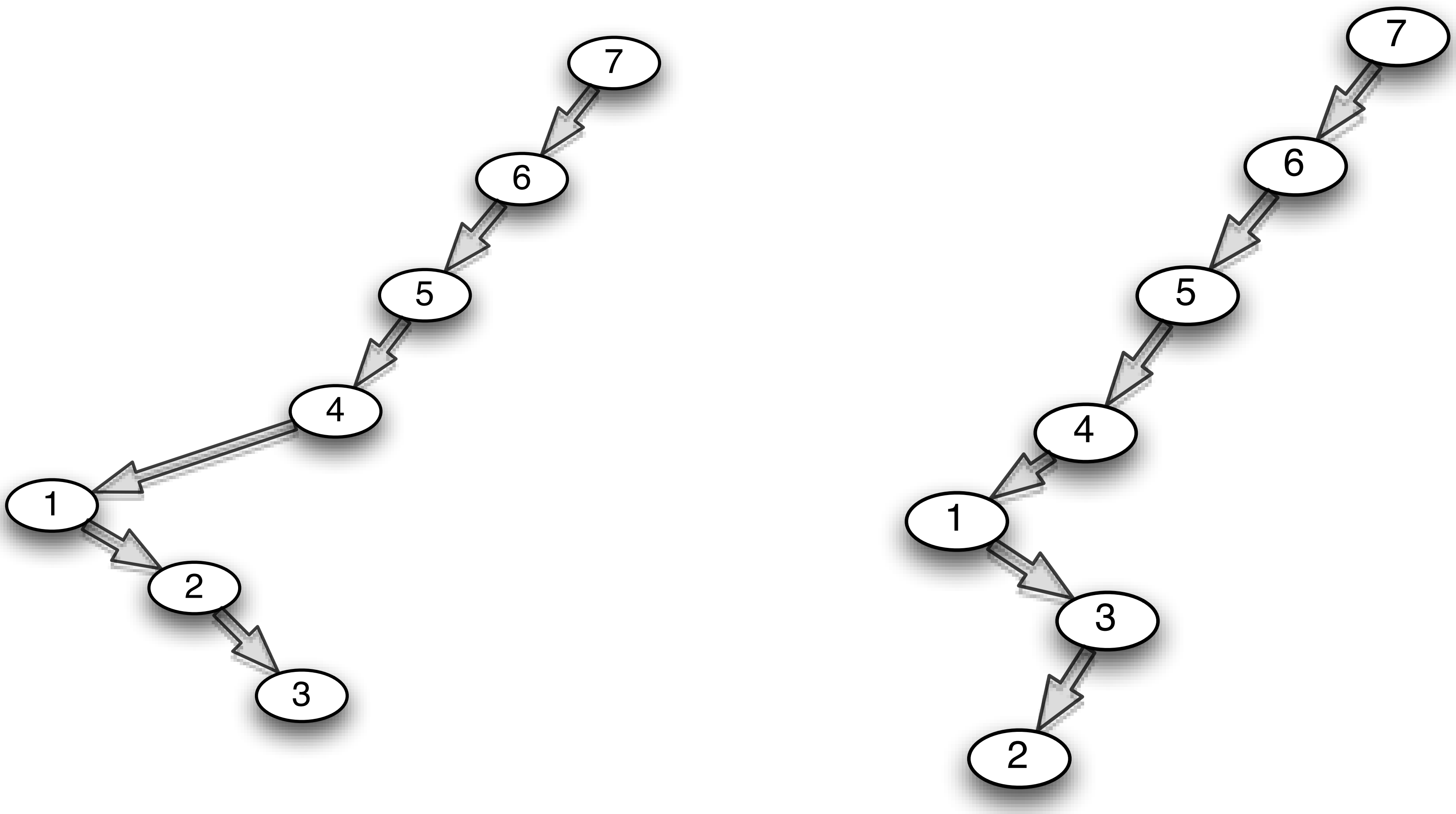
- Use when node is the right-right child (or left-left)
- Reverse the order of grandparent->parent->node
  - Make it node->parent->grandparent

Case 2 versus Single Rotations 1

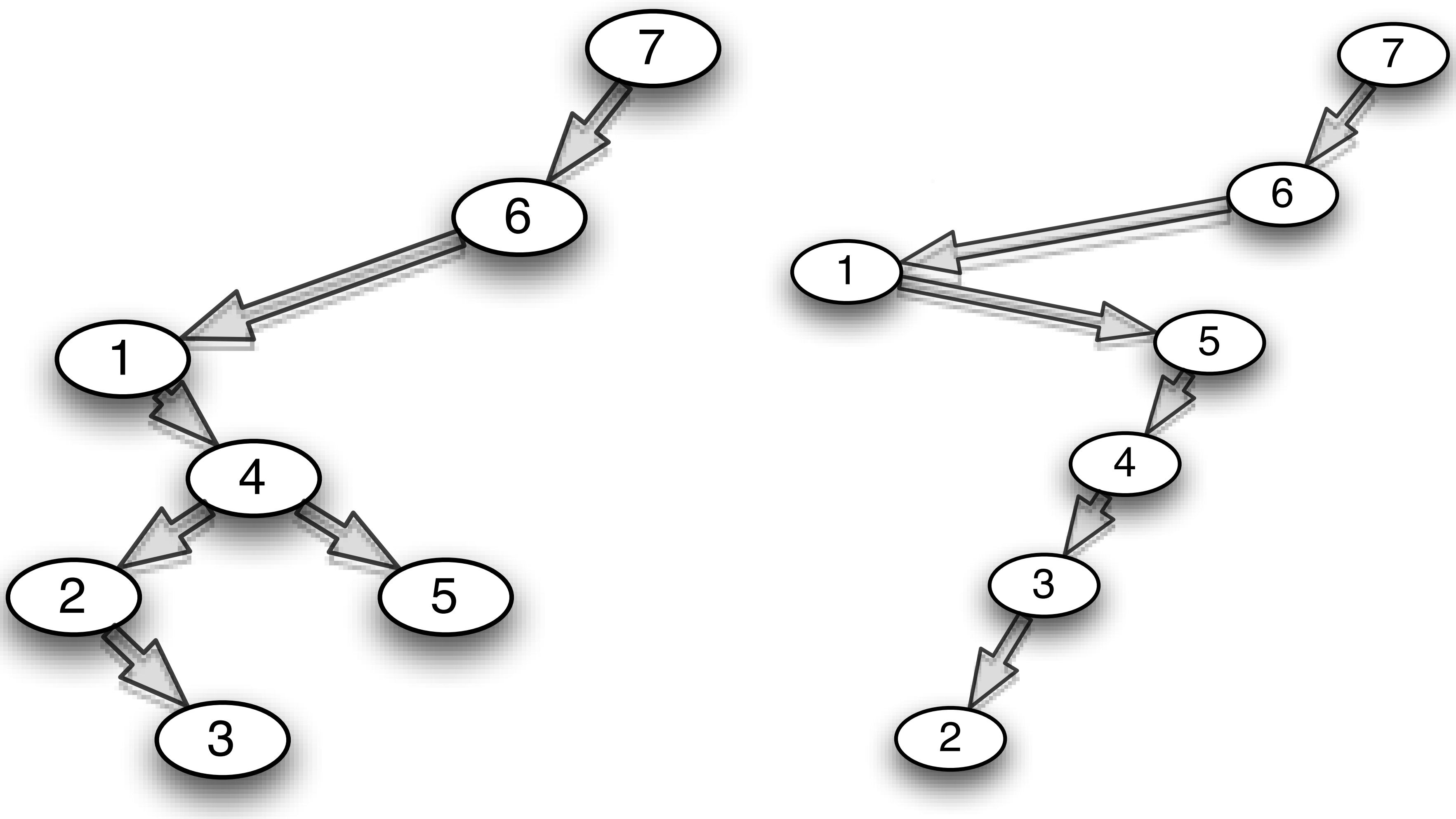




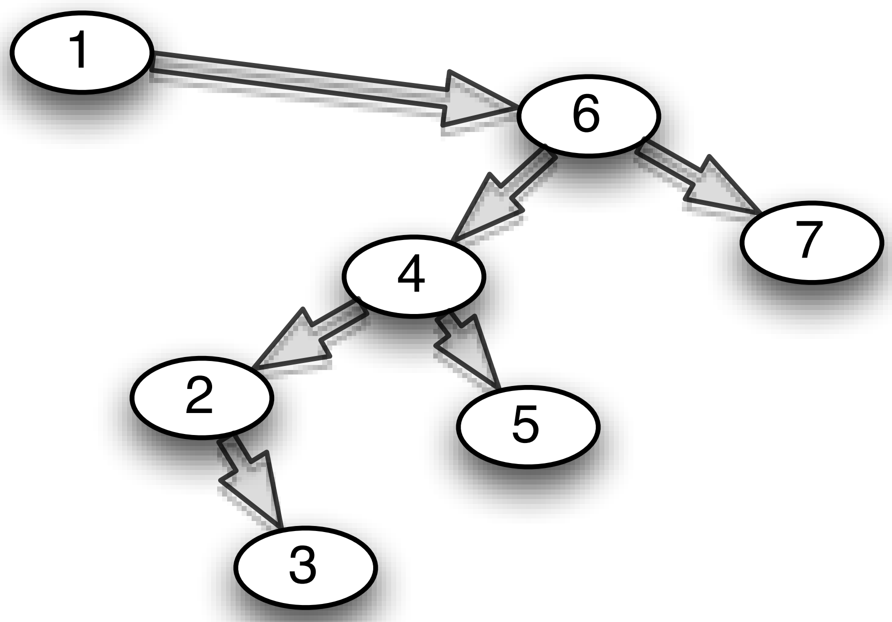
Case 2 versus Single Rotations 2

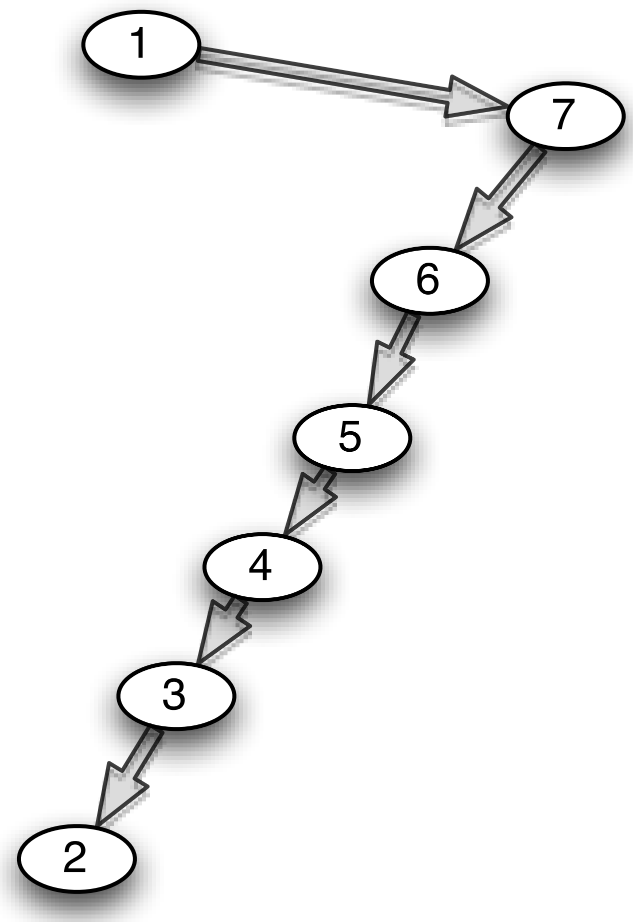


Case 2 versus Single Rotations 3



Case 2 versus Single Rotations 4



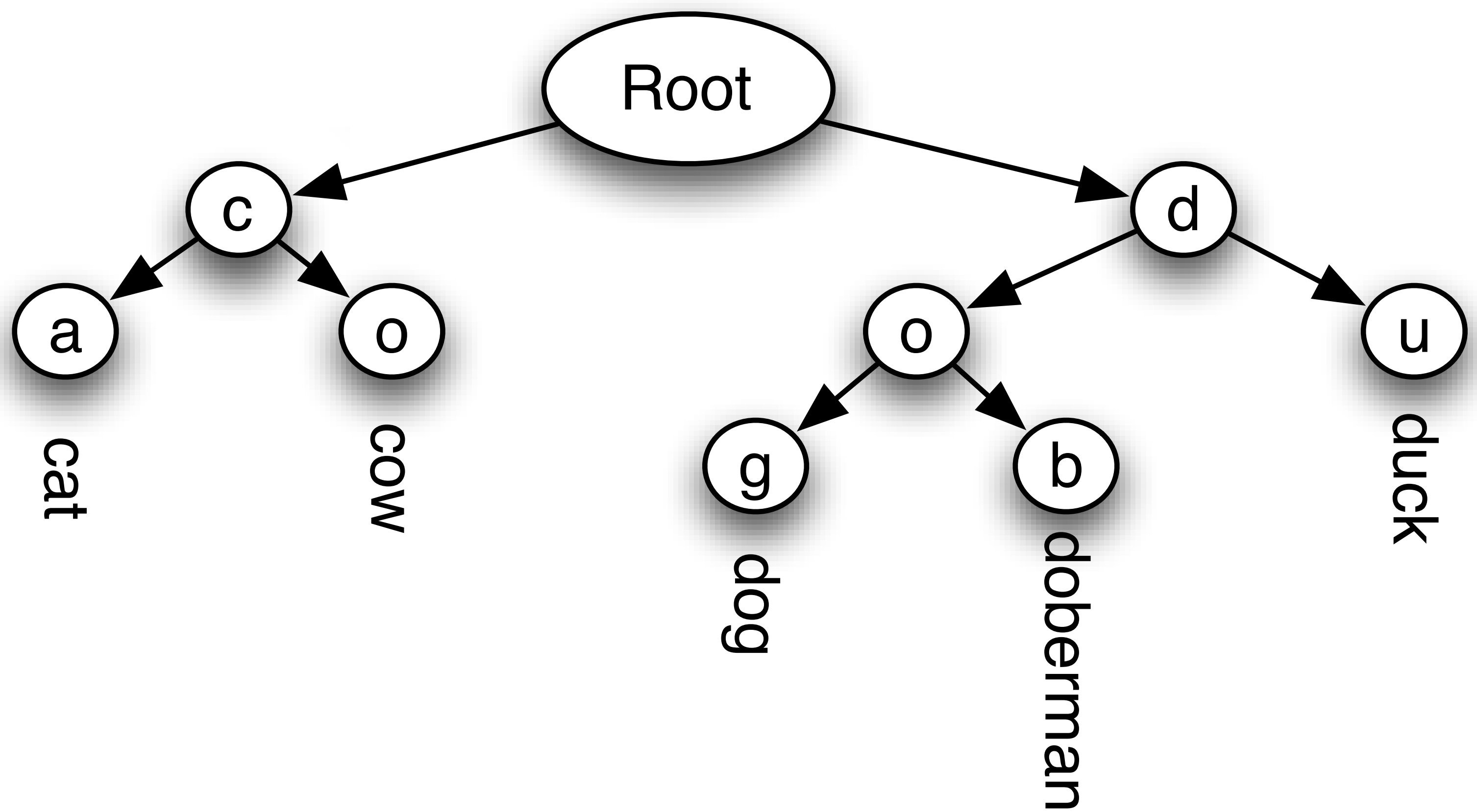


Prefix Trees (Tries)

- Nicknamed “Trie”, short for re**trie**val
- Efficiently store objects for fast retrieval via keys
  - Usually key is a String
- Basic strategy:
  - split into sub-tries based on current letter

Trie Example

- “cat”, “cow”, “dog”, “doberman”, “duck”



Trie Details

- Not all words are at leaves
  - cat, cataclysm, cataclysmic
- Initially, one letter is enough to uniquely identify
- When a new word is inserted that conflicts, need to branch
  - Originally-unique word must be moved to lower level

Trie Analysis

- In the worst case, inserting a key of length **k** or (looking up) is **O(k)**
- This is not dependent on **N**! (surprise, not factorial)
- Much better than **log(N)** for huge data like dictionaries
- Sometimes we can access words even faster.
  - E.g., we can find qwerty uniquely with just “qw”