COMP 3170 - Analysis of Algorithms & Data Structures

Shahin Kamali

Lecture 10 - Jan. 24, 2018

CLRS 12.2, 12.3, 13.2, read problem 13-3

University of Manitoba
A height-balanced BST has depth $\Theta(\log n)$.

In order to perform dictionary operations in $\Theta(\log n)$, we maintain a height-balanced BST.
AVL Trees

- Introduced by Adel’son-Vel’skii and Landis in 1962
- An AVL Tree is a height-balanced BST
  - The heights of the left and right subtree differ by at most 1.
  - (The height of an empty tree is defined to be $-1$.)
- At each non-empty node, we store $height(R) - height(L) \in \{-1, 0, 1\}$:
  - $-1$ means the tree is left-heavy
  - $0$ means the tree is balanced
  - $1$ means the tree is right-heavy
- We could store the actual height, but storing balances is simpler and more convenient.
To perform $\text{insert}(T, k, v)$:

- First, insert $(k, v)$ into $T$ using usual BST insertion
- Then, move up the tree from the new leaf, updating balance factors.
- If the balance factor is $-1$, $0$, or $1$, then keep going.
- If the balance factor is $\pm 2$, then call the $\text{fix}$ algorithm to “rebalance” at that node.
How to “fix” an unbalanced AVL tree

**Goal**: change the *structure* without changing the *order*

Notice that if heights of $A, B, C, D$ differ by at most 1, then the tree is a proper AVL tree.
This is a right rotation on node \( z \); apply when balance of \( z \) is -2 and balance of \( y \) is -1 or 0.
Right Rotation

This is a right rotation on node z; apply when balance of z is -2 and balance of y is -1 or 0.

Note: Only two edges need to be moved, and two balances updated.
Left Rotation

This is a *left rotation* on node $z$; apply when balance of $z$ is 2 and balance of $y$ is 1 or 0.

![Diagram of left rotation]

Again, only two edges need to be moved and two balances updated.
### Pseudocode for rotations

**rotate-right** (\( T \))

- \( T \): AVL tree
- returns rotated AVL tree
  1. \( newroot \leftarrow T.left \)
  2. \( T.left \leftarrow newroot.right \)
  3. \( newroot.right \leftarrow T \)
  4. \textbf{return} newroot

**rotate-left** (\( T \))

- \( T \): AVL tree
- returns rotated AVL tree
  1. \( newroot \leftarrow T.right \)
  2. \( T.right \leftarrow newroot.left \)
  3. \( newroot.left \leftarrow T \)
  4. \textbf{return} newroot
Double Right Rotation

This is a *double right rotation* on node $z$; apply when balance of $z$ is -2 and balance of $y$ is 1.

First, a left rotation on the left subtree ($y$).
Double Right Rotation

This is a *double right rotation* on node $z$; apply when balance of $z$ is -2 and balance of $y$ is 1.

First, a left rotation on the left subtree ($y$).
Second, a right rotation on the whole tree ($z$).
Double Left Rotation

This is a *double left rotation* on node $z$; apply when balance of $z$ is 2 and balance of $y$ is -1.

Right rotation on right subtree ($y$), followed by left rotation on the whole tree ($z$).
Fixing a slightly-unbalanced AVL tree

**Idea:** Identify one of the previous 4 situations, apply rotations

\[
\text{fix}(T)
\]

\(T: \) AVL tree with \(T.balance = \pm 2\)

returns a balanced AVL tree

1. \(\text{if } T.balance = -2 \text{ then}\)
2. \(\text{if } T.left.balance = 1 \text{ then}\)
3. \(T.left \leftarrow \text{rotate-left}(T.left)\)
4. \(\text{return rotate-right}(T)\)
5. \(\text{else if } T.balance = 2 \text{ then}\)
6. \(\text{if } T.right.balance = -1 \text{ then}\)
7. \(T.right \leftarrow \text{rotate-right}(T.right)\)
8. \(\text{return rotate-left}(T)\)
AVL Trees

AVL Tree Operations

search: Just like in BSTs, costs $\Theta(height)$

insert: Shown already, total cost $\Theta(height)$
fix will be called at most once.

delete: First search, then swap with successor (as with BSTs),
then move up the tree and apply fix (as with insert).
fix may be called $\Theta(height)$ times.
Total cost is $\Theta(height)$. 
Example: \textit{insert}(8)
AVL Trees

AVL tree examples

**Example**: insert(8)
AVL Trees

AVL tree examples

Example: $\text{insert}(8)$
Example: \textit{insert}(8)
**AVL Trees**

AVL tree examples

**Example**: `insert(8)`

![AVL Tree Example](image)
AVL Trees

AVL tree examples

Example: delete(22)
**Example:** *delete*(22)
AVL Trees

AVL tree examples

Example: delete(22)
Example: delete(22)
AVL Trees

AVL tree examples

Example: delete(22)
Since AVL-trees are height-balanced, their height is $\Theta(\log n)$ (previous class)

Search can be done as before (no need for rebalancing)

$\text{Insert}(x)$ takes $\Theta(\log n)$ and involves at most one fix.
Since AVL-trees are height-balanced, their height is $\Theta(\log n)$ (previous class).

Search can be done as before (no need for rebalancing).

Insert($x$) takes $\Theta(\log n)$ and involves at most one fix.

Delete($x$) takes $\Theta(\log n)$ and involves at most $\Theta(\log n)$ fixes.

$\Rightarrow$ search, insert, delete all cost $\Theta(\log n)$. 