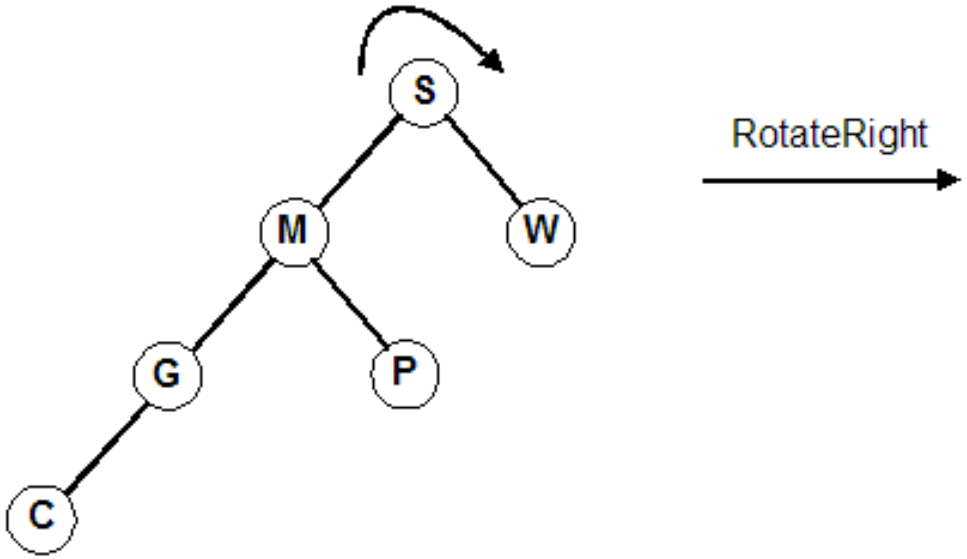
Recap

- Binary search tree (BST)
  - Search
  - Insertion
  - Deletion
  - Rotation
    - Left rotation
    - Right rotation

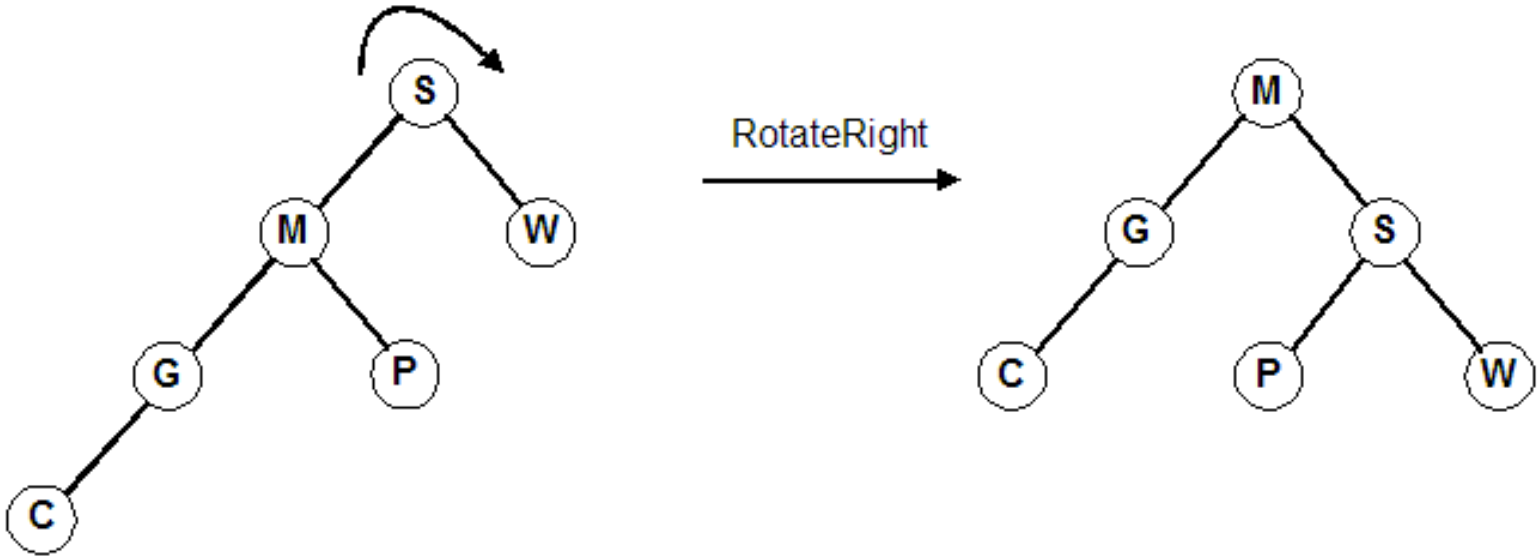
Right Rotation

- Rotate right around the root, S (Same as promoting M)



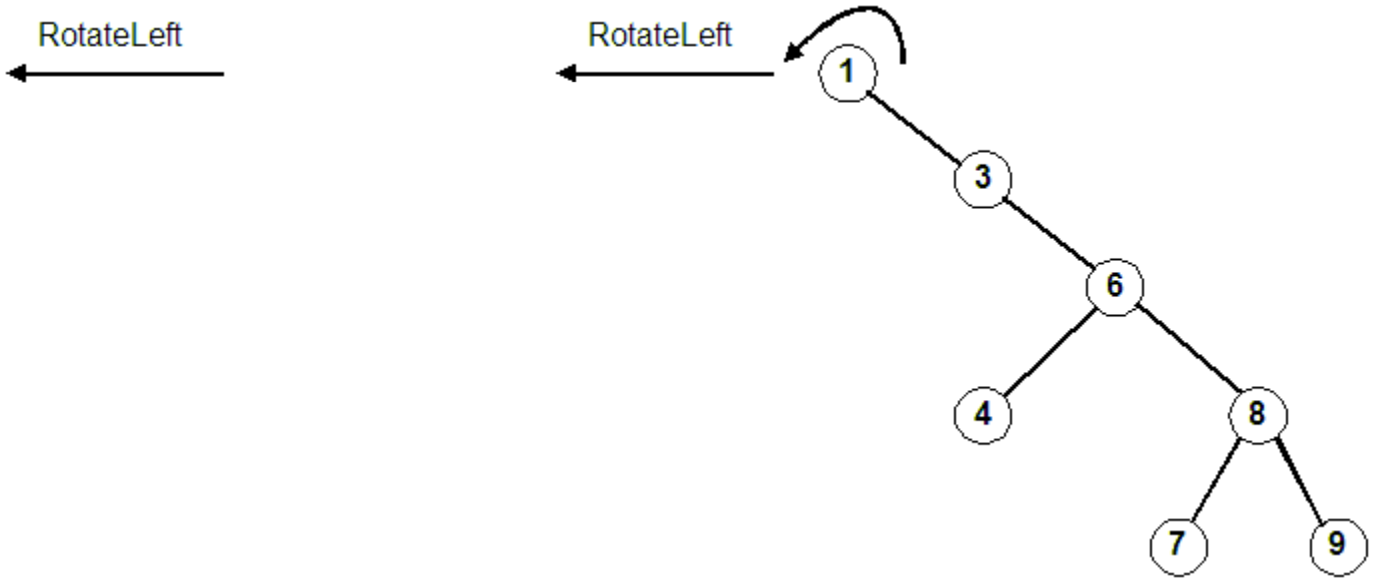
Right Rotation

- Rotate right around the root, S (Same as promoting M)



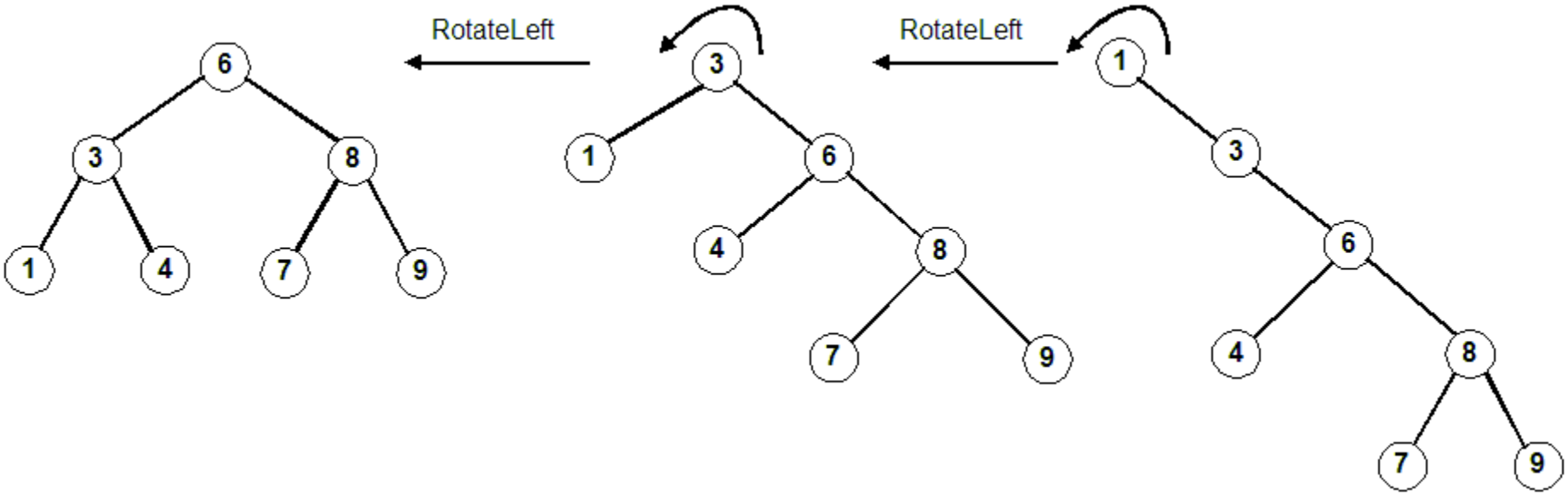
Left Rotation

- Rotate left twice around the root. First around 1, then around 3. (Same as promoting 3 then 6)



Left Rotation

- Rotate left twice around the root. First around 1, then around 3. (Same as promoting 3 then 6)

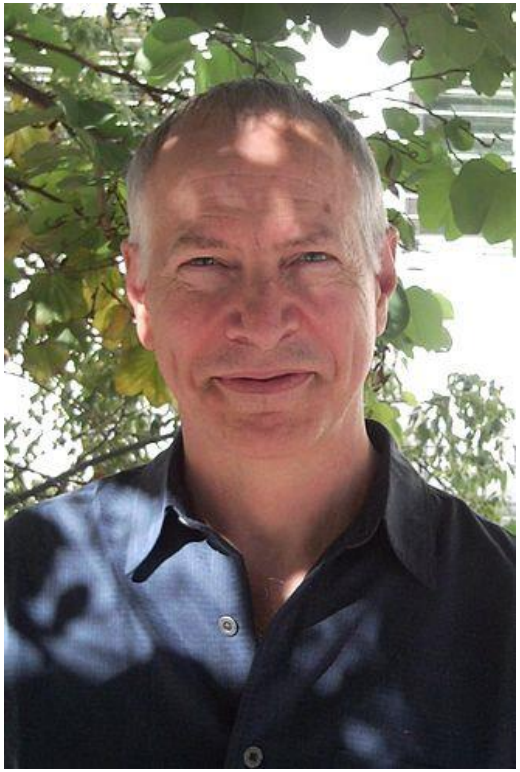


Recall: Rotation of Nodes

- The primary operation used in balancing BSTs
- Promoting a node is the same as rotating around the node's parent
  - The direction of rotation is implied by the position of the child node with respect to the parent:
    - Left -> Rotate right.
    - Right -> Rotate left.
- Sort order is preserved after rotation.

Splay Tree (1985)

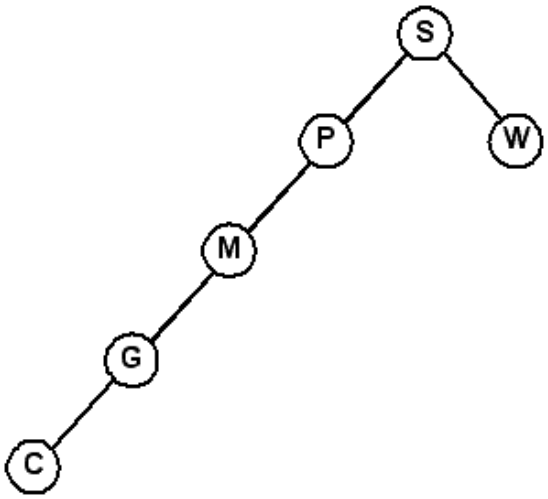
Daniel Dominic Kaplan Sleator
1953-
Professor, Computer Science,
Carnegie Mellon University



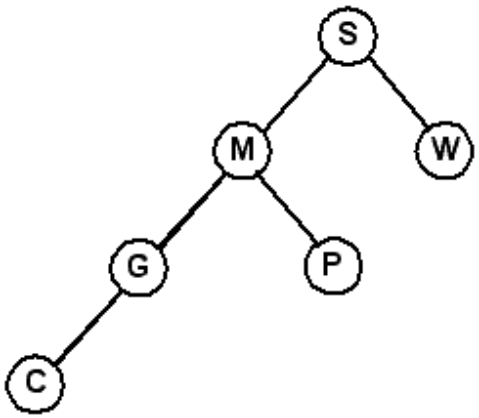
Robert Tarjan
1948-
Distinguished University Professor,
Computer Science, Princeton University
Recipient of Turing Award 1986

Observations

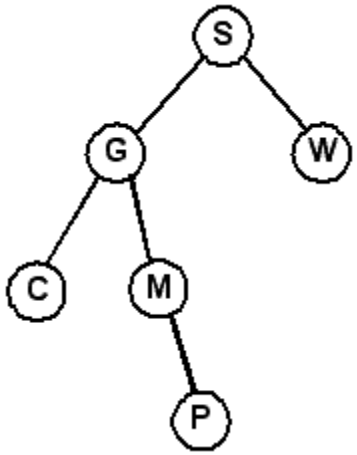
- The trees above all contain the same data
  - Why are there four different representations of the data?



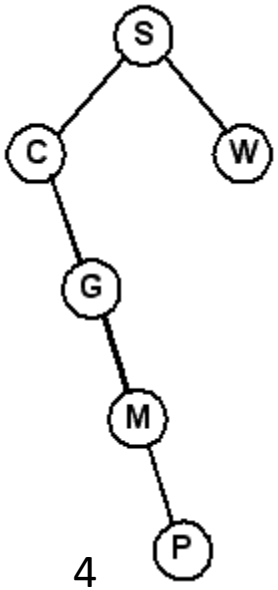
1



2



3



4

Observations

- Most of the time, we don't make any assumptions about the data.
  - Usually assume equal distribution of data and random values.

- Non-random data can lead to worst case situations
  - Think of building a BST from already sorted data

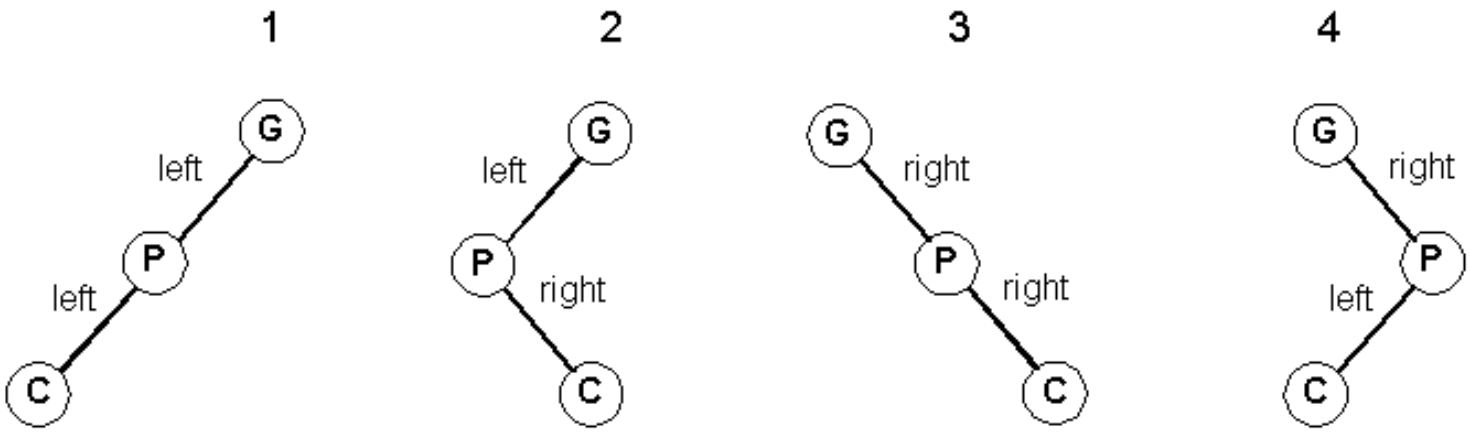
Splay Trees

- A splay tree uses this knowledge to an advantage.
- Newly inserted items are propagated to the root
  - Here “propagated” = “promoted”
- This propagation occurs when writing and reading an item (i.e. insert and access)
  - Nodes just inserted/searched
  - Nodes whose children are deleted
- We call this propagation of a node **splaying**.

Splaying Algorithm

- We want to splay a node two levels at a time.
  - This means we want to promote the node to the position of its grandparent (parent's parent)
- The algorithm depends on the node's orientation to its grandparent
- This leads to 4 possible orientations based on the child, parent and grand-parent nodes.

4 Types of Orientations



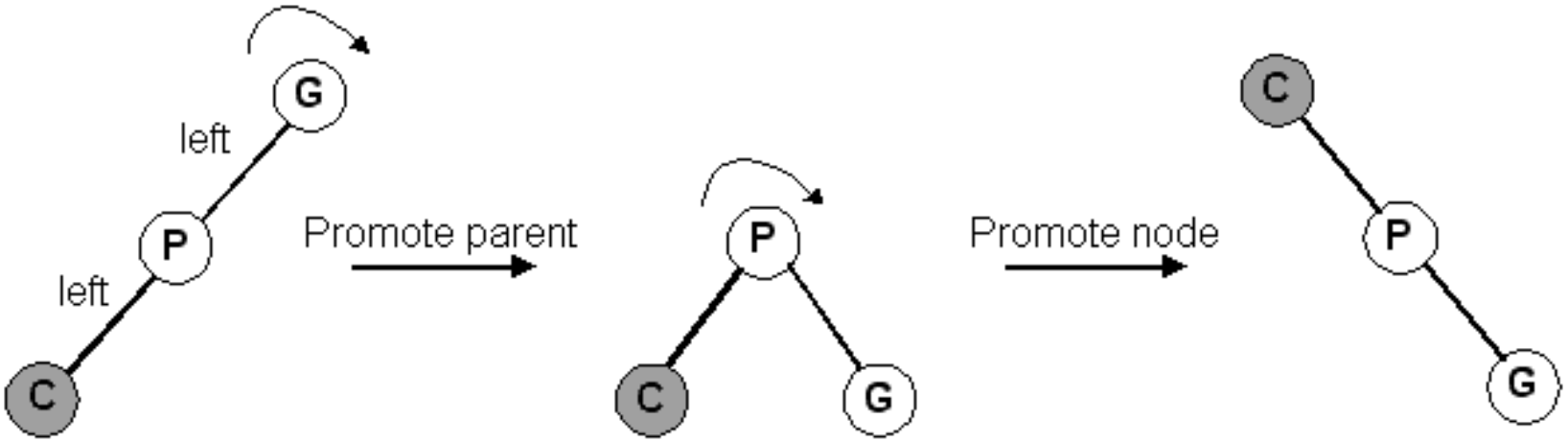
1. left-left orientation: promote the parent, promote the node
2. left-right orientation: promote the node, promote the node (node is promoted twice)
3. right-right orientation: promote the parent, promote the node
4. right-left orientation: promote the node, promote the node (node is promoted twice)

Promoting a Node

- Remember that promoting a node simply means rotating about the node's parent.

- Promoting doesn't require to specify left or right. The direction is implied.
  - If a node is a right-child rotate parent LEFT.
  - If a node is a left-child rotate parent RIGHT.

1. LEFT-LEFT Orientation (ZIG-ZIG)

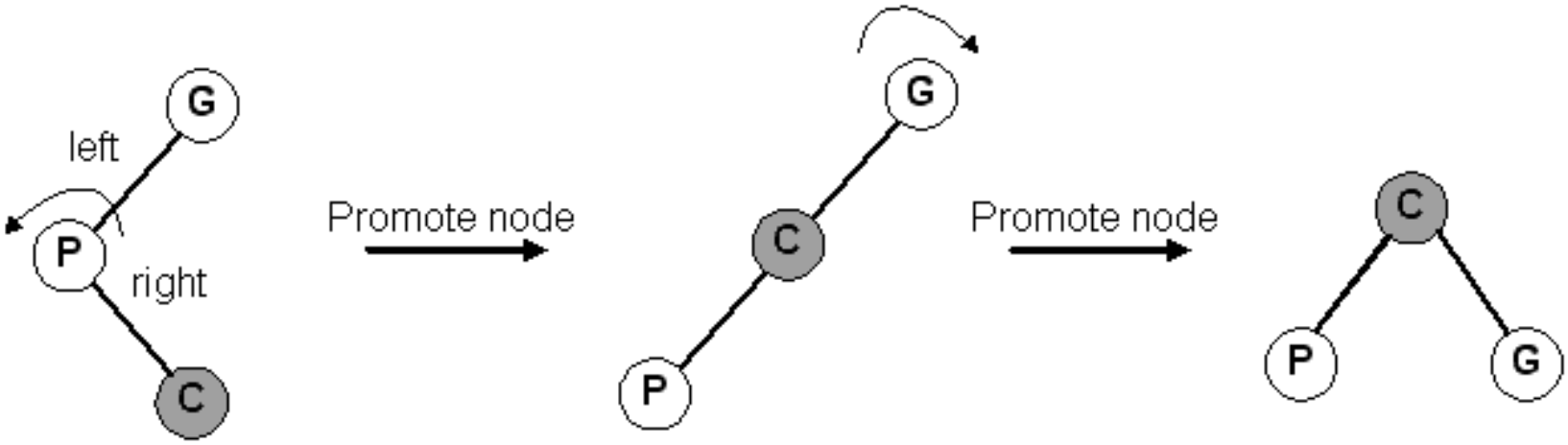


left-left orientation: promote the parent, promote the node

Promoting a node:

- If a node is a right-child rotate parent LEFT.
- If a node is a left-child rotate parent RIGHT.

2. LEFT-RIGHT Orientation (ZIG-ZAG)

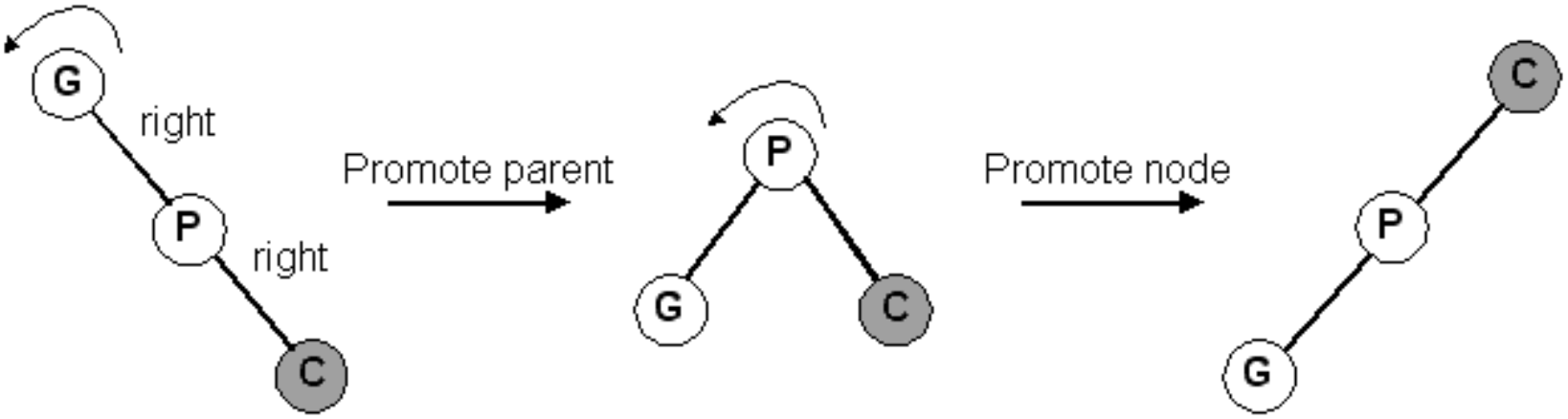


left-right orientation: promote the node, promote the node (node is promoted twice)

Promoting a node:

- If a node is a right-child rotate parent LEFT.
- If a node is a left-child rotate parent RIGHT.

3. RIGHT-RIGHT Orientation (ZIG-ZIG)

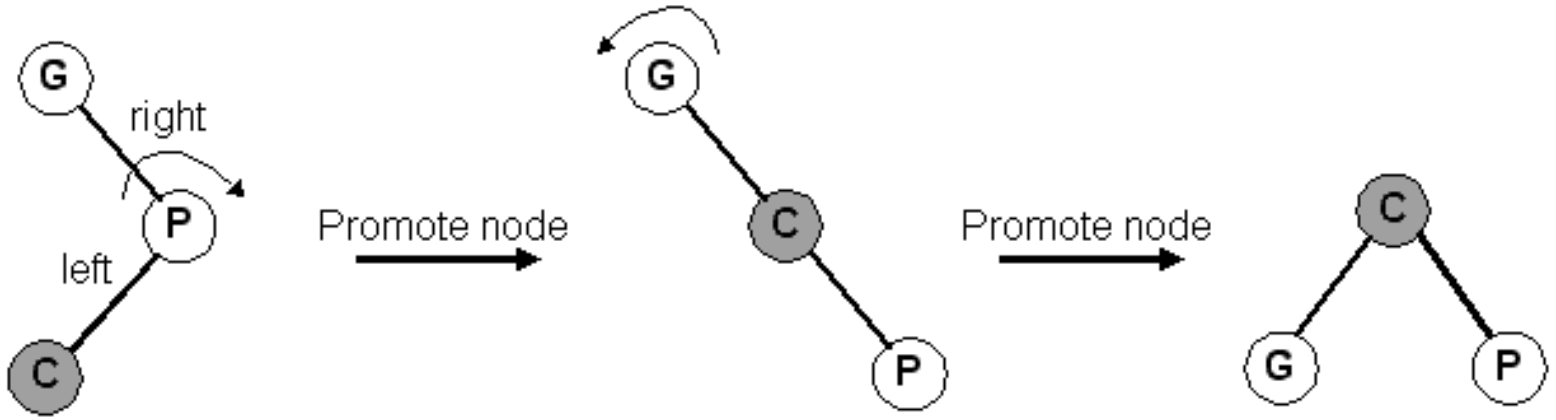


right-right orientation: promote the parent, promote the node

Promoting a node:

- If a node is a right-child rotate parent LEFT.
- If a node is a left-child rotate parent RIGHT.

4. RIGHT-LEFT Orientation (ZIG-ZAG)



Right-left orientation: promote the node, promote the node (node is promoted twice)

Promoting a node:

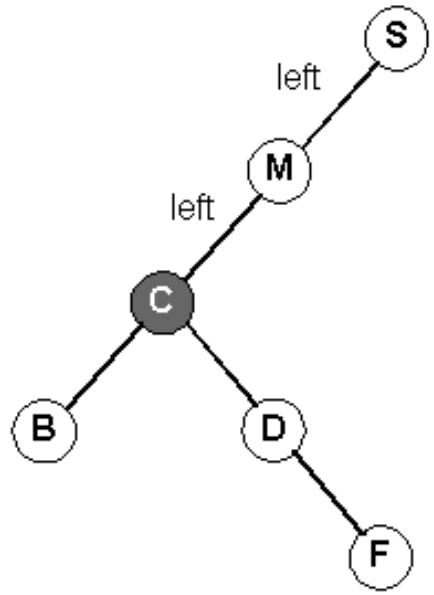
- If a node is a right-child rotate parent LEFT.
- If a node is a left-child rotate parent RIGHT.

Special Case

- We continue to promote until we reach the root.

- The special case is if our parent is the root
  - Simply perform a rotation to bring the node to the root

Splaying Algorithm: Example



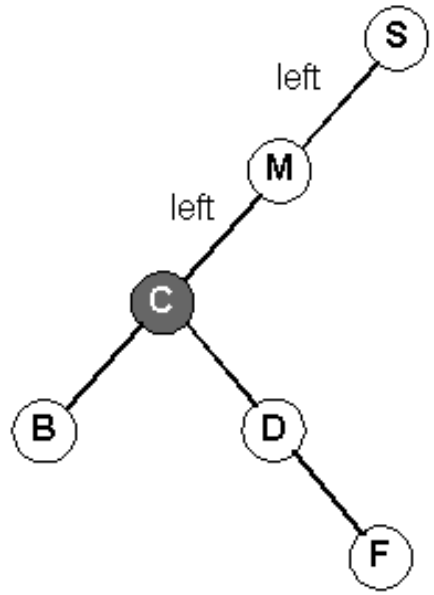
Orientation is left-left

left-left orientation: promote the parent, promote the node

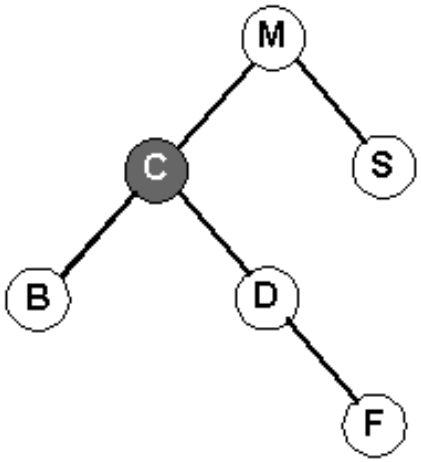
Promoting a node:

- If a node is a right-child rotate parent LEFT.
- If a node is a left-child rotate parent RIGHT.

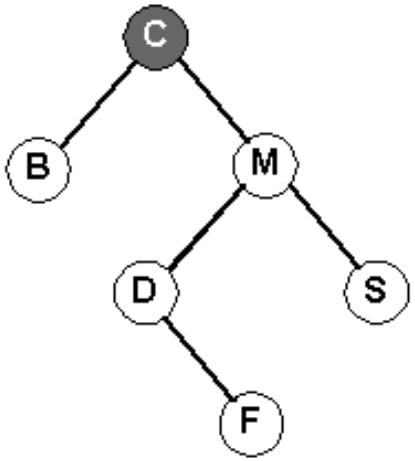
Splaying Algorithm: Example



Orientation is left-left



Promote parent



Promote node

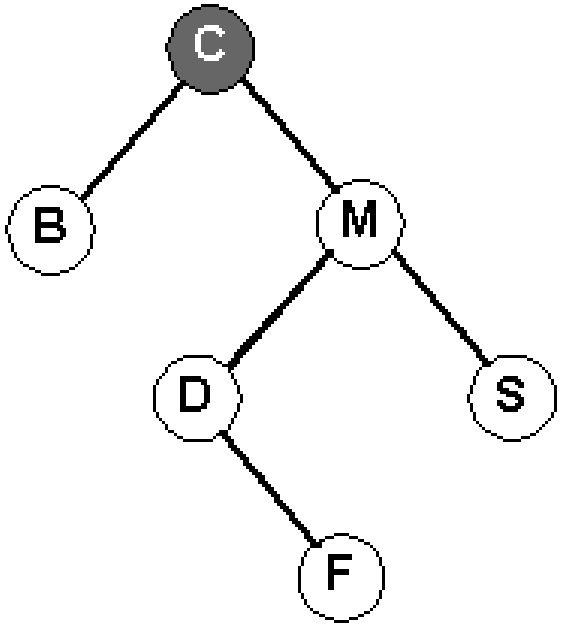
left-left orientation: promote the parent, promote the node

Promoting a node:

- If a node is a right-child rotate parent LEFT.
- If a node is a left-child rotate parent RIGHT.

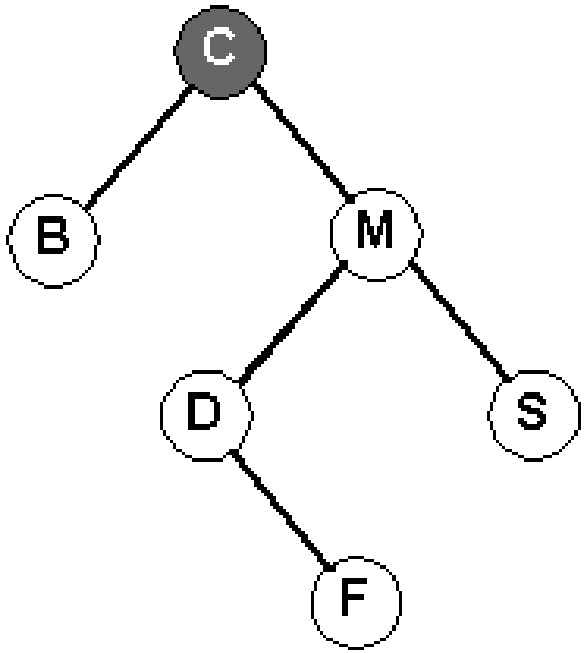
Exercise

- What would be the result of splaying **F** to the root, assuming the current tree is:



Exercise

- What would be the result of splaying **F** to the root, assuming the current tree is:



Steps to follow for splaying node **F:**

1. Consider the parent and grandparent of node F – M,D and F are in the left-right orientation.
2. For left-right orientation, the node F is promoted twice. For promoting a node, we need to rotate left or right depending on whether the node is right or left child.
3. If the node F is child of root node, then just promote the node F.

Summary

- Moves the node being splayed upward and shorten the path to any nodes along the path to the splayed node
- Makes the tree more balanced
- Splay trees are not guaranteed to be balanced.
  - Worst-case is not guaranteed to be good
- Average time to access a node will be better.
- Operations for splaying a node are simple.
  - Variation of the more general BST insertion
- Behaves like a built-in caching mechanism
- Works well with non-uniform access patterns over a small working set.

Animations

- https://www.cs.usfca.edu/~galles/visualization/SplayTree.html
- http://www.ibr.cs.tu-bs.de/courses/ss98/audii/applets/BST/SplayTree-Example.html
- http://techunix.technion.ac.il/~itai/