CSCI 340
AVL Trees

Jon Lehuta

Northern Illinois University

October 8, 2019
AVL Trees - Outline

AVL Trees

Introduction

Rotation

Insertion

Deletion
INTRODUCTION

• Binary search trees are great, but can we make them better?
  • Binary search trees (BSTs) are fast as long as they are shallow.
  • Since the worst case search speed is the same as the depth of the deepest node, problems occur when one branch is much longer than the other.

![Balanced vs. unbalanced trees](image)

All three of these binary search trees contain the same data, but the ones on the outside have way more levels.

Since the complexity of the search algorithm is proportional to the height of the tree, the middle configuration is much more desirable.

Figure: Balanced vs. unbalanced trees.
Types of Trees

Graph

Tree

Static
Dynamic

Game tree
Search Tree
Heap

BST

AVL Tree
2-3 Tree
Red-Black Tree
AVL trees are

- Named after Adelson-Velskii and Landis
- The first dynamically balanced trees to be proposed
- Binary search trees with a *balance condition* in which the subtrees of each node are allowed to differ by at most 1 in their height.
An AVL tree has the following properties:

- Subtrees of each node can differ by at most 1 in their height.
- Every subtrees must be an AVL tree.
AVL tree?

Figure: The left tree is AVL. The one on the right does not qualify. Why not?
AVL vs non-AVL

Figure: Examples of AVL vs. non-AVL BST’s
Insertion and Deletion

Operations on AVL trees (e.g., search) are implemented as they were on binary trees, except for insertion and deletion which may need to perform additional computations.

One more thing is associated with each node $x$ in the AVL tree $\Rightarrow$

$$\text{balanceFactor} = \text{height(leftSubtree)} - \text{height(rightSubtree)}$$

The balance factor of any node of an AVL tree may be one of $\{-1, 0, +1\}$

The temporarily recomputed balance factor of a node after an insertion will be in the range $[-2, +2]$.

After inserting a node, it is necessary to check each of the node’s ancestors (in sequence, parent, grandparent etc.) to fix any violations of the balance condition that may have been introduced by the new node.

If a node is unbalanced, can perform single or double rotation or reconstruction to correct balance.
**Tree rotation**

An operation on a binary tree that changes the structure without interfering with the order of the elements.

A tree rotation moves one node or subtree up in the tree and one down. It is used to change the shape of the tree, and in particular to decrease its height by moving smaller subtrees down and larger subtrees up, resulting in improved performance of many tree operations.

![Diagram of simple tree rotation](image)

**Figure: Simple rotation**
Types of rotation used in AVL trees

**LL** - Single right rotation

**RR** - Single left rotation

**LR** - Left then right

**RL** - Right then left

Figure: The four types of rotation needed for AVL trees.
**Rotation steps**

- **IF tree is right heavy (heights differ by more than 1)**
  - **IF tree’s right subtree is left (L) heavy (RL)**
    - R rotation to the subtree and then L rotation to the node (double rotation)
  - **ELSE IF tree’s right subtree is right (R) heavy (RR)**
    - L rotation to the unbalanced node (single rotation)

- **ELSE IF tree is left heavy**
  - **IF tree’s left subtree is right heavy (LR)**
    - L rotation to the subtree and then R rotation to the node (double rotation)
  - **ELSE IF tree’s left subtree is left heavy (LL)**
    - R rotation to the unbalanced node (single rotation)

**Note:** During a rotation, the moved children must be connected to a parent and not another child.
To insert into an AVL tree, there are two steps:

1. Insert the new node where it should be based on BST rules.
2. Rotate to fix any imbalances that may have occurred from that insertion.

Figure: An example of AVL insertion algorithm
**AVL Trees**  Insertion

**Insert 1, 2, 3, 4, 5, 6, 7 into an AVL Tree**

1. Insert 1
2. Insert 2
3. Insert 3
4. Insert 4
5. Insert 5
6. Insert 6
7. Insert 7

**Figure:** Tree after each number is inserted.
**Insert 1, 2, 3, 4, 5, 0, 7, 6 into an AVL Tree**

Figure: AVL insertion, with intermediate steps shown.
**Inserting 20, 40, 60**

Figure: Example of insertion (with rotation): 20, 40, 60
Inserting 20, 40, 60, 10, 15

Figure: Example of insertion (with rotation): 20, 40, 60, 10, 15
Inserting 20, 40, 60, 10, 15, 12, 11

Figure: Example of insertion (with rotation) 20, 40, 60, 10, 15, 12, 11
Inserting 50, 25, 10, 5, 7, 3, 30, 20, 8, 15

Figure: Inserting 50, 25, 10, 5, 7, 3, 30, 20, 8, 15 into an AVL Tree
Deletion from AVL Trees

We first do the normal BST deletion:

- 0 children: just delete it
- 1 child: delete it, connect child to parent
- 2 children: put predecessor/successor in your place, delete predecessor/successor

After deleting the node, the resulting tree might no longer be an AVL tree. As in the case of insertion into a non-AVL tree, the height at the top could decrease or increase by 1.

Which nodes’ heights may have changed:

- 0 children: path from deleted node to root
- 1 child: path from deleted node to root
- 2 children: path from deleted successor leaf to root

Similar rotations to insert, but in case there is an imbalance at a node and the left subtree height is equal to right subtree height, then perform either rotation (single or double rotate).
Deletion Example 1

Initial tree (is AVL)

Remove 15 first

Remove 12 next

Done.

Imbalance, rotate to fix

Figure: Delete 15 and 12 using INORDER SUCCESSOR logic
Deletion Example 2

Figure: Deleting the node with key 4
What type of rotation to fix?

This tree is not an AVL tree. It does not meet the balance condition. Fix with (a) or (b).

(a) A single left rotation with the highlighted nodes
(b) OR a RL rotation with these highlighted nodes.

After the single, left rotation.

After the RL rotation.

Figure: Two ways to fix the tree shown on the left.
Pros and Cons of AVL Trees

Arguments for AVL trees:

• Fast search operations because trees are always balanced.
• The height balancing adds no more than a constant factor to the speed of insert and delete.

Arguments against AVL trees:

• More space used to store height field
• Rebalancing takes a extra time → post-search insertion/deletion slower than ordinary BST on random data
• Most large searches are done in database-like systems on disk and use other structures (e.g., B-trees).
USEFUL LINKS

AVL Tree animation
https://www.cs.usfca.edu/~galles/visualization/AVLtree.html

AVL Tree Insertion https://www.youtube.com/watch?v=rbg7Qf8GkQ4

The AVL Tree Rotations Tutorial

AVL Tree insert algorithm
http://www.geeksforgeeks.org/avl-tree-set-1-insertion/

AVL Tree delete algorithm
ACKNOWLEDGMENTS


Prof. Sin-Min Lee AVL notes
http://www.cs.sjsu.edu/faculty/lee/cs146/cs146.htm

Prof. Freedman AVL notes

Prof. Zohu AVL notes

Dan Grossman AVL notes

https://en.wikipedia.org/wiki/AVL_tree