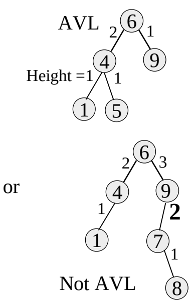
CSE 373 Lecture 8: Trees, Trees, and More Trees

- Today's Topics:
  - AVL Trees
  - Splay Trees
  - B-Trees
- Covered in Chapter 4 of the text

Recall from Last Time: AVL Trees

- AVL trees are height-balanced binary search trees - for every node, heights of left and right subtree differ by no more than 1
- Balance factor of a node = height(left subtree) - height(right subtree)
- An AVL tree has balance factor of 1, 0, or –1 at every node
- Can prove: Height of an AVL tree of N nodes is always O(log N) (see text)



Insertions in AVL Trees

Let the node that needs rebalancing be α.

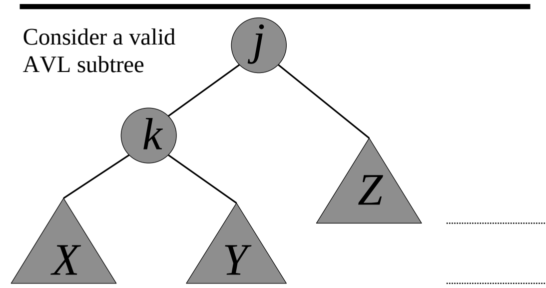
There are 4 cases:
Outside Cases (require single rotation) :
1. Insertion into left subtree of left child of α.
2. Insertion into right subtree of right child of α.

Inside Cases (require double rotation) :
3. Insertion into right subtree of left child of α.
4. Insertion into left subtree of right child of α.

The rebalancing is performed through four separate rotation algorithms.

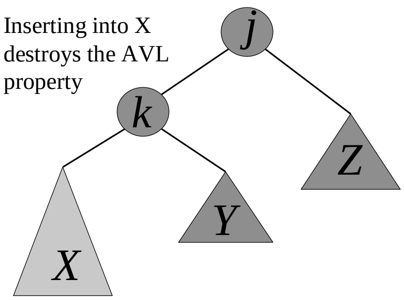
Insertions in AVL Trees: Outside Case

Consider a valid AVL subtree



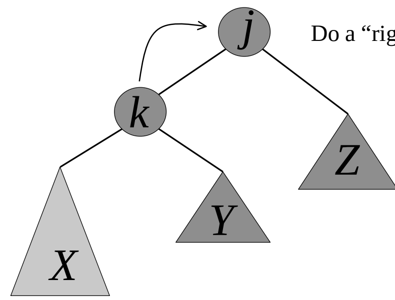
Insertions in AVL Trees: Outside Case

Inserting into X destroys the AVL property



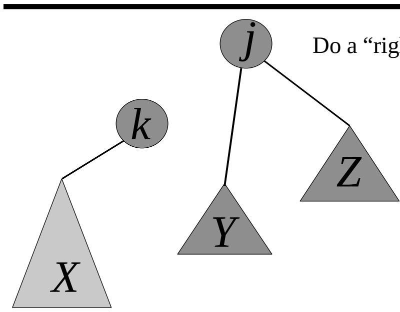
Insertions in AVL Trees: Outside Case

Do a “right rotation”



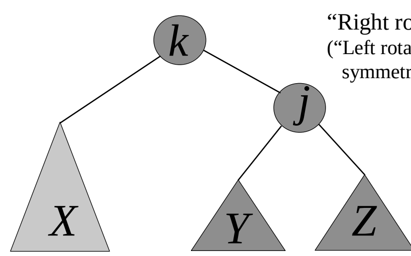
Insertions in AVL Trees: Outside Case

Do a “right rotation”



Insertions in AVL Trees: Outside Case

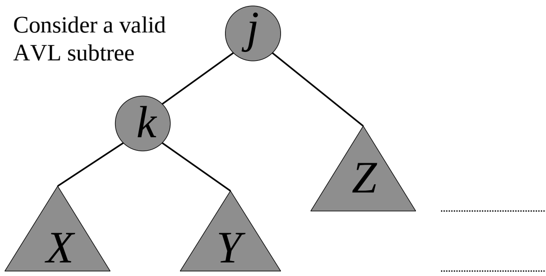
“Right rotation” done!
(“Left rotation” is mirror symmetric)



AVL property has been restored!

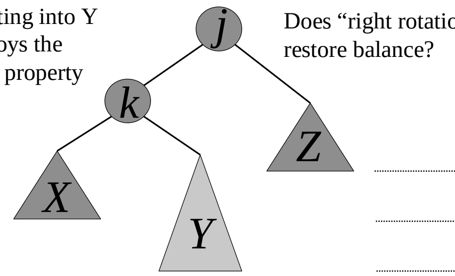
Insertions in AVL Trees: Inside Case

Consider a valid AVL subtree



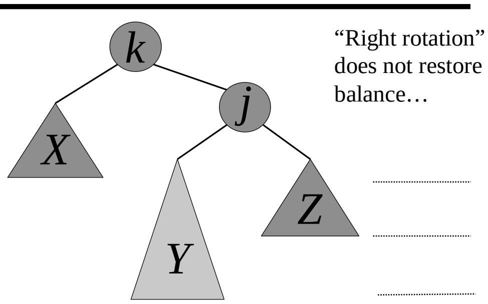
Insertions in AVL Trees: Inside Case

Inserting into Y destroys the AVL property



Does "right rotation" restore balance?

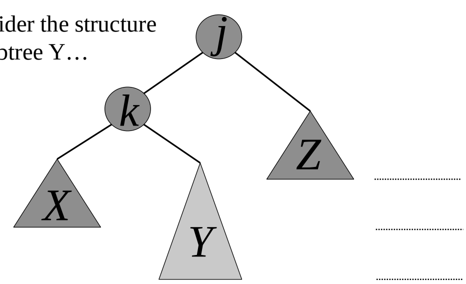
Insertions in AVL Trees: Inside Case



"Right rotation" does not restore balance…

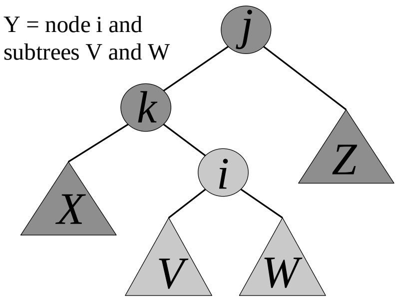
Insertions in AVL Trees: Inside Case

Consider the structure of subtree Y…



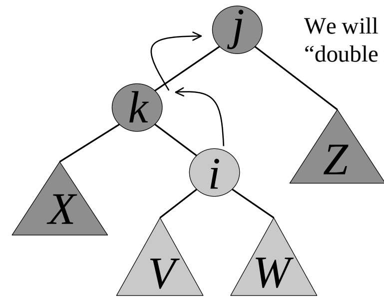
Insertions in AVL Trees: Inside Case

Y = node i and subtrees V and W



Insertions in AVL Trees: Inside Case

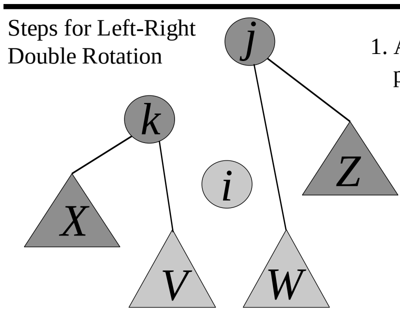
We will do a left-right "double rotation" . . .



Insertions in AVL Trees: Inside Case

Steps for Left-Right Double Rotation

1. Adjust child pointers…



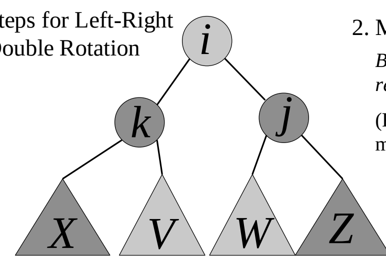
Insertions in AVL Trees: Inside Case

Steps for Left-Right Double Rotation

2. Make ***i*** the root

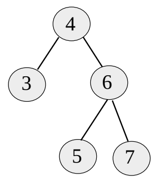
Balance has been restored!

(Right-left case is mirror-symmetric)



AVL Tree Example

✦ Exercise: Insert 8, 1, 0 into following AVL tree:



✦ Exercise: Next, insert 2

Pros and Cons of AVL Trees

Arguments for AVL trees:

1. Search is O(log N) since AVL trees are always balanced.
2. The height balancing adds no more than a constant factor to the speed of insertion.

Arguments against using AVL trees:

1. Difficult to program & debug; more space for height info.
2. Asymptotically faster but can be slow in practice.
3. Most large searches are done in database systems on disk and use other structures (e.g. B-trees).
4. May be OK to have O(N) for a single operation if total run time for many consecutive operations is fast (e.g. Splay trees).

Splay Trees

Splay trees are tree structures that:

1. Are not perfectly balanced all the time
2. Allow actual Find operations to balance the tree so that future operations may run faster

Based on the heuristic:
If X is accessed once, it is likely to be accessed again.

- After node X is accessed, perform “splaying” operations to bring it up to the root of the tree.

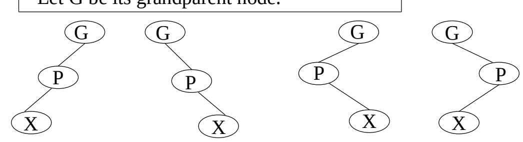
- Do this in a way that leaves the tree more balanced as a whole.

Splay Tree Terminology

• Let X be a non-root node with ≥ 2 ancestors.

• Let P be its parent node.

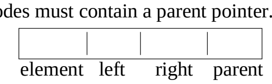
• Let G be its grandparent node.



Will X always have a P and a G?

Splay Tree Operations

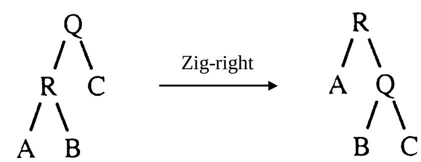
1. Nodes must contain a parent pointer.



2. When X is accessed, apply one of six rotation routines.
- Single Rotations (X has a P but no G)
  - zig_left, zig_right
- Double Rotations (X has both a P and a G)
  - zig_zig_left, zig_zig_right
  - zig_zag_left, zig_zag_right

Splay Trees: Zig operation

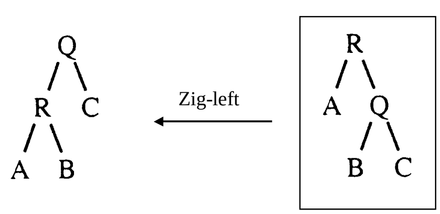
- ✦ "Zig" is just a single rotation, as in an AVL tree
- ✦ Suppose R was the node that was accessed (e.g. using Find)



- ✦ Zig-right moves R to the top → can access R faster next time

Splay Trees: Zig operation

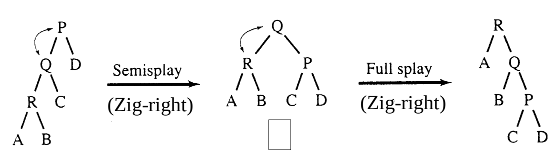
- ✦ Suppose Q is now accessed (e.g. using Find)



- ✦ Zig-left moves Q to the top

Splay Trees: Zig-Zig operation

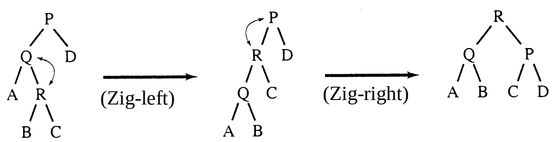
- ✦ "Zig-Zig" consists of two single rotations of the same type (assume R is the node that was accessed):



- ✦ Again, due to "zig-zig" splaying, R has bubbled to the top!

Splay Trees: Zig-Zag operation

- "Zig-Zag" consists of two rotations of the opposite type (assume R is the node that was accessed):

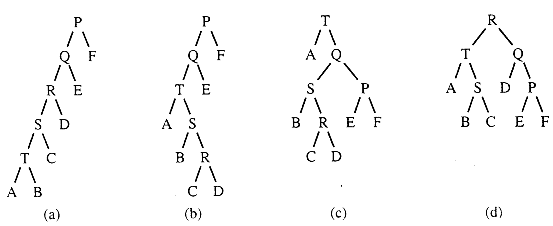


- "Zig-Zag" splaying also causes R to move to the top.

Splay Trees: Example

- Exercise:
- Insert the keys 1, 2, …, 7 into an empty splay tree in decreasing order.
- What happens when you keep accessing "1"?

Splay Trees: Example 2



Restructuring a tree with splaying after accessing *T* (a–c) and then *R* (c–d).

Analysis of Splay Trees

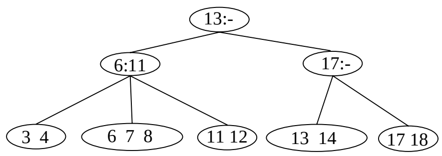
Examples suggest that splaying causes tree to get balanced. The actual analysis is rather advanced and is in Chapter 11.

> Result of Analysis: Any sequence of M operations on a splay tree of size N takes O(M log N) time.
>
> So, the amortized running time for one operation is O(log N).

This guarantees that even if the depths of some nodes get very large, you cannot get a long sequence of O(N) searches because each search operation causes a rebalance.

Beyond Binary Search Trees: Multi-Way Trees

- E.g. B-tree of order 3: Tree has 2 or 3 children per node



- Example: Search for 8

Next Class:

More on B-Trees

Heaps (Priority Queues)

To Do:

Finish Chapter 4 and Start Chapter 6

Homework # 2