CS1200: Intro. to Algorithms and their Limitations Prof. Anurag Anshu

Lecture 6: Binary Search Trees

Harvard SEAS - Spring 2026 *2026-02-10*

1 Announcements

- SRE2 today.
- Anurag OH after class.
- No class next Monday (President's day)

2 Recommended Reading

- Hesterberg–Vadhan, Chapter 4.4 - 4.6
- CLRS Sec 12.0-12.3

3 Insertion in a BST

We concluded the last lecture with the following table:

Data structure runtimes		
Queries	Sorted Array	Binary Search Tree
`search`	$O(\log n)$	$O(h)$
`predecessor`	$O(\log n)$	$O(h)$
`successor`	$O(\log n)$	$O(h)$
`min`	$O(1)$	$O(h)$
`max`	$O(1)$	$O(h)$
`rank`	$O(\log n)$	$O(n)$ [$O(h)$ using augmentation]
`insert`	$O(n)$	$O(h)$
`delete`	$O(n)$	$O(h)$

Table 1: List of query runtimes for sorted arrays and binary search trees.

Theorem 3.1. *Given a binary search tree of height h, all of the* Dynamic Predecessors & Successors *operations (queries and updates), as well as* `min` *and* `max` *queries, can be performed in time $O(h)$.*

Proof. • **Insertions:**

```
   Insert(T,(K,V)):
 0 if T = ∅ then
 1 |
 2 else
 3 |   Let v be the root of T, T_L its left-subtree, and T_R its right-subtree;
 4 |   if K ≤ K_v then
 5 |   |
 6 |   |
 7 |   |
 8 |   else
 9 |   |
10 |   |
11 |   |
```

Algorithm 3.1: `Insert()`

Runtime:

Proof of Correctness:

- **Search:**

  The algorithm is as follows:

- **Minimum (and Maximum):**
  The algorithm is to move to the left child until we reach a vertex with no left child, and then output the key at that vertex.

- **Predecessor (and Successor):**
  The algorithm for next-smaller follows along related lines to search algorithm:

- **Deletions:** The algorithm for deletion is the subject of today's Sender–Receiver Exercise.

□

4 Balanced Binary Search Trees

So far we have seen that a variety of operations 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 BST property.

There are several different approaches for retaining balance in binary search trees. We'll focus on the data structure produced by one such approach that is relatively easy to describe, called AVL trees, after mathematicians named Adelson-Velsky and Landis.

Definition 4.1 (AVL Trees)**.** An *AVL Tree* is a binary search tree that is:

1.

2.

Lemma 4.2. *Every nonempty height-balanced tree with n vertices has height at most $2\log_2 n$.*

5 Computational Models

So far, our conception of an algorithm has been informal: "a well-defined procedure for transforming inputs to outputs" whose runtime is measured as the number of "basic operations" performed on a given input. This is unsatisfactory: how can we identify the fastest algorithm to solve a given problem if we don't have agreement on what counts as an algorithm or as a basic operation?

To address this, we need to specify a *computational model*. A computational model is any precise way of describing computations. In approximate order from "far from physical" to "close to physical," some examples of computational models are:

0. Natural language (e.g. 'calculate the prime factorization of the following number'):
1. Declarative and functional programming languages:
2. Imperative high-level programming languages:
3. "Close to the metal" imperative programming languages:
4. Architecture-level models:
5. Hardware models:

Translation from higher-level to lower-level models:
Also need a complexity measure to model "time" (or other resources).