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
Binary Search Trees (review)

- 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 $\text{height}(R) - \text{height}(L) \in \{-1, 0, 1\}$:
  - $-1$ 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.
AVL insertion

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.
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.

\[ \text{Note: Only two edges need to be moved, and two balances updated.} \]
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.

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

\[
\text{rotate-right}(T) \\
T: \text{ AVL tree} \\
\text{return rotated AVL tree}
\]

1. \( \text{newroot} \leftarrow T.\text{left} \)
2. \( T.\text{left} \leftarrow \text{newroot}.\text{right} \)
3. \( \text{newroot}.\text{right} \leftarrow T \)
4. \( \text{return newroot} \)

\[
\text{rotate-left}(T) \\
T: \text{ AVL tree} \\
\text{return rotated AVL tree}
\]

1. \( \text{newroot} \leftarrow T.\text{right} \)
2. \( T.\text{right} \leftarrow \text{newroot}.\text{left} \)
3. \( \text{newroot}.\text{left} \leftarrow T \)
4. \( \text{return newroot} \)
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 ($y$).
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

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

\[T: \text{AVL tree with } T.\text{balance} = \pm 2\]

returns a balanced AVL tree

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

**search**: Just like in BSTs, costs $\Theta(\text{height})$

**insert**: Shown already, total cost $\Theta(\text{height})$

$\text{fix}$ will be called *at most once*.

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

Example:

```
22
-1
10
  4
    6
      16
      0
  14
    13
      18
      -1
    0
  31
    28
      0
    37
      46
      0
  0
```

COMP 3170 - Analysis of Algorithms & Data Structures 13 / 14
AVL tree analysis

- 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 $\Theta(\log n)$. 