1 Content

The major content of the lecture:

- Admin (key/values pairs, operations, trees etc.)
- Rotations
- Using Rotations
  - AVL trees
  - Splay trees

2 Introduction

Given a set of pairs (key, value) (such as student names, ID#'s) that we will access using key (ID#) and realize follow operations:

- Insert (key, value)
- Delete (key)
- Find (key) (return value)
- Min (-)
- Max (-)

For unsorted array:

- Insert: add the element at end
- Find: search every element
- Delete find: remove, then compress

For sorted array:

- Insert: find the location of the element, push back the rest ones
3 Rotations

Definition 1. For every node $x$ in a binary search tree, all left descendants
have key $< x$, and all right descendants have key $> x$.

Example 2. Insert 15, 6, 3, 2, 7, 4, 13, 18, 20, 17, 9.

Definition 3. The height of a binary search tree is number of levels below root.

Example 4. The height of the binary search tree in Figure 3.1 is 4.

Proposition 5. Let $h$ be height of a binary search tree, then the operation time
of insert, delete, find key, maximum and minimum all take $O(h)$.

Example 6. Insert 1, 2, 3, 4, 5, 6 which generates the worst operation case $O(n)$.

Definition 7. A balanced binary search tree is any node-based binary search
tree that has smallest possible height.
Definition 8. Rotation is a local operation in a search tree that preserves the inorder key ordering. There are two kinds of rotations: left rotations and right rotations.

Figure 3.4: Rotation Operation

4 Using Rotations

4.1 AVL Trees

Definition 9. Invariant means the height of left child and height of right child differ by $\leq 1$.

- Insert: find where key should go, rebalance using rotation.
- Delete: won’t go over
- Find, Min, Max: use normal Binary Tree Search method

Idea: Keep track of height at each node. Look at the lowest node $A$ which breaks the invariant.

Three Cases:

- Case 1: right-right ($A$’s right child is longer, and $A$ is the right child of its parent) or left-left case, then only 1 rotation is necessary

Figure 4.1: Right-Right Case ($X>Y$)
• Case 2: right-left or left-right, then 2 rotations are necessary

![Diagram](image1.png)

Figure 4.2: Right-Left Case (X < Y)

• Case 3: root unbalanced, rotate once

4.2 Splay Tree

**Definition 10.** A splay tree is a self-adjusting binary tree with the additional property that recently accessed elements are quick to access again.

**Proposition 11.** Worst case time $O(n)$, but average operation cost is $O(\log n)$. For any $k$ operations starting from an empty tree which never has size $> n$ is $O(k \log n)$ even in worst case.

The algorithm $\text{Find}(key)$ requires two basic steps:

- Step 1: Find as in normal Binary Search Tree
- Step 2: Splay-rotate to move key to top

There are three cases of the key’s position in the procedure to fulfill splay.

- **Case 1:** right-left or left-right, “zig-zag”

![Diagram](image2.png)

Figure 4.3: Right-Left Case

- **Case 2:** right-right or left-left, “zig-zig”
Figure 4.4: Left-Left Case

- Case 3: Key is child of root, rotate up

**Example 12.** `Find(7)`.

**Example 13.** `Find(1)`.
Figure 4.6: Example Find(1)