AVL Trees - Recitation 7

December 6, 2022

   In this recitation we will review the new data structures you've seen - Binary search trees, and specifically AVL trees. We'll revise the different operations, go over the important concepts of balance factor and rotations and see some examples. If there's time left - we will prove that the height of an AVL tree is $O(\log(n))$.

1 AVL trees

Reminders:

> **Binary Tree.**
>
> A binary tree is a tree (T, r) with $r \in V$, such that $\deg(v) \leq 2$ for any $v \in V$.

> **Height.**
>
> A tree's height $h(T)$ (sometimes $h(r)$) is defined to be the length of the longest simple path from r to a leaf.

> **Binary Search Tree.**
>
> A binary search tree is a binary tree (T, r) such that for any node x (root of a subtree) and a node in that subtree y:
>
>   1. if y is in the left subtree of x then $y.key \leq x.key$
>
>   2. if y is in the right subtree of x then $x.key < y.key$
>
> Note that this is a (relatively) local property.

For example:
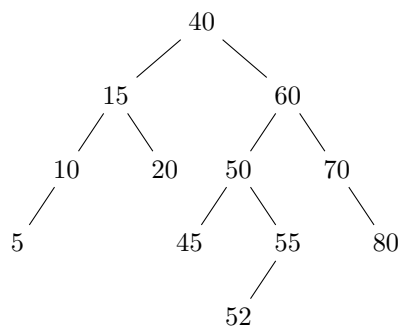
remark Go over the properties, calculate the tree's height. Make sure you understand the definitions!

Last time, we have seen some operations that can be performed on BSTs, and proved correctness for some of them. These were: $Search(x), Min(T), Max(T), Pred(x), Succ(x), Insert(x), Delete(x)$. All of these operations take $O(h(T))$ in the worst case.

The main two operations that may cause problems are $Insert$ and $Delete$, as they change the tree's height (consider inserting $81, 82, 83, 84$ to our working example). To address this problem, we introduce another filed: for each node v add a field of $h(v) =$ the height of the subtree who's root is v. This allows us to maintain the AVL property:

AVL Tree.

An AVL tree is a balanced BST, such that for any node x, it's left and right subtrees' height differ in no more than 1. In other words:

$$|h(left(x)) - h(right(x))| \leq 1$$

This field allows us to calculate the Balance Factor for each node in $O(1)$:

Balance Factor.

For each node $x \in T$, it's Balance Factor is defined

$$hd(x) := h(left(x)) - h(right(x))$$

In AVL trees, we would like to maintain $|hd(x)| \leq 1$

Example. For our working example, the node 60's hd is $h(50) - h(70) = 1$, and $hd(50) = h(45) - h(55) = -1$. You can check and see that this is an AVL tree. So to make sure that we can actually maintain time complexity $O(\log(n))$, we'd want to:

1. Show that for an AVL tree, $h(T) = \theta(\log(n))$ (If there's time left)

2. See how to correct violations in AVL property - using **rotations**

3. See how to $Delete$ and $Insert$, while maintaining the height field.

1.1 Rotations
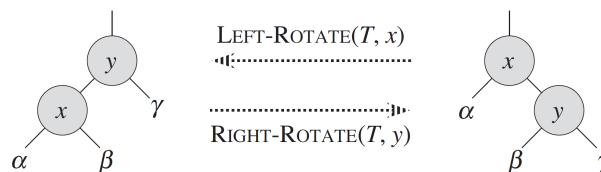
Rotations allow us to maintain the AVL property, in $O(1)$ time (you've discussed this in the lecture - changing subtree's roots). In this schematic representation of rotations, x, y are nodes and α, β, γ are subtrees. Note that the BST property is maintained!

The Balance factor allows us to identify if the AVL property was violated, and moreover - the exact values of the bad Balance factors will tell us which rotations to do to fix this:

Taken from last year's recitation:

| RL	LR	RR	LL	סוג ההפרה
				ציור
1) גורם האיזון בשורש תת העץ הוא 2-. 2) גורם האיזון בבן הימני של השורש הוא 1.	1) גורם האיזון בשורש תת העץ הוא 2 2) גורם האיזון בבן השמאלי של השורש הוא 1-	1) גורם האיזון בשורש תת העץ הוא 2-. 2) גורם האיזון בבן הימני של השורש הוא 1-.	1) גורם האיזון בשורש תת העץ הוא 2. 2) גורם האיזון בבן השמאלי של השורש הוא 1.	גילוי סוג ההפרה
1) רוטציית R על הבן הימני של השורש 2) רוטציית L על השורש	1) רוטציית L על הבן השמאלי של השורש 2) רוטציית R על השורש	1) רוטציית L על השורש	1) רוטציית R על השורש	הרוטציות המתאימות

Let's analyze one of these cases:

Example. Let's see why R rotation Fixes LL violation:
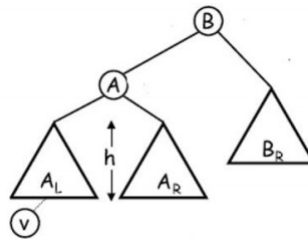
This is the general form of LL violations.

Analyze the heights of subtrees. Denote h the height of A_L before inserting v. So A_R's height has to also be h. If it's height was $h + 1$, the insertion wouldn't have created a violation in B (A's height would have stayed the same). If it was $h - 1$, the violation would've appeared first in A (not in B). Thus, A's height is $h + 1$. B_R's height is also h: If it was $h + 1$ or $h + 2$, no violation in B had occurred.

After the rotation, the tree looks like this: So all the nodes here maintain AVL property; why is it
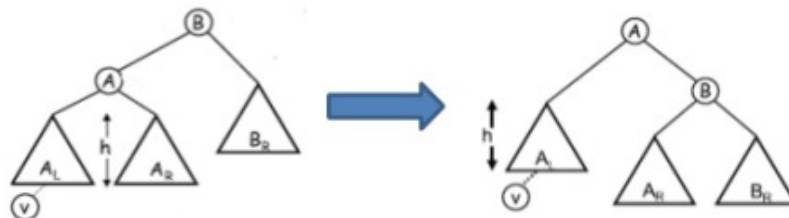
maintained in general?

Detect the violation in the following tree, and perform the necessary rotations:
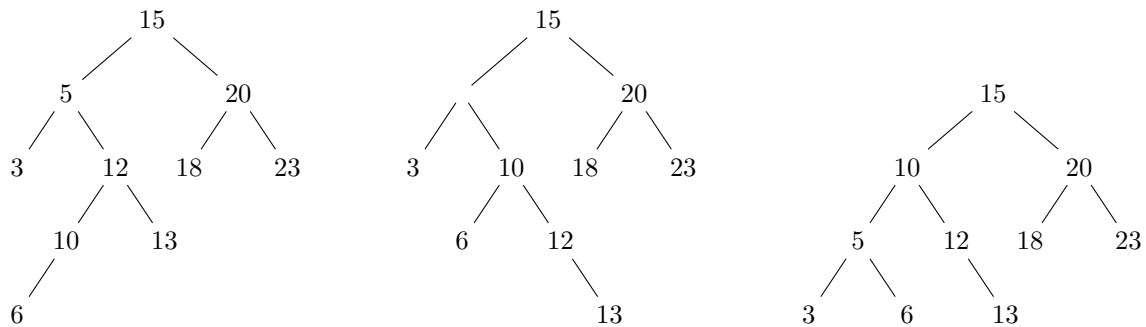

Figure 1: The first node in which a violation occurred is 5, this is an RL violation. Perform R rotation on the right child. Then perform L rotation on the root of the relevant subtree (5):

1.2 Delete, Insert.

The principles of the *Delete* and *Insert* operations are the same as in regular BST, but we will need to rebalance the tree in order to preserv AVL property. A single insertion or deletion may change the height difference of subtrees by at most 1, and might affect only the subtrees with roots along the path from r to the point of insertion/ deletion. More concretely - we will add a recursive operation of traversing the tree "back up" and checking violations. Had we found one - we'll fix it using rotations. Since rotations can be done in $O(1)$, the entire correction process will take $O(\log(n))$, so we maintain a good time complexity.

AVL Insert(r,x)

1 Call Insert (r, x) // (The standart insertion routine for BST)
2 Let p be the new node appended to the tree.
3 **while** $p.parent \neq \emptyset$ **do**
4   Check in which of the four states we are.
5   Apply the right rotation.
6   Update the height of each toched vertex
7      $p \leftarrow p.parent$

1.3 AVL tree's height

Let n_h be the minimal number of nodes in an AVL tree of height h.

Theorem. n_h is strictly increasing in h.

Proof. Exercise.

Theorem. $n_h = n_{h-1} + n_{h-2} + 1$.

Proof. For an AVL tree of height 0, $n_0 = 1$, and of height 1, $n_1 = 2$ (by checking directly).
Let's look at a minimal AVL tree of height h. By the AVL property, one of it's subtrees is of height $h - 1$ (WLOG - the left subtree) and by minimality, it's left subtree has to have n_{h-1} nodes. T's right subtree thus has to be of height $h - 2$: It can't be of height $h - 1$: $n_{h-1} > n_{h-2}$ by previous theorem, and if the

right subtree is of height $h-1$ - we could switch it with an AVL tree of height $h-2$, with less nodes - so less nodes in T, contradicting minimality. So the right subtree has n_{h-2} nodes (once again, by minimality), and thus the whole tree has $n_h = n_{h-1} + n_{h-2} + 1$ (added the root) nodes.

Corollary. $n_h > 2n_{h-2} + 1$

Corollary. $h = O(\log(n))$

Proof. Assume k is even (why can we assume that?). It can be shown by induction that:

$$n_h > 2n_{h-2} + 1 > 2(2n_{h-4} + 1) + 1 = 4n_{h-4} + (1+2)\ldots > 2^{\frac{h}{2}} + \sum_{i=0}^{\frac{h}{2}-1} 2^i = \sum_{i=1}^{\frac{h}{2}} 2^i = \frac{2^{\frac{h}{2}} - 1}{2-1} = 2^{\frac{h}{2}} - 1$$

So $n_h \geq 2^{\frac{h}{2}} - 1$, thus

$$h \leq 2\log(n_h + 1)$$

and for generall AVL tree with n nodes and height h:

$$h \leq 2\log(n_h + 1) \leq 2\log(n+1) = O(\log(n))$$

remark In fact, one can show that $n_h > F_h$ and F_h is the h'th Fibonacci number. Recall that $F_h = C(\varphi^h - (\psi)^h)$, and this gives a tighter bound on n_h.