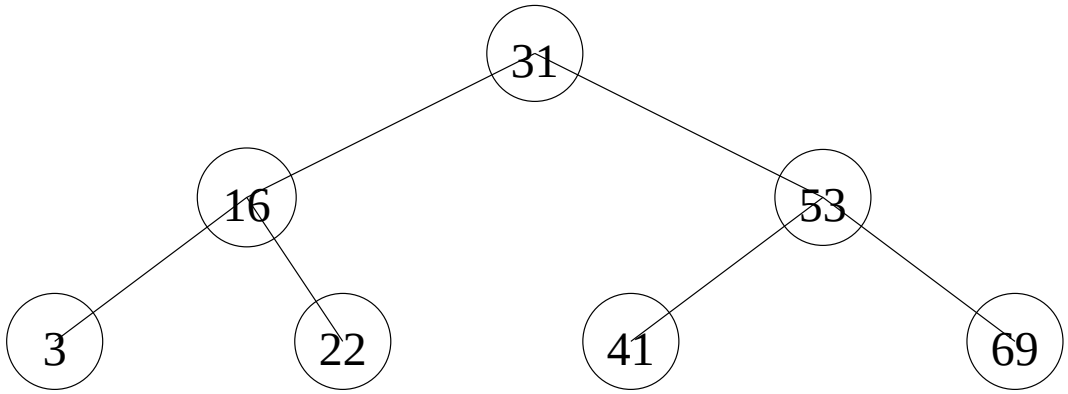
Overview

- Binary Search Trees

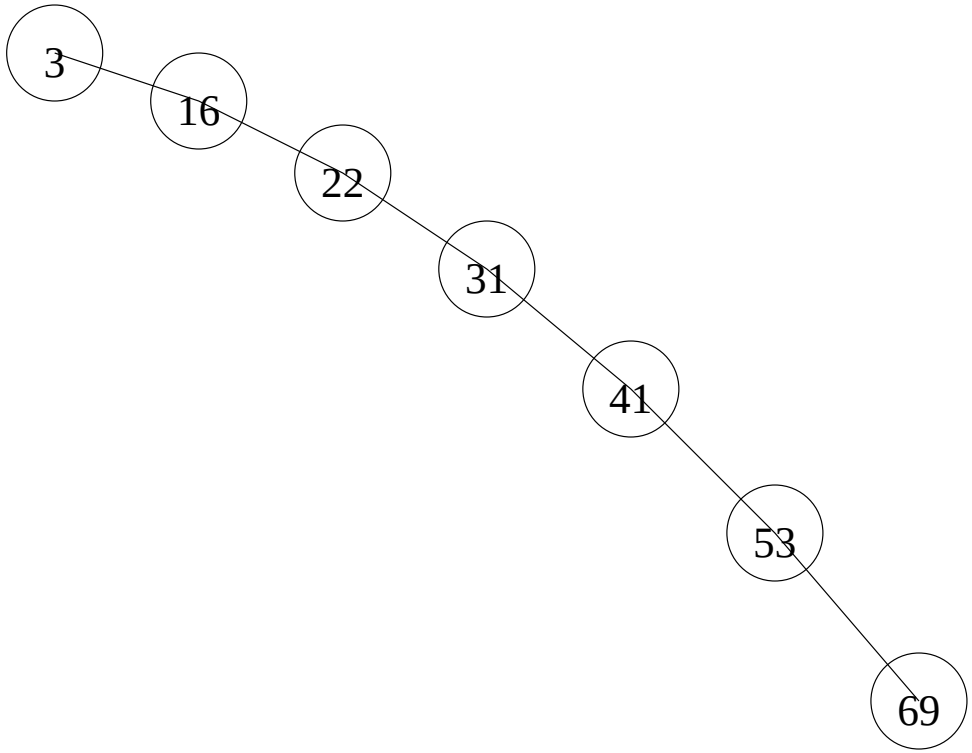
Binary Search Trees

- In a binary search tree, each node's key is greater than its left child's key and less than its right child's key.



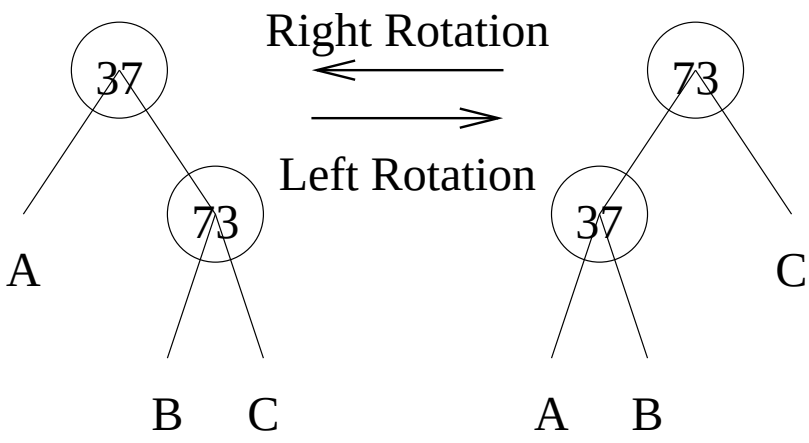
Binary Search Trees (cont.)

- The same keys might be arranged to form a "perfectly" unbalanced tree.



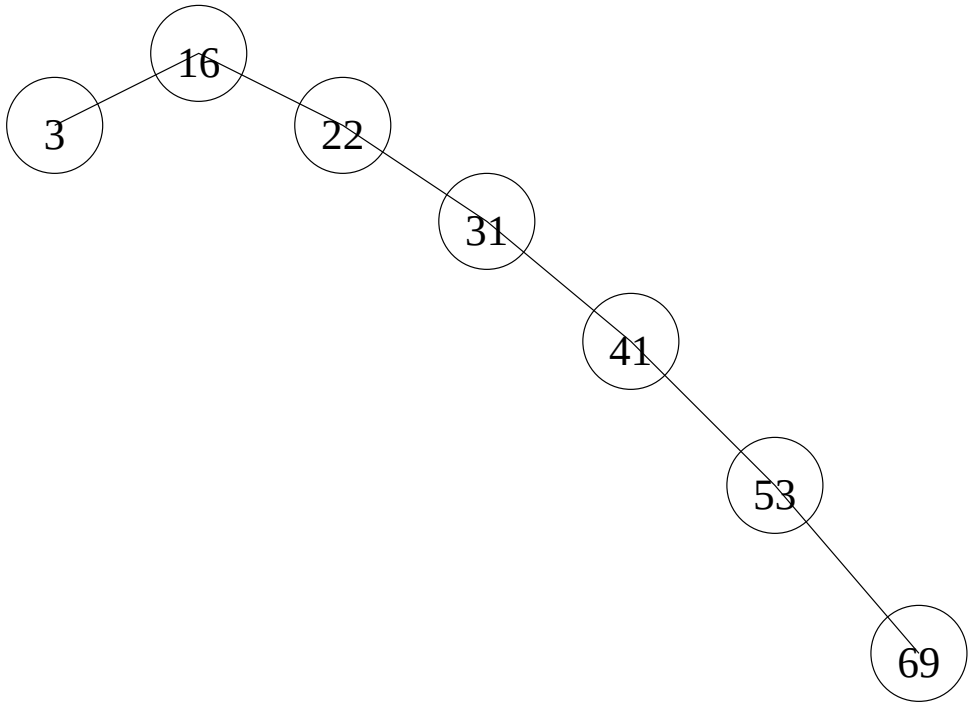
Rotation

- Rotation preserves the binary search tree property.



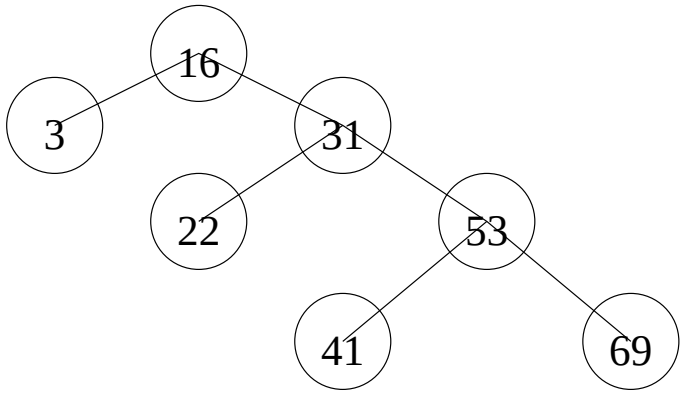
A Single Rotation Applied

- A single left rotation on the original root ("3") of the tree produces:



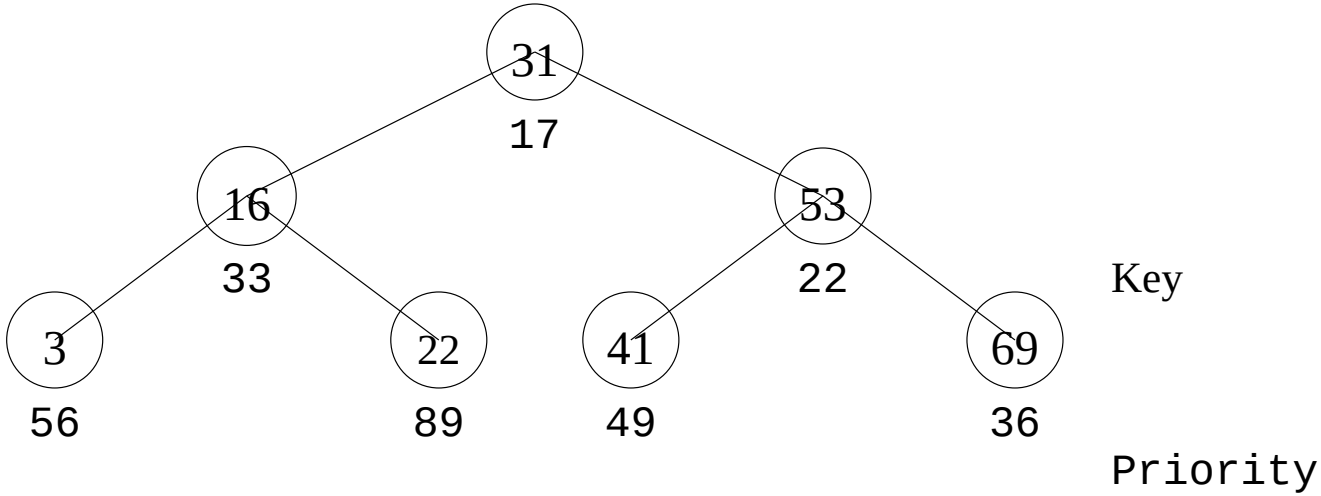
Multiple Rotations Applied

- Performing a left rotation on alternating nodes ("3", "22", and "41") of the original tree produces:



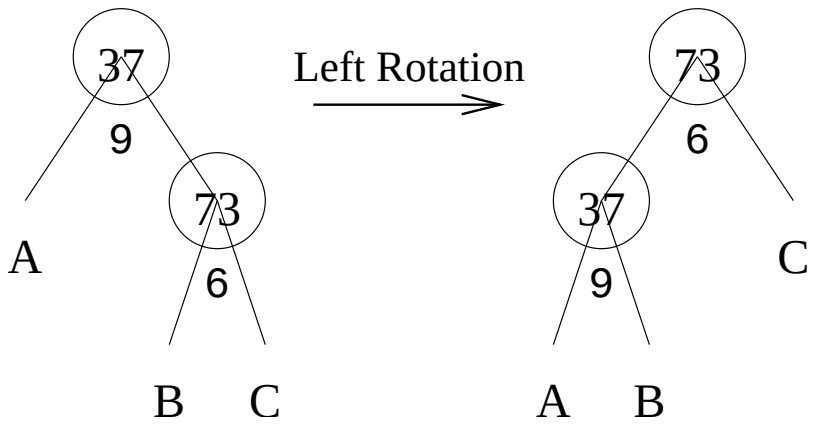
An Idea: The Treap

- Suppose that each node has a distinct *key* and *priority*,
  - and that we maintain the BST property on the key and the heap property on the priority.



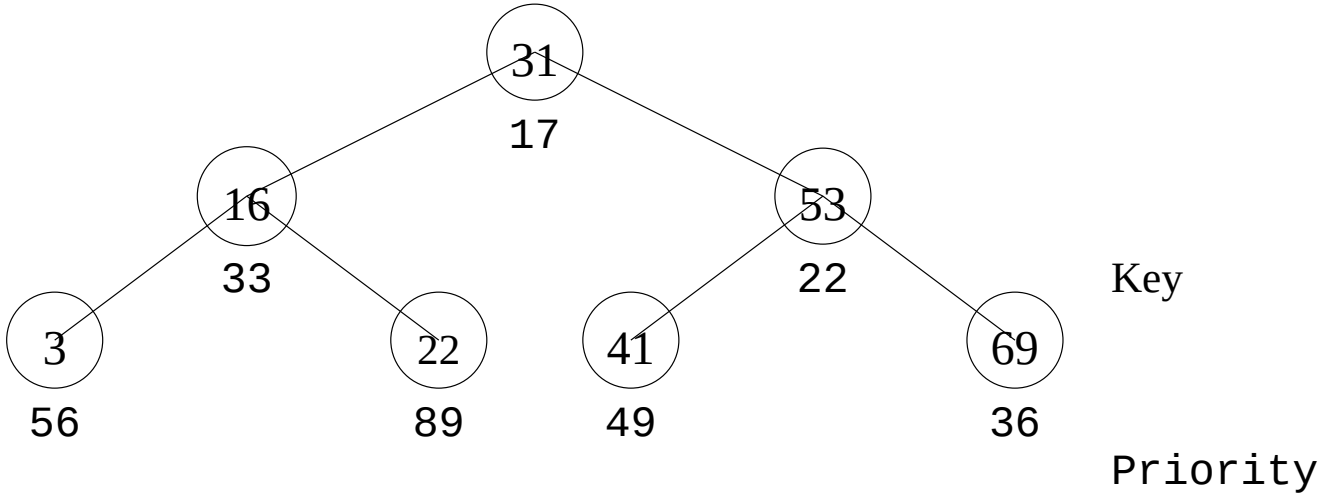
An Idea: The Treap (cont.)

- The insertion procedure is straightforward:
  1. Insert the node (*key*,*priority*,*object*) by *key*, just like a BST.
  2. If the node's *priority* is less than its parent's *priority*, rotate around the parent, lifting the node above its parent.
     - For example, suppose we had inserted (key=73, priority=6) into the following treap.



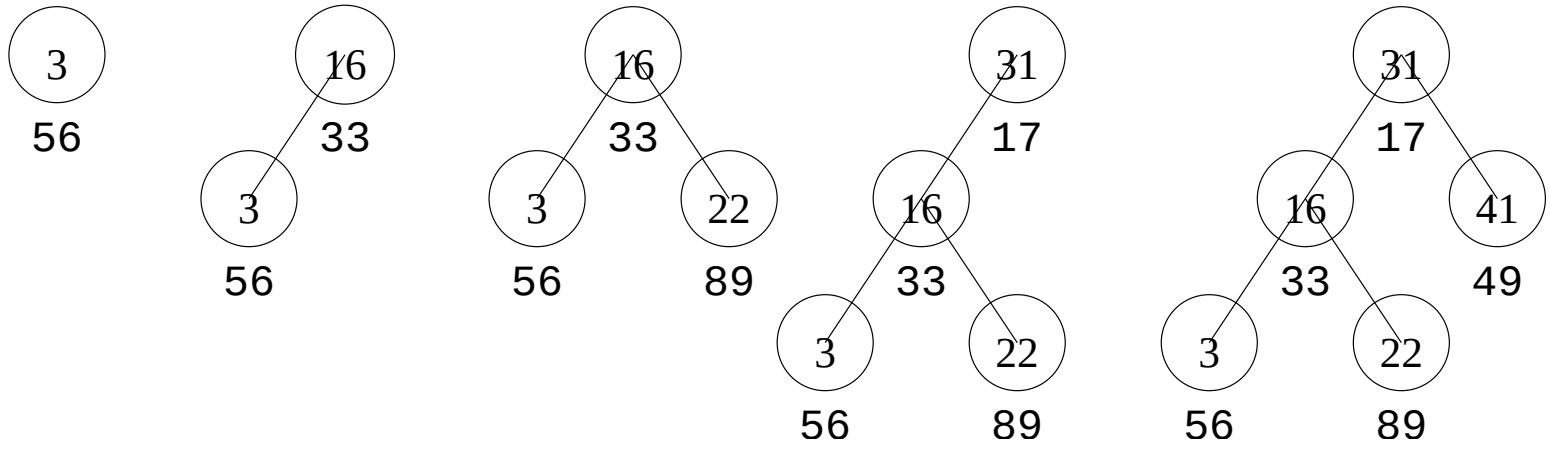
An Idea: The Treap (cont.)

- What happens if we insert (key=49, priority=19) into the following treap?

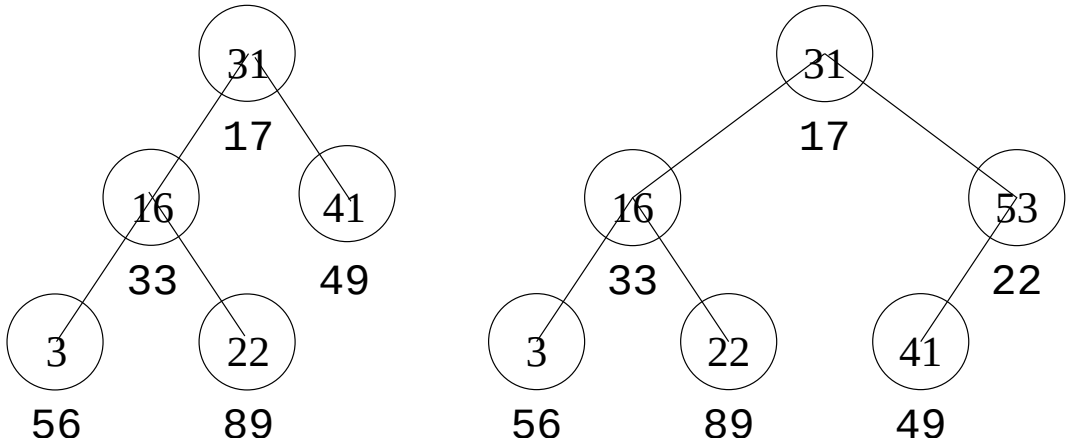


The Punchline

- Suppose that we insert (key=3,priority=56), (16,33), (22,89), (31,17), (41,49), (53,22), and (69,36) in order of increasing *key*.
  - What happens? Note: ignoring the *priority*, this produces a worst-case BST.



The Punchline (cont.)



The Punchline (cont.)

- The punchline: The same treap will result *regardless of the order of insertion*.
  - Suppose that you don't require the *priority* for your application. The priority can be:
    - A randomly generated number
    - A "hash" of the *key*