A self-balancing BST

Random binary search trees are *very likely* to be balanced
Self-balancing trees are *guaranteed* to be balanced
Balanced search tree survey
Red–black trees (tomorrow)

**Main idea:** Every node has an extra bit marking it “red” or “black”

**Local invariant:** No red node has a red parent

**Global invariant:** Equal number of black nodes from root to every leaf
2-3 trees

Main idea: 2-nodes have one element and two children; 3-nodes have two elements and three children

Local invariant: All subtrees of a node have the same height

Global invariant: Every leaf is at the same depth
2-4 trees

**Main idea:** Like 2-3 trees, but also has 4-nodes with three elements and four children

**Local invariant:** All subtrees of a node have the same height

**Global invariant:** Every leaf is at the same depth
B-trees

Main idea: Generalization of 2–4 trees to 2–k trees

Local invariant: Like 2–4 trees, but allow some number of missing subtrees

Global invariant: Every leaf is at the same depth
Splay trees

Main idea: Cache recently accessed elements near the root of the tree

Local invariant: *Complicated; required amortized analysis*

Global invariant: Paths are *very likely* to be $\mathcal{O}(\log n)$
AVL trees

**Main idea:** Maintain a *balance factor* giving the difference between each node’s subtrees’ heights

**Local invariant:** Balance factor between -1 and 1, maintained via rotations

**Global invariant:** Tree is approximately height-balanced

(AVL stands for Georgy Adelson-Velsky and Evgenii Landis)
AVL trees
Example of an AVL tree
Local invariant maintains global property

- Balance factors are maintained locally
- Never recompute them from scratch
- Yet the whole tree stays reasonably balanced
AVL insertion

- First do a normal leaf insertion
- Track balance factors on the way back up to the root
- Adjust with rotations as necessary
AVL insertion example

Let’s insert H:

```
AVL insertion example

Let’s insert H:

<table>
<thead>
<tr>
<th>J</th>
<th>+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>+1</td>
</tr>
<tr>
<td>V</td>
<td>-1</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
</tr>
<tr>
<td>Q</td>
<td>0</td>
</tr>
<tr>
<td>U</td>
<td>0</td>
</tr>
<tr>
<td>N</td>
<td>0</td>
</tr>
<tr>
<td>L</td>
<td>+1</td>
</tr>
<tr>
<td>F</td>
<td>-1</td>
</tr>
<tr>
<td>D</td>
<td>-1</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
</tr>
<tr>
<td>L</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>0</td>
</tr>
<tr>
<td>J</td>
<td>0</td>
</tr>
</tbody>
</table>
```

The tree structure is shown with nodes labeled and balance factors indicated.
AVL insertion example

Let’s insert H:
AVL insertion example

Let’s insert H:
AVL insertion example

Let’s insert H:
Another AVL insertion example

Let’s insert B:

```
  J
 /   \
F     P
 / \
D   G  L
 |
C   
```

16
Another AVL insertion example

Let’s insert B:
Another AVL insertion example

Let’s insert B:

```
+1
-1
0
-1
0
+1
+1
0
-1
0
0
0
0
```
Another AVL insertion example

Let’s insert B:
Another AVL insertion example

Let’s insert B:
Another AVL insertion example

Let’s insert B:

```
B 0 0
C 0 0
D 0 0
F -1 0
G 0 0
J +1 0
P +1 0
L +1 0
N 0 0
S 0 0
Q 0 0
U 0 0
V -1 0
X 0 0
```
Maintaining the AVL property

Suppose we have an AVL tree:

(Convention: triangles represent equal-height subtrees.)
Maintaining the AVL property

Suppose we have an AVL tree:

![AVL Tree Diagram]

(Convention: triangles represent equal-height subtrees.)

Right now the balance factor is 0. So if we insert into A or C and that subtree grows in height, it becomes -1 or 1.
Right now the balance factor at B is +1.

Suppose we insert into A. What happens to B’s balance factor?
Maintaining the AVL property

Right now the balance factor at B is +1.

Suppose we insert into A. What happens to B’s balance factor?

- If no change in A’s height then no change in B’s balance
- If A’s height grows then B’s balance factor goes to 0
Maintaining the AVL property

Right now the balance factor at B is +1.

Suppose we insert into C. What happens to B’s balance factor?
Maintaining the AVL property

Right now the balance factor at B is +1.

Suppose we insert into C. What happens to B’s balance factor?

- If no height change then B’s balance doesn’t change
- If C grows then B’s balance factor becomes +2
Maintaining the AVL property

Right now the balance factor at B is +1.

Suppose we insert into C. What happens to B’s balance factor?

- If no height change then B’s balance doesn’t change
- If C grows then B’s balance factor becomes +2—not okay!
Maintaining the AVL property

Right now the balance factor at B is +1.

Likewise, suppose we insert into E. What happens to B’s balance factor?

- If no height change then B’s balance doesn’t change
- If E grows then B’s balance factor becomes +2—not okay!
The right-right case

If the height of the right-right subtree (formerly E) increases, we get a situation like this:
The right-right case

If the height of the right-right subtree (formerly E) increases, we get a situation like this:
The right-left case

If the height of the right-left subtree (formerly C) increases, we get a situation like this:
The right-left case

If the height of the right-left subtree (formerly C) increases, we get a situation like this:

Before:

```
B
 +2
 D
   -1
 0
A
```

After, right-right case:

```
B
 +2
 D
   +1
 0
A
```

But this is now the right-right case, which we know how to handle!
The right-left case

If the height of the right-left subtree (formerly C) increases, we get a situation like this:

But this is now the right-right case, which we know how to handle!
Maintaining the AVL property

- We’ve seen the right-right and right-left cases
- The left-left and left-right cases are symmetrical
- Deletion is like ordinary BST deletion, with the same rebalancing cases
Next time: red–black trees