CS120: Intro. to Algorithms and their Limitations Prof. Salil Vadhan

Lecture 6: Balanced BSTs, the RAM Model

Harvard SEAS - Fall 2021 *Sept. 21, 2021*

1 Announcements

- Pset deadlines are now 5pm
- Salil OH Thursday after c.ass
- Sanket OH this evening

Recommended Reading:

- Roughgarden II Section 11.4
- CLRS Exercise 13–3

Problem Set 1 Recurring Feedback:

- Median time has increased to the 12–16 hour range, with 22% of respondents spending 20+ hours. We don't want that! While we will continue to make adjustments, if you find yourself spending that long, please stop and use a revision opportunity.
- The time spent on theory problems in ps1 was much more reasonable than on ps0, but the experimental portion took a lot of time. Again a miscalibration with your prior experience - sorry!
- *Hopefully* ps2 and ps3 are actually more manageable...
- Students report lecture pace is good, but lectures are too disconnected from psets.
  - PS0 and PS1 were meant to be drawing on skills developed prior to CS120, which we miscalibrated. Hopefully the connection is clearer with PS2.
  - Lecture is not effective for problems-solving practice. I will try to do more examples and model more proofs, but much of the purpose of lecture is to give you the bigger conceptual framework in which the pset problems fit. (The active learning exercises are a good model of rigor: your proof explanation is rigorous enough if a receiver could fill in the details as needed.)
  - Section is meant as a bridge between lectures and problem sets. They are highly recommended, especially if you are finding gaps!
- Expectations on problems not clear enough. Noted and agreed. We are paying more attention to this!

- Finding pset clarifications on Ed is difficult. Solution: for every pset, we will have a main Ed post and thread, which we will update with important clarifications, using email notifications. (So you don't have to scroll through all student questions.)
- More starter code and test data. Will do!

Other announcements:

- Pset deadlines moved to *Wednesday 5pm*. Wed eve is for prepping for section (by reading next pset) and for any Thursday active learning exercises.
- Revised ps2 (clarifying problem 1) posted on Saturday.

2 Balanced Binary Search Trees

So far we have seen that a variety of operations (search, insertion, deletion, min, max, predecessor, successor) can be performed on Binary Search trees in time $O(h)$, where h is the height of the tree. Thus, we are now motivated to figure out how we can ensure that our binary search trees remain shallow (e.g. of height $O(\log n)$ where n is the number of items currently in the dataset). While doing so, we need to be sure that we retain the binary-search-tree property (the ordering of keys between a node and its left-descendants and its right-descendants). The approximate balancing operation in Problem Set 0 does *not* retain the binary-search-tree property (but it is useful in other applications that are beyond the scope of this course).

There are several different approaches for retaining balance in binary search trees.

Definition 2.1 (AVL Trees)**.** An *AVL Tree* is a binary search tree in which:

- Every node has an additional attribute containing its *height* in the tree (length of the longest path from the node to a leaf). ("data structure augmentation")
- Every pair of siblings in the tree have heights differing by at most 1 (where we define the height of the an empty subtree to be -1). ("height-balanced")

Lemma 2.2. *Every AVL Tree with n nodes has height (=depth) at most $2\log_2 n$.*

Proof. Attempt 1: let $h(n) =$ max height of an AVL Tree on $\leq n$ nodes. Then we would try to write a recurrence of the form $h(n) \leq h(c \cdot n) + c'$ (e.g., for $c = 1/2$). But it is not clear what the relation between how to relate a tree of n nodes to trees of at most cn nodes — the height-balanced property doesn't seem to imply that both children a node have subtrees of size at most cn.

However, the definition does allow us to reason about the height. Let us invert the problem and think of n as a function of h instead of h as a function of n (we turn the problem around).

Attempt 2: let $n(h) =$ min number of nodes in an AVL tree of height $\geq h$. Then, by the second AVL property, for $h \geq 2$, $n(h) \geq n(h-1) + n(h-2) + 1$. If we drop some constants, then

$$n(h) \geq n(h-1) + n(h-2) + 1 \geq 2n(h-2).$$

Now we are at a good place for unenrolling the expression:

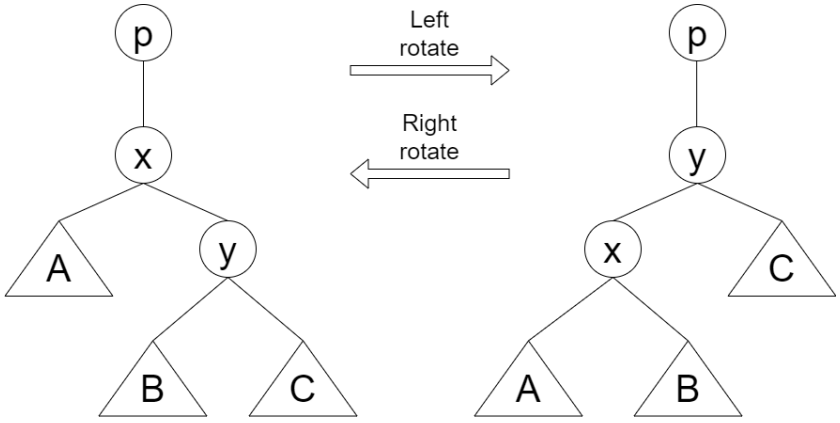
$$\begin{aligned} n(h) &\geq 2n(h-2) \\ &\geq 4n(h-4) \\ &\geq 8n(h-6) \\ &\vdots \\ &\geq n^{n/2}n(0) \\ &= 2^{h/2}. \end{aligned}$$

(The above is for the case when h is even; a similar analysis can be done when h is odd.)

□

So we can apply all of the aforementioned operations on an AVL Tree in time $O(\log n)$. But an insertion or deletion can ruin the AVL property. To fix this, we need additional operations that allow us to move nodes around while preserving the binary search tree property. One important example is a *rotation*.

We first investigate **rotations on BSTs.**



On the left: all keys in A are $\leq K_x$, all keys in B are $\in [K_x, K_y]$, and all keys in C are $\geq K_y \implies$ BST property is preserved in the right.

From left to right we have the operation `Left.rotate(x)`. From right to left we have the operation `Right.rotate(y)`.

How much time does a rotation take?

Rotations only take $O(1)$ time! One only needs to change a constant number of pointers – there is no traversing or other expensive operations.

Theorem 2.3. *We can insert a new item-key pair into an AVL Tree while preserving the AVL Tree property in time* $O(\log n)$

Proof. (sketch)

The algorithm runtime is as follows:

- Do an ordinary BST insert $\rightarrow$ adding a leaf L.

- Update heights working all the way from L to the roof.
- When we see that we violate the AVL tree property we use rotations to fix it.

In total, this requires time $O(\text{height}) = O(\log n)$.

□