COMP251: Binary search trees, AVL trees & AVL sort

Jérôme Waldispühl
School of Computer Science
McGill University

From Lecture notes by E. Demaine (2009)
Midterm #1

• Wednesday September 30.

• Duration of the exam:
  o Theory: 1h30
  o In practice: 3h30. The latter accounts for technical issues (i.e. internet) and barrier of learning.

• When? We will open a timeframe of 24h during which you can take the exam. **WARNING:** submissions will start on Sep. 30 at 9:00am (EDT) and *close* on Oct. 1 at 9:00am (EDT). If you start later than 5h30am (EDT) on Oct 1, you will have less than 3h30 to complete the exam. Submit early!

• How? We will use crowdmark.

• We will send a list of problems and will implement a practice question on crowdmark to allow you to test the platform.
Outline

• Review of binary search trees
• AVL-trees
• Rotations
• BST & AVL sort
Binary search trees (BSTs)

- $T$ is a rooted binary tree
- Key of a node $x \geq$ keys in its left subtree.
- Key of a node $x \leq$ keys in its right subtree.
Operations on BSTs

• Search(T,k): $\Theta(h)$
• Insert(T,k): $\Theta(h)$
• Delete(T,k): $\Theta(h)$

Where $h$ is the height of the BST.
Height of a tree

Height(n): length (#edges) of longest downward path from node n to a leaf.

Height(x) = 1 + max( height(left(x)), height(right(x)) )
Example

$h(a) = ?$

$= 1 + \max( h(b), h(g) )$

$= 1 + \max( 1 + \max( h(c), h(d) ), 1 + h(h) )$

$= 1 + \max( 1 + \max( 0, h(d) ), 1 + 0 )$

$= 1 + \max( 1 + \max( 0, 1 + h(e) ), 1 )$

$= 1 + \max( 1 + \max( 0, 1 + (1 + h(f)) ), 1 )$

$= 1 + \max( 1 + \max( 0, 1 + (1 + 0) ), 1 )$

$= 1 + \max( 3, 1 )$

$= 4$
Height vs. Depth

Good vs. Bad BSTs

Balanced
\[ h = \Theta(\log n) \]

Unbalanced
\[ h = \Theta(n) \]
AVL trees

**Definition:** BST such that the heights of the two child subtrees of any node differ by at most one.

- AVL trees are self-balanced binary search trees.
- Insert, Delete & Search take $O(\log n)$ in average and worst cases.
- To satisfy the definition, the height of an empty subtree is -1.
Height of a AVL tree

\[ N_h = \text{minimum \#nodes in an AVL tree of height } h. \]

Let \( k = \frac{h}{2} - 1 \):

\[ N_h > 2^{k} \cdot N_{h-2k} \]

\[ N_h > 2^{(\frac{h}{2}-1)} \cdot N_1 \]

\[ N_h > 2^{(\frac{h}{2}-1)} \]

\[ N_h = 1 + N_{h-1} + N_{h-2} \]

\[ > 2 \cdot N_{h-2} \]

\[ \Rightarrow N_h > \Theta(2^{h/2}) \]

\[ \Rightarrow h < 2 \cdot \log N_h \]

\[ \Rightarrow h = O(\log n) \]

(Note: a tighter bound can be found using Fibonacci numbers)
Balance factor

\[ \beta_N = \begin{cases} 
N_{h-1} & : \text{Left tree is higher (left-heavy)} \\
N_{h-1} & : \text{Balanced} \\
N_{h-1} & : \text{Right tree is higher (right-heavy)} 
\end{cases} \]

\[ N_{h-1} > 1 \]

Violate AVL property
Insert in AVL trees

1. Insert as in standard BST
2. Restore AVL tree properties
Insert in AVL trees

```python
Insert(T, 15)
```
Insert in AVL trees

Insert(T, 15)

How to restore AVL property?
Rotations change the tree structure & **preserve the BST property**.

**Proof:** elements in B are $\geq x$ and $\leq y$...
Example (right rotation)
Example: Insert in AVL trees

Right rotation at 27
Example: Insert in AVL trees

Insert(T, 50)
RotateRight(T, 57)

How to restore AVL property?
Example: Insert in AVL trees

Left rotation at 43

We remove the zig-zag pattern

RotateLeft(T,43)
Example: Insert in AVL trees

Right rotation at 57

AVL property restored!

RotateRight(T,57)
Algorithm: Insert in AVL trees

1. Suppose x is lowest node violating AVL
2. If x is right-heavy:
   • If x’s right child is right-heavy or balanced: Left rotation (case A)
   • Else: Right followed by left rotation (case B)
3. If x is left-heavy:
   • If x’s left child is left-heavy or balanced: Right rotation (symmetric of case A)
   • Else: Left followed by right rotation (sym. of case B)
4. then continue up to x’s ancestors.
Proof: Case A

Left rotation

$h - 1 \rightarrow A$
$h + 1 \rightarrow B$
$h - 1 \rightarrow C$

Left rotation

$h - 1 \rightarrow A$
$h - 1 \rightarrow B$
$h - 1 \rightarrow C$

Left rotation

$h - 1 \rightarrow A$
$h + 1 \rightarrow B$
$h - 1 \rightarrow C$
Proof: Case B

Right rotation at y &
Left rotation at x

\[
\begin{align*}
A & \quad h-1 \\
B & \quad h \\
C & \quad h \\
D & \quad h+1 \\
X & \quad h-1 \\
Y & \quad h \\
Z & \quad h \\
\end{align*}
\]

\[
\begin{align*}
A & \quad h-1 \\
B & \quad h \\
C & \quad h \\
D & \quad h-1 \\
X & \quad h-1 \\
Y & \quad h \\
Z & \quad h \\
\end{align*}
\]
Proof: Case B

Right rotation at y

Left rotation at x
Running time AVL insertion

- Insertion in $O(h)$
- At most 2 rotations in $O(1)$
- Running time is $O(h) + O(1) = O(h) = O(\log n)$ in AVL trees.
Sorting with BSTs

1. BST sort
   • Simple method using BSTs
   • Problem: Worst case $O(n^2)$

2. AVL sort
   • Use AVL trees to get $O(n \cdot \log n)$
In-order traversal & BST

inorderTraversal(treeNode x)
  inorderTraversal(x.leftChild);
  print x.value;
  inorderTraversal(x.rightChild);

• Print the nodes in the left subtree (A), then node x, and then the nodes in the right subtree (B)
• In a BST, keys in A ≤ x, and keys in B ≥ x.
• In a BST, it prints first keys ≤ x, then x, and then keys ≥ x.
In-order traversal & BST

8, 12, 15, 20, 27, 36, 43, 57

All keys come out sorted!
BST sort

1. Build a BST from the list of keys (unsorted)

2. Use in-order traversal on the BST to print the keys.

| 36 | 12 | 8 | 57 | 43 | 27 |

→ 8, 12, 27, 36, 43, 57

Running time of BST sort: insertion of n keys + tree traversal.
Running time of BST sort

- In-order traversal is $\Theta(n)$
- Running time of insertion is $O(h)$

**Best case:** The BST is always balanced for every insertion.

$$\Omega(n \log(n))$$

**Worst case:** The BST is always un-balanced. All insertions on same side.

$$\sum_{i=1}^{n} i = \frac{n \cdot (n-1)}{2} = O(n^2)$$
AVL sort

Same as BST sort but use AVL trees and AVL insertion instead.

• Worst case running time can be brought to $O(n \log n)$ if the tree is always balanced.
• Use AVL trees (trees are balanced).
• Insertion in AVL trees are $O(h) = O(\log n)$ for balanced trees.