Binary Search Trees and AVL Trees

Tufts University

Binary Search Trees - warmup questions

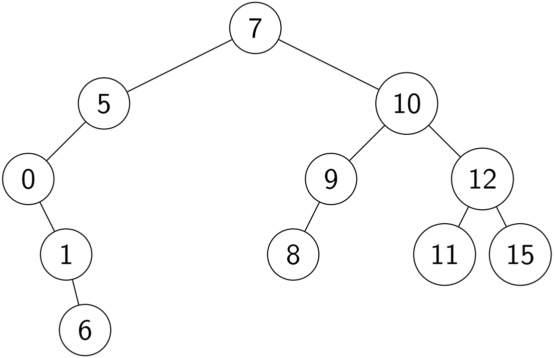
Q1: Create a BST by inserting the following keys in order: 7, 10, 5, 0, 9, 12, 15, 1, 4, 8, 11

Q2: What's the maximum height of a BST containing n elements?

Q3: What's the minimum height of a BST containing n elements?

Binary Search Trees

Container DS for fast insertion/deletions/search operations.

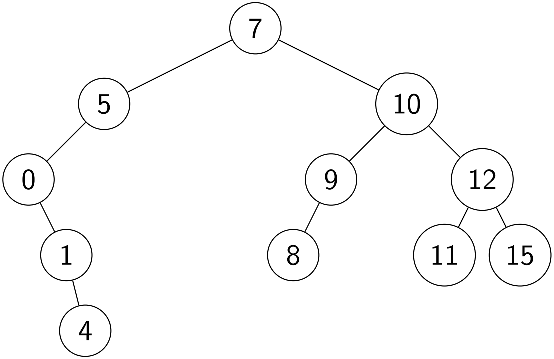


- Main structure is a **binary tree**
- Each node stores an object with a unique key
- **Main invariant**: objects in the right subtree of node *n* have smaller key than the object in *n*.

  Similarly, left subtree stores objects with larger keys

Binary Search Trees

Container DS for fast insertion/deletions/search operations.

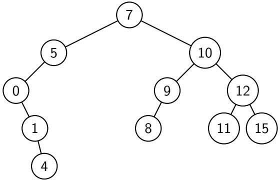


- Main structure is a **binary tree**
- Each node stores an object with a unique key

  Main invariant: objects in the right subtree of node *n* have smaller key than the object in *n*.

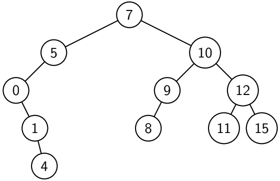
  Similarly, left subtree stores objects with larger keys

BST Operations



- Search
- Insert
- Predecessor/Successor
- Find minimum/maximum
- Delete
- (sorted) Print
- ...

Search/Insert



SEARCH(target)

$c \leftarrow$ root

While ($c \neq$ NIL)

If c.key = target $\Rightarrow$ Return true

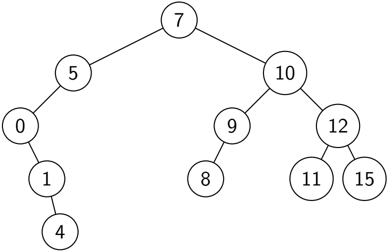
Else if $c.key <$ target $\Rightarrow c \leftarrow c$.left

Else $c \leftarrow c$.right

Return false

Exercise: modify code for an Insertion instead

Predecessor/Successor

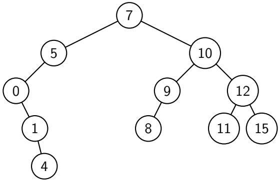


$\textsc{Successor}(x)$: smallest number in T that is $\geq x$

- $\textsc{Successor}(1)=1$
- $\textsc{Successor}(2)=4$
- $\textsc{Successor}(16)=\infty$
- ...

Strict Successor: smallest number in T that is $> x$

Further discussion



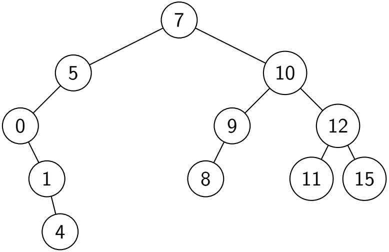
Let's find an algorithm for StrictSuccessor

- Prove that if y is the strict successor of x, then either y is an ancestor of x or x is an ancestor of y
- Give pseudocode for StrictSuccessor.
- Prove that your algorithm is correct. The above result may be helpful.

What would you change to compute Successor?

What about Predecessor/StrictPredecessor?

Finding Smallest/Largest element

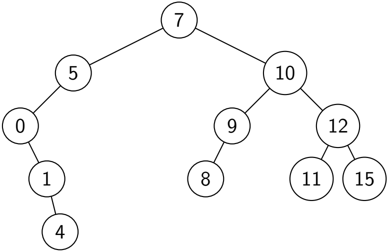


Maximum right $\Rightarrow$ right $\Rightarrow$ ... until no right child

Minimum left $\Rightarrow$ left $\Rightarrow$ ... until no left child

Exercise: Prove that strategy works!

Deletions



Exercise! Things to look out for

Code depends on degree!

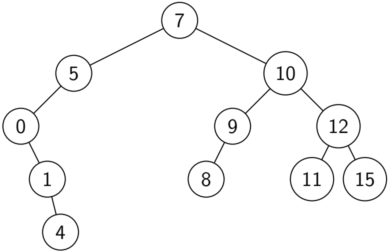
- Deleting a leaf is easy (just remove)
- Degree one → do a bypass
- Degree two → replace and recursively delete

Sorted print

Just do an **In-Order Traversal** of T

$\textsc{Inorder}(x)$

- if $x \neq$ NIL then
  - $\textsc{Inorder}(x.\mathit{left})$
  - Print(x)
  - $\textsc{Inorder}(x.\mathit{right})$



Exercise: What happens if we do preorder/postorder traversal?

Runtime?

What is the runtime of each of these operations?

- Search $\Theta(h)$
- Insert $\Theta(h)$
- Predecessor/Successor $\Theta(h)$
- Find minimum/maximum $\Theta(h)$
- Delete $\Theta(h)$
- (sorted) Print $\Theta(n)$
- ...

Where h is height of the tree $h = n$ in the worst case

Fun interlude: building a BST

- Remember heaps?

  Single insertion in $\Theta(\log n)$

  Insert n bottom-up in $\Theta(n)$
- Can we do something similar with BSTs? Yes!

Randomized-BST-Build(A)

Step 1: Randomly shuffle input array A

Step 2: For i from 1 to n insert $A[i]$ in BST

Runtime?

Example: Let's insert 6, 23, 12, 4, 20, 1, ...

Same as QuickSort!

First pivot = root of the BST
Second pivots = children of root
Etc ...

Correspondence between comparisons of both algorithms

Corollary: the expected height of the tree is $\Theta(\log n)$

$\Rightarrow$ n insertions will need $\Theta(n \log n)$ expected time

Balancing BSTs

Runtime of most operations in BSTs is $\Theta(h)$

If we insert all nodes at once we can have $h \approx \log n$

Can we maintain balance in other situations? Yes!

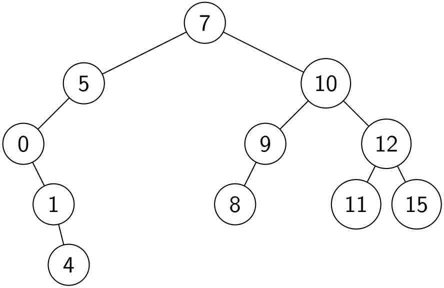
- 2–3 trees
- AA trees
- AVL trees $\Leftarrow$ this class
- B-tree
- Red–black tree
- and more!

Introducing AVL trees

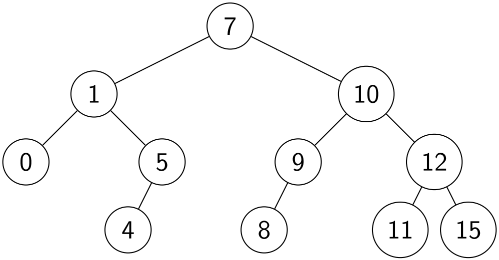
Adelson-Velsky and Landis trees (AVL trees for short) are BST with one more invariant:

- For any node, the difference in heights of subtrees is ≤ 1.

Is this tree an AVL tree? No!



AVL trees - Height



Lemma: the height of an AVL tree is $\Theta(\log n)$.

Corollary: most operations need $\Theta(\log n)$ time.

Proof: next lecture.

Exercise: prove **Leaf Ratio Lemma**: If x and y are leaves then $\frac{d(y)}{d(x)} \leq 2$

$d(x)$ denotes the number of edges from the root to x.

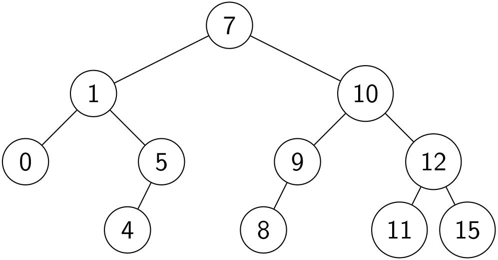
What changes in AVL trees?

Compared to BST, what do we need to change in ...

- Search
- Insert $\Leftarrow$
- Predecessor/Successor
- Find minimum/maximum
- Delete $\Leftarrow$
- (sorted) Print
- ...

Inserting in AVL tree

What changes in an insertion?

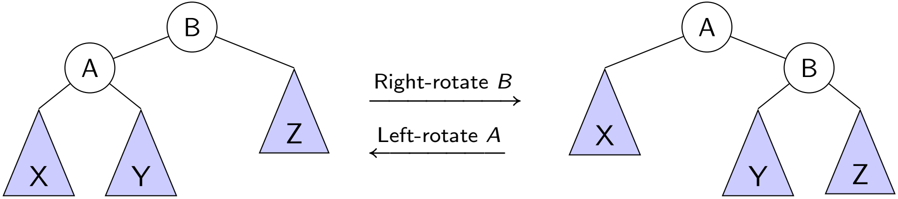


Ex:
Insert(6)
Insert(3)
Insert(2)

After insertion we may have to rebalance

Rotations

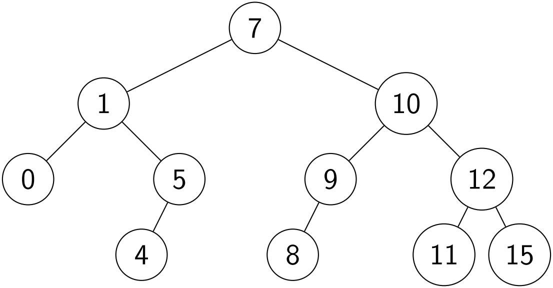
ROTATE(Node, direction)



- Rotation preserves BST properties because $X \leq A \leq Y \leq B \leq Z$
- Let's use it to preserve balance invariant in AVL trees

Insertion in AVL trees

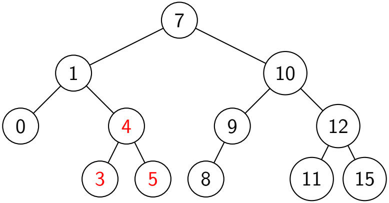
Back to INSERT(3)



5 is unbalanced (left child too deep) ⇒ Right rotate 5

AVL trees - Insert

INSERT(3)



5 is unbalanced (left child too deep) $\Rightarrow$ Right rotate 5

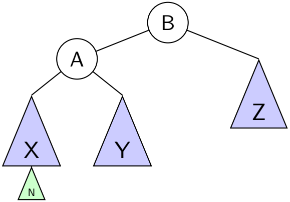
General rule

After inserting node N travel back to root checking balance.

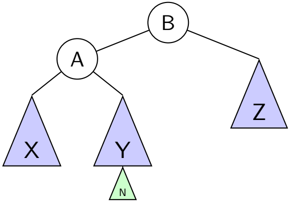
- If all nodes balanced $\Rightarrow$ done!
- Otherwise, stop at first (i.e. deepest) unbalanced ancestor N
- Wlog, assume N is in the left subtree of B. Two cases:

  N in left subtree of $A = B.left$
  N in right subtree of $A = B.left$

CASE 1

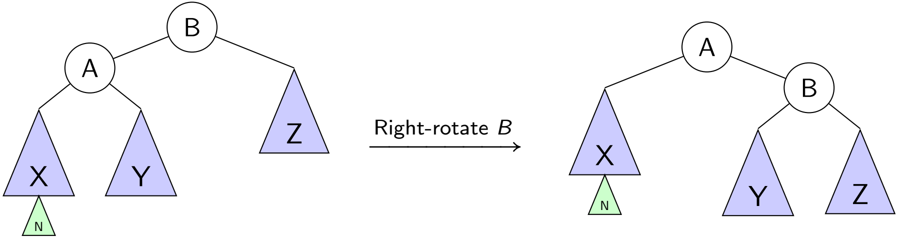


CASE 2



AVL trees - Insert: Case 1

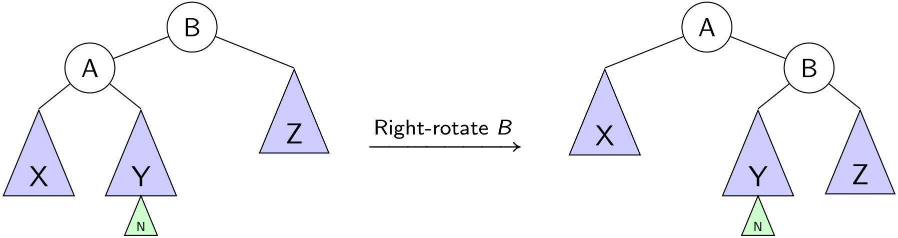
CASE 1



Done! No more height changes!

AVL trees - Insert: Case 2

CASE 2

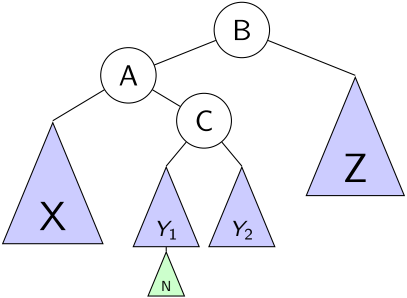


Case 2 cannot be fixed with 1 rotation ☹

AVL trees - Insert: Case 2 FIXED

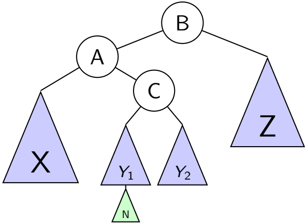
To fix Case 2 we "zoom into" $C == A.right$.

CASE 2A

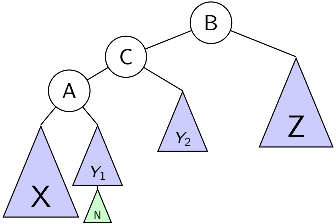


How can we rebalance this tree? Hint: use 2 rotations.

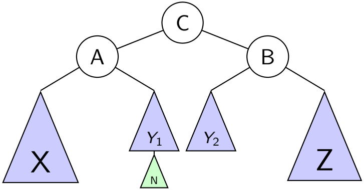
AVL trees - Insert: Case 2A



Left-rotate A $\longrightarrow$



$\downarrow$ Right-rotate B

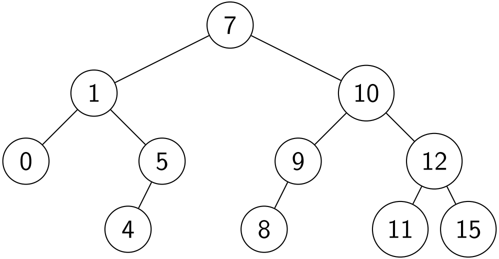


WOW!!!

No more height changes after rotation

Exercise: what if Y_1 is empty?

What about deletions?



- Perform standard BST-Delete. Check upwards for unbalance
- As before, only need to check ancestors
- Rotations can be more complicated than Insert

  (up to $\Theta(\log n)$ rotations)