Lecture 6: Balanced Binary Search Trees

Lecture Overview

- The importance of being balanced
- AVL trees
  - Definition and balance
  - Rotations
  - Insert
- Other balanced trees
- Data structures in general
- Lower bounds

Recall: Binary Search Trees (BSTs)

- rooted binary tree
- each node has
  - key
  - left pointer
  - right pointer
  - parent pointer

See Fig. 1

Figure 1: Heights of nodes in a BST
BST property (see Fig. 2).

height of node = length (# edges) of longest downward path to a leaf (see CLRS B.5 for details).

The Importance of Being Balanced:

- BSTs support insert, delete, min, max, next-larger, next-smaller, etc. in $O(h)$ time, where $h = \text{height of tree} (= \text{height of root})$.
- $h$ is between $\lg n$ and $n$: Fig. 3.

balanced BST maintains $h = O(\lg n) \Rightarrow$ all operations run in $O(\lg n)$ time.
AVL Trees: Adel’son-Vel’skii & Landis 1962

For every node, require heights of left & right children to differ by at most ±1.

- treat nil tree as height -1
- each node stores its height (DATA STRUCTURE AUGMENTATION) (like subtree size) (alternatively, can just store difference in heights)

This is illustrated in Fig. 4

![AVL Tree Concept](image)

Balance:

Worst when every node differs by 1 — let $N_h = (\text{min.}) \# \text{ nodes in height-}h \text{ AVL tree}$

$\implies N_h = N_{h-1} + N_{h-2} + 1$

$> 2N_{h-2}$

$\implies N_h > 2^{h/2}$

$\implies h < \log N_h$

Alternatively:

$N_h > F_h$ (nth Fibonacci number)

- In fact $N_h = F_{n+1} - 1$ (simple induction)

- $F_h = \frac{\varphi^h}{\sqrt{5}}$ rounded to nearest integer where $\varphi = \frac{1 + \sqrt{5}}{2} \approx 1.618$ (golden ratio)

$\implies$ max. $h \approx \log_{\varphi} n \approx 1.440 \log n$
AVL Insert:

1. insert as in simple BST
2. work your way up tree, restoring AVL property (and updating heights as you go).

Each Step:

- suppose $x$ is lowest node violating AVL
- assume $x$ is right-heavy (left case symmetric)
- if $x$’s right child is right-heavy or balanced: follow steps in Fig. 5

![Figure 5: AVL Insert Balancing](attachment:image.png)

- else: follow steps in Fig. 6

![Figure 6: AVL Insert Balancing](attachment:image.png)

- then continue up to $x$’s grandparent, greatgrandparent . . .
Example: An example implementation of the AVL Insert process is illustrated in Fig. 7.

Comment 1. In general, process may need several rotations before done with an Insert.

Comment 2. Delete(-min) is similar — harder but possible.
AVL sort:

- insert each item into AVL tree \( \Theta(n \lg n) \)
- in-order traversal \( \Theta(n) \)

Balanced Search Trees:

There are many balanced search trees.

AVL Trees Adel’son-Velsii and Landis 1962
B-Trees/2-3-4 Trees Bayer and McCreight 1972 (see CLRS 18)
BB[\( \alpha \)] Trees Nievergelt and Reingold 1973
Red-black Trees CLRS Chapter 13
(A) — Splay-Trees Sleator and Tarjan 1985
(R) — Skip Lists Pugh 1989
(A) — Scapegoat Trees Galperin and Rivest 1993
(R) — Treaps Seidel and Aragon 1996

(R) = use random numbers to make decisions fast with high probability
(A) = “amortized”: adding up costs for several operations \( \implies \) fast on average

For example, Splay Trees are a current research topic — see 6.854 (Advanced Algorithms) and 6.851 (Advanced Data Structures)

Big Picture:

Abstract Data Type(ADT): interface spec.

vs.

Data Structure (DS): algorithm for each op.

There are many possible DSs for one ADT. One example that we will discuss much later in the course is the “heap” priority queue.

<table>
<thead>
<tr>
<th>Priority Queue ADT</th>
<th>heap</th>
<th>AVL tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q = new-empty-queue()</td>
<td>( \Theta(1) )</td>
<td>( \Theta(1) )</td>
</tr>
<tr>
<td>Q.insert(x)</td>
<td>( \Theta(\lg n) )</td>
<td>( \Theta(\lg n) )</td>
</tr>
<tr>
<td>x = Q.deletemin()</td>
<td>( \Theta(\lg n) )</td>
<td>( \Theta(\lg n) )</td>
</tr>
<tr>
<td>x = Q.findmin()</td>
<td>( \Theta(1) )</td>
<td>( \Theta(\lg n) \to \Theta(1) )</td>
</tr>
<tr>
<td>Predecessor/Successor ADT</td>
<td>heap $\Theta(1)$</td>
<td>AVL tree $\Theta(1)$</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>$S = \text{new-empty}()$</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>$S.\text{insert}(x)$</td>
<td>$\Theta(\lg n)$</td>
<td>$\Theta(\lg n)$</td>
</tr>
<tr>
<td>$S.\text{delete}(x)$</td>
<td>$\Theta(\lg n)$</td>
<td>$\Theta(\lg n)$</td>
</tr>
<tr>
<td>$y = S.\text{predecessor}(x)$ → next-smaller</td>
<td>$\Theta(n)$</td>
<td>$\Theta(\lg n)$</td>
</tr>
<tr>
<td>$y = S.\text{successor}(x)$ → next-larger</td>
<td>$\Theta(n)$</td>
<td>$\Theta(\lg n)$</td>
</tr>
</tbody>
</table>