CSE 332: Data Abstractions

Outline

1. Introducing AVL Trees
2. Tree Representation in Code
3. How Does an AVL Tree Work?
4. Why Does an AVL Tree Work?
5. AVL Tree Examples

AVL Balance Condition!

Left and right subtrees recursively have heights differing by at most one.

Definition (balance)

\[ \text{balance}(n) = \text{abs}(\text{height}(n\text{.left}) - \text{height}(n\text{.right})) \]

Definition (AVL Balance Property)

An AVL tree is balanced when:

\[ \text{For every node } n, \text{ balance}(n) \leq 1 \]

- This ensures a small depth
- It's relatively easy to maintain

AVL Trees

Structure Property:
0, 1, or 2 children

BST Property:
Keys in Left Subtree are smaller
Keys in Right Subtree are larger

AVL Balance Property:
Left and Right subtrees have heights that differ by at most one.

That is, all AVL Trees are BSTs, but the reverse is not true.

AVL Trees rule out unbalanced BSTs.

Tree Representation in Code

This Definition Leads to Redundant Code

```java
boolean find(Node current, int data) {
    if (current == null) {
        return false;
    }
    else if (current.data == data) {
        return true;
    }
    else if (current.data < data) {
        return find(current.left, data);
    }
    else {
        return find(current.right, data);
    }
}
```

But that's what we've been writing! Why is it ugly?

- It's redundant
- The left and right cases are the same, why write them twice?
- It's not idiomatic (e.g., the right abstraction would allow us to write the two cases found vs. not found)
This course is about making the right data abstractions. This is a perfect example of where we could improve.

Keep an array of children!

The BST Worst Case

Worst Case

When we insert 3, we violate the AVL Balance condition. What to do?

FIXING The Worst Case

This is just the same rotation in the other direction!

AVL Rotation

This “fix” is called a rotation. We’re “rotating” the child node “up”:

This is the only fundamental of AVL Trees!

You can either look at this as “the only way to correctly rearrange the subtrees” or it’s helpful to think of it as gravity.

More Complicated Now...
AVL Rotation: The Other Way

10

Rotation

The Code

```c
void rotate(Node current) {
    Node child = current.left;
    current.left = child.right;
    child.right = current;
    child.height = child.updateHeight();
    current.height = current.updateHeight();
    current = child;
}
```

AVL Rotations... Are We Done?

11

We Want...

Cases We've Handled

Cases To Handle

Another Case

12

Second Case

When we insert 2, we violate the AVL Balance condition. What to do?

There's only one tree with the BST Property and the Balance Property:

FIXING The Second Case

It Doesn't Look Like a Single Rotation Will Do...

13

Double Rotation

First, we rotate b.

Now, we're back to the line case.

And now it's balanced!

And The Code...

14

Double Rotation

```c
void doubleRotation(Node current) {
    rotation(current.right, RIGHT);
    rotation(current, LEFT);
}
```

Putting Together the AVL Operations

15

AVL Operations

- `find(x)` is identical to BST `find`
- `insert(x)` by (1) doing a BST `insert`, and (2) fixing the tree with either a rotation or a double rotation
- `delete(x)` by either a similar method to `insert`–or doing lazy delete

AVL Fields

- We've seen that the code is very redundant if we use `left` and `right` fields; so, we should use a `children` array
- We've seen quick access to `height` is very important; so, it should be a field

Okay, so does it work?
Does an AVL Tree Work?

We must guarantee that the AVL property gives us a small enough tree.

Our approach: Find a big lower bound on the number of nodes necessary to make a tree with height $h$.

**What is the smallest number of nodes to get a height $h$ AVL Tree?**

For $h = 0$

For $h = 1$

For $h = 2$

This is not an AVL tree!

**Pros of AVL trees**

- All operations logarithmic worst-case because trees are always balanced.
- Height balancing adds no more than a constant factor to the speed of insert and delete.

**Cons of AVL trees**

- Difficult to program & debug.
- More space for height field.
- Asymptotically faster but rebalancing takes a little time.
- Most large searches are done in database-like systems on disk and use other structures (e.g., B-trees, our next data structure).

So, efficiency?

So, since $n \geq \phi^h - 1$, taking log of both sides gives us:

$$\log(n) > \log(\phi^h - 1) = \log(\phi^h) - \log(\phi)$$

So, $h \in \Theta(\log n)$.

- Worst-case complexity of find: $O(\log n)$
- Worst-case complexity of insert: $O(\log n)$
  - Tree starts balanced
  - A rotation is $O(1)$ and there’s an $O(\log n)$ path to root
  - (Same complexity even without one-rotation-is-enough fact)
  - Tree ends balanced
- Worst-case complexity of buildTree: $O(n \log n)$
- Worst-case complexity of delete: (requires more rotations) $O(\log n)$
- Worst-case complexity of lazyDelete: $O(1)$

Proving the Closed Form

In this case, we see that $f(h)$ pretty quickly converges to $\phi (1.618 \ldots)$. Before trying to prove this closed form, we should look at a few examples:

- $f(0) = 1$ vs. $\phi^0 = 1$
- $f(1) = 2$ vs. $\phi^1 = \phi$

We want to show that $f(h)$ gets some closed form, but looking at the first few cases, $f(1.9$. So, we’ll prove $f(h) > \phi^{h-1}$ instead.

**Induction Proof**

- Base Cases: Note that $f(0) = 1 > 1 - 1 = 0$ and $f(1) = 2 > \phi - 1 = 0.618$
- Induction Hypothesis: Suppose that $f(h) > \phi^{h-1}$ for all $0 \leq h \leq k$ for some $k \geq 1$.

**Induction Step:**

$$f(n+1) \geq f(n) + f(n-1) + 1$$

By IH:

$$= \phi^{n-1} + (\phi^{n-1} - 1) + 1$$

$$= \phi^{n-1} + \phi^{n-2}$$

In the step labeled “by $\phi$”, we used the property $\phi^2 = \phi + 1$. 

Does an AVL Tree Work?
Some Examples 22

Example (Insert a, b, c, d into an AVL Tree)

- \( \text{insert}(a) \)
- \( \text{rotate}(a) \)
- \( \text{insert}(b) \)
- \( \text{rotate}(a) \)
- \( \text{insert}(c) \)
- \( \text{rotate}(c) \)
- \( \text{rotate}(e) \)

Some Examples 23

Example (Which Rotation?)

- Which insertions would cause a single rotation?

Some Examples 24

Example (Which Rotation?)

- Which insertions would cause a double rotation?

Some Examples 25

Example (Which Rotation?)

- Which insertions would cause no rotation?

Some Examples 26

Example (Insert 3, 33, 18, 32)

- \( \text{insert}(3) \)
- \( \text{rotate}(15) \)
- \( \text{insert}(33) \)
- \( \text{insert}(18) \)
Some Examples

Example (Insert 3, 33, 18, 32)

10
5
2
3
9
20
15
12
17
18
30
33

Insert(32)

rotate(33)

Example (Insert 3, 33, 18, 32)

10
5
2
3
9
20
15
12
17
18
32
30
33

rotate(30)