CSE 326: Data Structures
Lecture #8
Balancing Act *and* What AVL Stands For

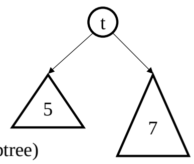
Henry Kautz

Winter Quarter 2002

Beauty is Only Θ(log n) Deep

- Binary Search Trees are fast if they're shallow
  - *e.g.:* complete
- Problems occur when one branch is much longer than the other
  - *How to capture the notion of a "sort of" complete tree?*

Balance

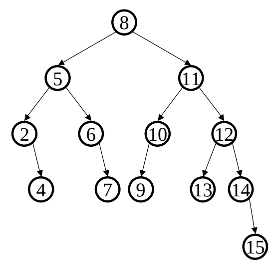


- Balance
  - height(left subtree) - height(right subtree)
  - zero everywhere ⇒ perfectly balanced
  - small everywhere ⇒ balanced enough

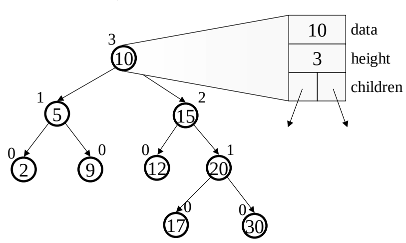
Balance between -1 and 1 everywhere ⇒ maximum height of 1.44 log n

AVL Tree Dictionary Data Structure

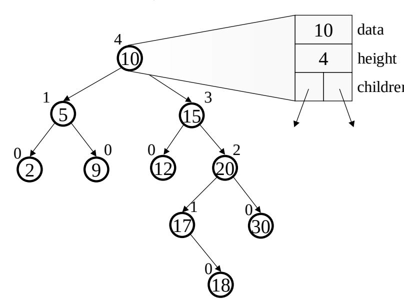
- Binary search tree properties
- Balance property
  - balance = height of left child – height of right child
  - NULL child has height -1
  - balance of every node is: $-1 \le b \le 1$
  - result:
    - depth is $\Theta(\log n)$



An AVL Tree

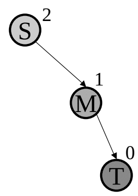


Not An AVL Tree

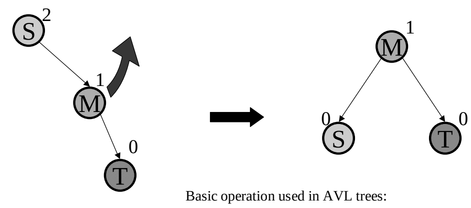


Bad Case #1

Insert(small)
Insert(middle)
Insert(tall)



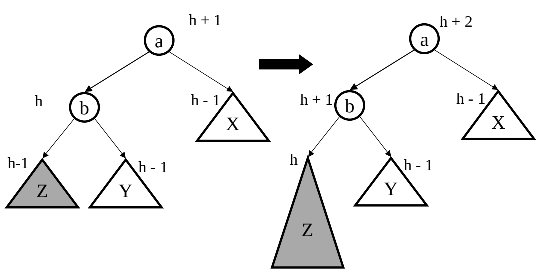
Single Rotation



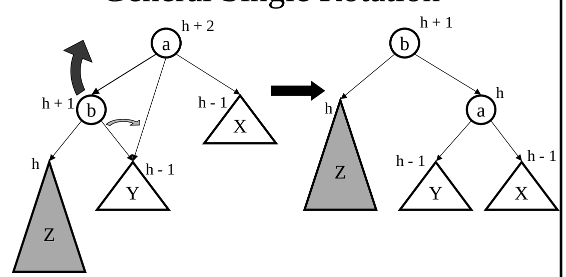
Basic operation used in AVL trees:

A right child could legally have its parent as its left child.

General Case: Insert Unbalances



General Single Rotation

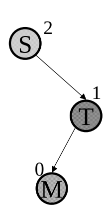


- Height of root same as it was before insert!
- *We can stop here!*

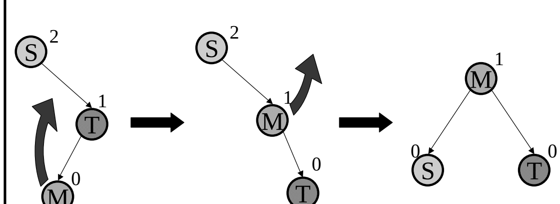
Bad Case #2

Insert(small)
Insert(tall)
Insert(middle)

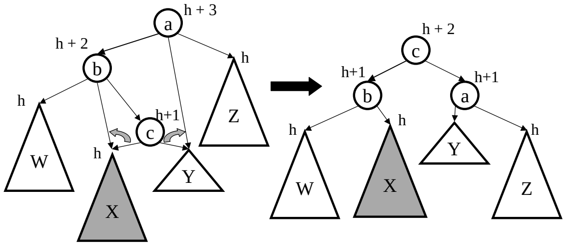
Will a single rotation (bringing T up to the top) fix this?



Double Rotation

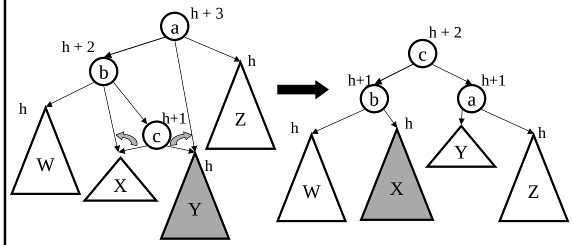


General Double Rotation



- Initially: insert into X unbalances tree (root height goes to h+3)
- "Zig zag" to pull up c – restores root height to h+2, left subtree height to h

Another Double Rotation Case



- Initially: insert into Y unbalances tree (root height goes to h+2)
- "Zig zag" to pull up c – restores root height to h+1, left subtree height to h

Insert Algorithm

- Find spot for value
- Hang new node
- Search back up looking for imbalance
- If there is an imbalance:
  - "outside": Perform single rotation and exit
  - "inside": Perform double rotation and exit

AVL Insert Algorithm

AVL Insert Algorithm

```
void insert(Comparable x,  Node * & root){
  if ( root == NULL )
      root = new Node(x);
  else if (x < root->key){
      insert( x, root->left );
      if (root unbalanced) { rotate... } }
  else
      insert( x, root->right ); }
      if (root unbalanced) { rotate... } }
  root->height = max(root->left->height,
                     root->right->height)+1;
}
```

AVL

- Automatically Virtually Leveled
- Architecture for inVisible Leveling
- Articulating Various Lines
- Amortizing? Very Lousy!
- Amazingly Vexing Letters

Adelson-Velskii Landis

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 usually slower in practice!

Coming Up

- Splay trees
- Get going this weekend on Assignment #3!
- Read section 4.5

To hand in on Monday: One paragraph, in your own words:

1. How (roughly) do Splay Trees work?
2. What are their advantages?
3. What kind of data would give the very best performance for a Splay tree?