Set 11: Ordered Dictionary Abstract Data Types: AVL Trees

CS240: Data Structures and Data Management

Jérémy Barbay

Outline

Binary Search Trees

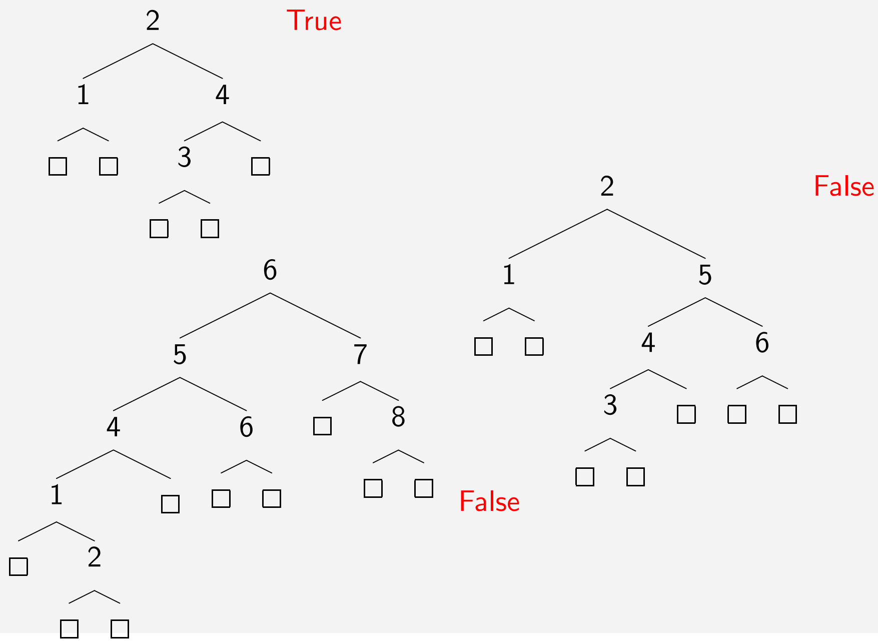
- The worst-case performance is $\Theta(h)$, i.e. $\Theta(n)$
- Randomly built trees perform well
  - Expected height $h = 1.386 \log(n+1)$
- Sequence of n^2 alternating inserts/deletes
  - Expected height $h \in \Theta(\sqrt{n})$
- Possible improvements?
  Keeping a small height will improve the worst case.

Height Balanced Trees

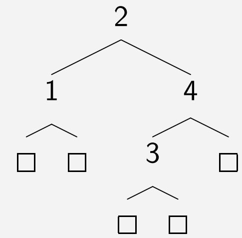
- Can we guarantee tree height?
  - Try to keep our search trees balanced
  - Must not affect the running time
- Balanced Node
  The heights of its subtrees differ by at most one
- AVL Tree
  A Binary Search Tree such that every node is balanced
  - *Adel'son-Vel'skii and Landis, 1962*

Which Trees are AVL?

Which nodes are balanced?



Recording Balance



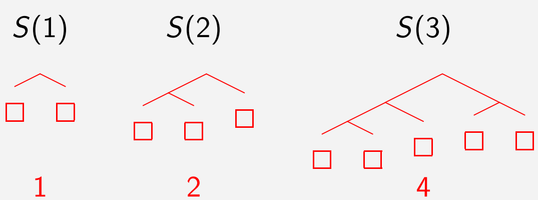
- We can explicitly record a height for each node
  It would take $O(\lg \lg n)$ bits per node.
- Or we can use condition codes
  - = – Balanced
  - > – Left-heavy (by one)
  - < – Right-heavy (by one)

  It would take 2 bits per node.

AVL Tree Height

General Idea

Let $S(h)$ be the fewest possible nodes for an AVL tree of height h (including placeholders)



$$S(h) = 1 + S(h-1) + S(h-2)$$

AVL Tree Height

General Idea

Theorem

h is $\Theta(\log n)$ for an AVL tree of height h and n internal nodes.

Proof:

- Recurrence relation (close to Fibonaci Sequence) gives

$$S(h) = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^{h+2}}{\sqrt{5}} + 1$$

- Note: $S(h) \leq n$.

$$\begin{aligned} h &\leq \frac{\lg n}{\lg \frac{1+\sqrt{5}}{2}} + o(1) \\ &\approx 1.44 \lg n \end{aligned}$$

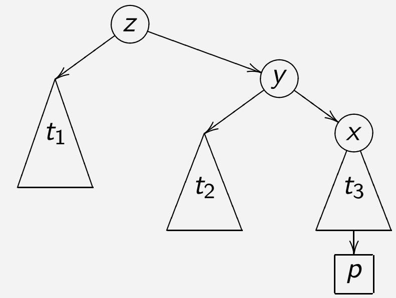
Operations

- `Find`:
  - As in a Binary Search Tree (BST).
- `Insert`
  - Find and insert as in a BST.
  - Update heights (codes) on path back to root
  - Locate a possible unbalanced node, z
  - Perform a rotation (see two next slides)
- `Delete`
  - Find and delete as in a BST
  - Update heights (codes) on path back to root
  - Locate possible unbalanced nodes and rotate them.

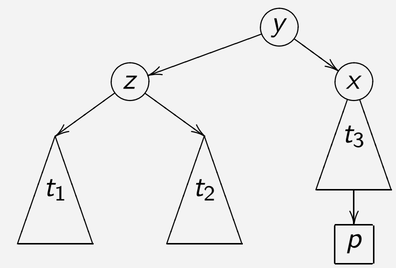
The complexity of `Find` is $O(h)$, i.e. $O(\lg n)$.

"Single" Rotation

- node z fails the AVL test after adding node p:

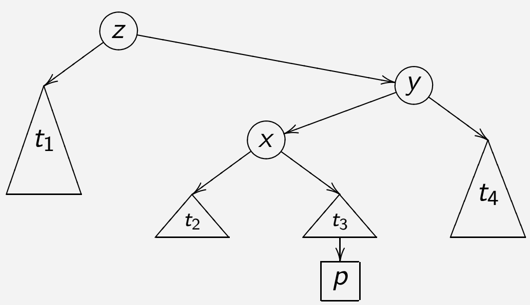


- single rotation regains balance:

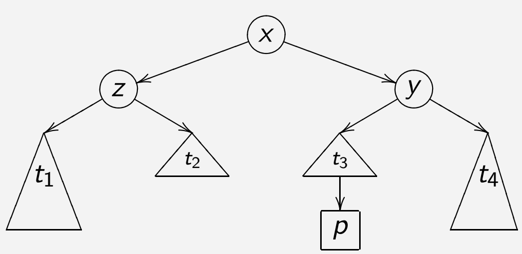


"Double" Rotation

- Node z fails the AVL test after adding node p:



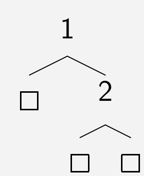
- Double rotation regains balance:



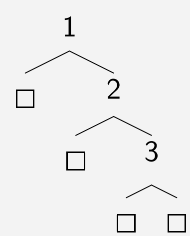
Examples of Insersion

"Single" Rotation

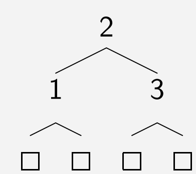
- From an empty tree: `Insert( 1 )`, `Insert( 2 )`



- `Insert( 3 )`



Is it an AVL? No!!!

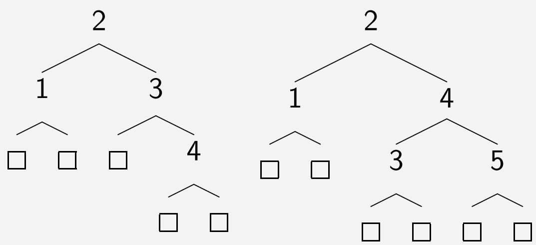


Now an AVL.

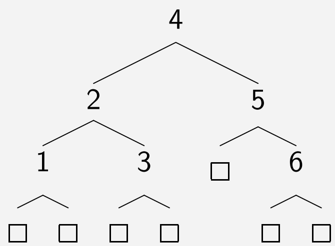
Examples of Insersion

“Single” Rotation (Cont)

- `Insert( 4 ), Insert( 5 )`



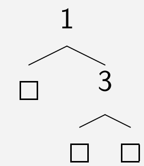
- `Insert( 6 )`



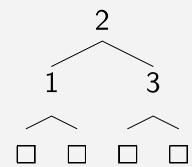
Examples of Insersion

“Double” Rotation

- From an empty tree: `Insert( 1 ), Insert( 3 )`

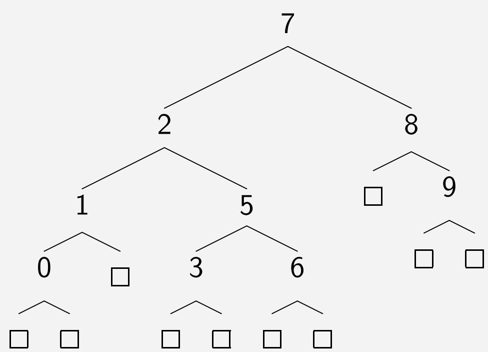


- `Insert( 2 )`



Examples of Insersion

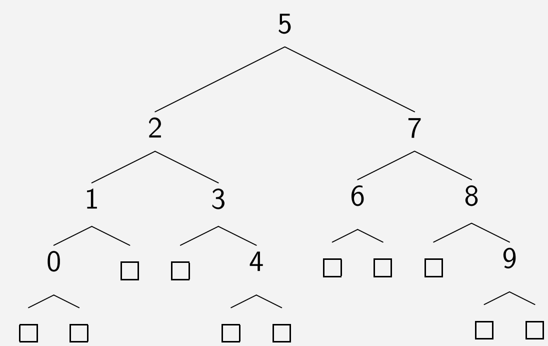
“Double” Rotation, Larger Example



- `Insert( 4 )`

Examples of Insersion

“Double” Rotation, Solution of the large example.

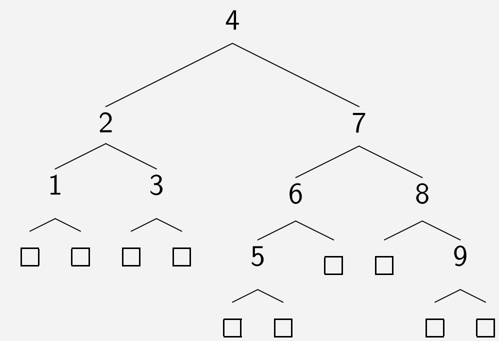


Cost of Insert

- How many rotations may be required for an `Insert`? (A double rotation counts as one rotation.)
  At most one!
- How expensive is a rotation?
  Constant
- Worst-case running time for `Insert`?
  Constant?
  NO!!!! Same cost as `Find`, hence $O(h)$, i.e. $O(\lg n)$.

Examples of Deletion

"Simple Rotation"

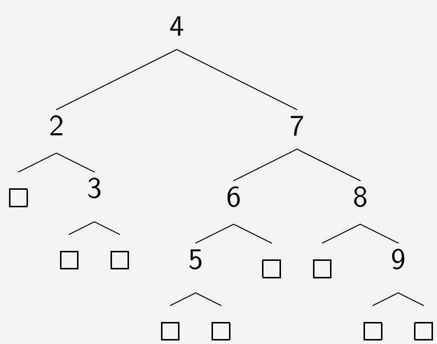


- `Delete( 1 )`
- `Delete( 2 )`

Examples of Deletion

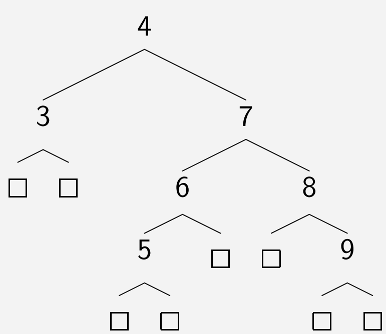
"Simple Rotation", solutions

`Delete( 1 )`



(Still an AVL)

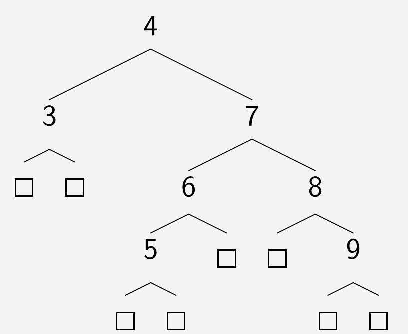
`Delete( 2 )`



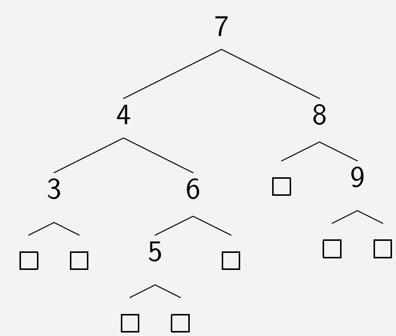
Not an AVL anymore!!!!

Examples of Deletion

"Simple Rotation", solutions (cont)



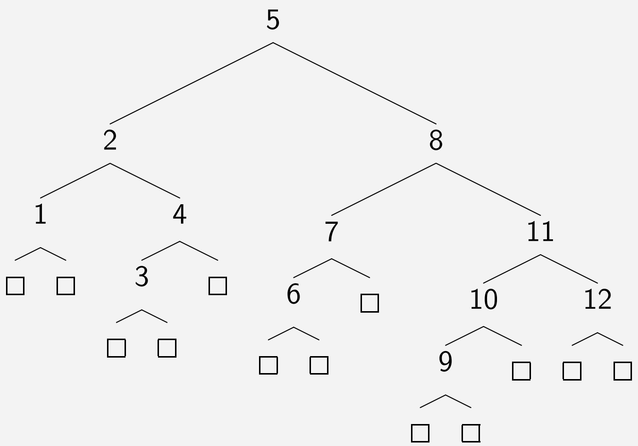
The unbalanced node is the root.
We perform a "Simple" rotation.



Now the tree is an AVL.

Delete

Larger Example

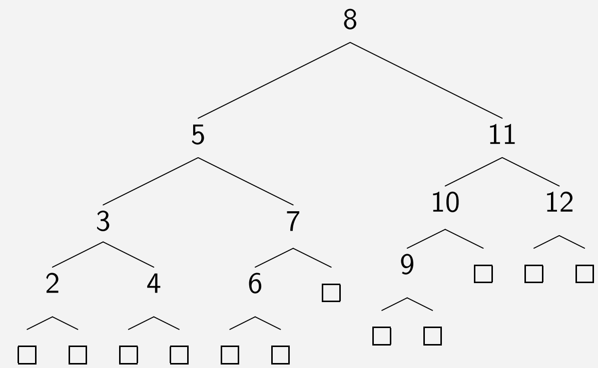


- `Delete( 1 )`

Delete

Solution of large Example

We need to rotate first around (2,4), and then around (5,8).



Cost of `Delete`

- How many rotations may be required for a `Delete`?
  at most h.
- How expensive is a rotation?
  constant (same rotation as for Insertsion)
- Worst-case running time for `Delete`?
  $O(h)$, i.e. $O(\lg n)$

Final Thoughts

- All major binary search tree operations have guaranteed worst-case $\Theta(\log n)$ performance
- Fairly large constant hidden in order notation
- Each internal node stores a condition code (or height)
- Condition code is usually represented by $\{-1, 0, 1\}$

Summary

- AVL Trees are Balanced Binary Search Trees.
- Their height is Logarithmic in their size.
- The time in which the operators are supported is
  - Search in time $O(\lg n)$
  - Insertion in time $O(\lg n)$ (not constant time!)
  - Deletion in time $O(\lg n)$

References:

- Goodrich and Tamassia: pp. 152-158
- Cormen, Leisersen, Rivest, Stein: pp. 296 (poorly covered)