Red-Black Trees
Red-Black Trees

- Red-Black as 2-3-4
- Red-Black as BST
- Bottom-Up Insertion
- Top-Down Insertion
Red-Black as 2-3-4

- Red-black trees simulate 2-3-4 trees using a BST.

```
15, 20, 25
```

```
20

15

25
```
Red-Black as 2-3-4

- A node with a single value is replaced by a “widget” which is a single node.

In Computer Science, when we replace one thing with another that simulates it, we call the replacement a “widget.”
Red-Black as 2-3-4

- Nodes with two values can be encoded as either left-leaning widgets...
Red-Black as 2-3-4

...or right-leaning widgets.
Red-Black as 2-3-4

- In all cases, the “root” of the 2-3-4 node is represented by a black node.
Red-Black as 2-3-4

- Notice that the # of children always works out properly. (As do the ranges of each child.)
Red-Black as 2-3-4

- Since the children are other 2-3-4 nodes, each child is black – the root of a new widget.
Red-Black as 2-3-4

- It's impossible for any red node to touch another red node.
Red-Black as 2-3-4

- Every descent through the tree contains the same number of black nodes.
Red-Black as 2-3-4

- This is because there is a 1-to-1 mapping between black nodes and 2-3-4 nodes.
Red-Black as 2-3-4

- Black leaves represent single-key leaves.
- Red leaves are keys in larger leaves.
Red-Black as 2-3-4

- Sometimes, we think of “virtual leaves.”
- If we do this, then every node has 2 children.
Red-Black as 2-3-4

- Each virtual leaf has the same “black height.”
- **Black height:** # of black nodes to the root.
Red-Black Trees

- Red-Black as 2-3-4
- Red-Black as BST
- Bottom-Up Insertion
- Top-Down Insertion
Red-Black as BST

- Typically: Think of red-black as BST, not 2-3-4
  - Add some simple rules.

- Because of the rules, it works like a B-Tree
  - Same distance to all leaves (black height)
  - No nodes with a single child
  - $h = O(\log n)$
The Rules of Red-Black Trees

- A “red-black tree” is a BST where we mark each node as “red” or “black”

**Rules**

1) Root is always black
2) All leaves are black, but store nothing (virtual leaves)
3) All leaves same “black height”
4) No adjacent reds
The Rules of Red-Black Trees

Rule 1:
Root is always black.
The Rules of Red-Black Trees

Rule 2:
Leaves are always black, but store nothing.
The Rules of Red-Black Trees

We'll often omit the "virtual leaves" from the picture.
Rule 3:
All leaves are always at the same “black height.”
The Rules of Red-Black Trees

This tree is still valid; it's OK to have black nodes that touch other black nodes.
The Rules of Red-Black Trees

**Rule 4:**
No adjacent reds.

**ILLEGAL!**
Red-Black Trees

- Red-Black as 2-3-4
- Red-Black as a special BST
- **Bottom-Up Insertion**
- Top-Down Insertion
Bottom-Up Insertion in Red-Black

- Insert as normal, mark new leaf red

- If parent of leaf is also red, then run a fixup routine toward the root
  - Simulates shifting keys around and/or splitting nodes in the 2-3-4 tree.
Insertion

Let's add 50.
Some Simple Insertion Examples

Add a new leaf, and mark it red.

What sort of action in a 2-3-4 tree does this simulate?
Insertion

These top 3 nodes model the root of the 2-3-4 tree.
The single black node (with no red children) represents a node with only one key.
Insertion

Adding a red child to 60 models adding a second key to the 2-3-4 node.
Some Simple Insertion Examples

This is how the tree looks like after the insertion.

60 was black, so no fixup routine is required.

This is because the 2-3-4 node would not split.
Some Simple Insertion Examples

We have inserted 45.

Now we have a red next to a red. Let's see what this represents.
Some Simple Insertion Examples

This is not a big deal in 2-3-4. There are only 3 keys. But we need to fix the red-black tree.

How?
Some Simple Insertion Examples

We would like to end up with the 50 as the black node, and with 45 and 60 as its children.
Some Simple Insertion Examples

We can do this with a rotation (followed by recoloring).
Some Simple Insertion Examples

Note that we had to RECOLOR the nodes after the rotation.
Some Simple Insertion Examples

There's a critical detail that we overlooked.

What if 60 already had a right child?
Some Simple Insertion Examples

This is where the “virtual leaves” can be helpful.

60 did have a right child. It was a virtual leaf – and it was black.

There were lots of virtual leaves, in fact.
This rotation is legal, so long as the sibling is black.

The sibling might be a virtual leaf – or an ordinary black node!

It doesn't matter.
Some Simple Insertion Examples

We have inserted 42.

This causes a problem, because 60 (the sibling of 45) is red.
Some Simple Insertion Examples

A rotation in this situation doesn't make things any better!

This is because we need to split the 2-3-4 node.
Some Simple Insertion Examples

The correct solution for this situation is to “bubble up” the red at 50.
Some Simple Insertion Examples

This keeps the black-height consistent.

But it creates a new red-next-to-red problem.

Can we see what this really represents in the 2-3-4 tree?
Here's the node (with 4 keys) right after insertion.
“Bubbling the red up” is equivalent to splitting the node. 50 gets pushed up, into the parent.
“Bubbling the red up” is equivalent to splitting the node. 50 gets pushed up, into the parent.
Some Simple Insertion Examples

Here's what it looks like in the red-black tree.

50 and 70 are both red. We need to fix this.

We use the same sort of algorithms as we did at the lower level.
Some Simple Insertion Examples

Again, the fix is to bubble the red up.

This is splitting the root 2-3-4 node.
Some Simple Insertion Examples

When we split the root node (making it red), we have to change the root back to black.

This is "adding a layer" in the 2-3-4 tree.
Some Simple Insertion Examples

We're done!

Notice that all of the paths through the tree have the same # of black nodes.
Some Simple Insertion Examples

This is another view of the same tree, just arranged to show the black layers.
Let's see the whole process again, focusing on the 2-3-4 nodes.
We start by inserting 42, which makes the leaf node too large.
We split the 2-3-4 node (bubbling up the red) which pushes 50 into the root node.
Insertion

Now, the root node has too many keys.
Insertion

Bubble the red up, which is a split.
Recolor the root.

Insertion

5
10
12
18
25
15
20
42
45
50
60
70
80
42
40
15
70
50
45
60
42
Bottom-Up Insertion in Red-Black

- Create a new leaf, make it red

- While red next to red:
  - Do one of the fixup operations
  - Might create a red next to red **higher up**
    - Recurse toward the root

- If needed, set root to black
Fixing Red/Red Problems

Fixup Case 1: Red next to black

NOP!
Fixing Red/Red Problems

**Fixup Case 2:** Uncle is black (leaf?).

Rotate at grandparent, recolor.
Fixing Red/Red Problems

**Fixup Case 2b:** Case 2, but LR instead of LL.

Hint: What did you do in an AVL tree?
Fixing Red/Red Problems

**Fixup Case 3:** Uncle is red.

Bubble the red up, then recurse up.
Fixing Red/Red Problems

Symmetric Cases

Each of these is simply the reflection of a case we've already handled.
Red-Black Trees

- Red-Black as 2-3-4
- Red-Black as BST
- Bottom-Up Insertion
- Top-Down Insertion
Top-Down Insertion in Red-Black

• Recurse through the tree, making changes as required.
  – Split (push black down) when possible
  – Use rotations to fix local red/red problems

Loop Invariant:
• The current node is black – or the node's sibling is.
Top-Down Insertion

Let's insert 30.
The recursion begins at the root.
Top-Down Insertion

Let's insert 30.

Because the current node has two red children, we bubble the red up.
Let's insert 30.

When we do this at the root, we have to fix the root, marking it black again.
Top-Down Insertion

Let's insert 30.

Now that we've updated the tree, we recurse.
Let's insert 30.

Recurse again.

Now we must bubble red up again, and recurse after we're done.
Top-Down Insertion

Let's insert 30.

We've arrived at a leaf. This leaf is black, so we can add a red child without any worry.
Top-Down Insertion

Let's insert 30.

We've arrived at a leaf. This leaf is black, so we can add a red child without any worry.

Next, let's see what happens when we add below red node.
Top-Down Insertion

Let's insert 35.

This works down through the same path, until it reaches 30.

While 30 is red, its sibling (a virtual leaf) is black.
Top-Down Insertion

Let's insert 35.

This creates a red/red problem, but because 30's sibling is black, we can rotate to fix it.
Top-Down Insertion

Let's insert 35.

This creates a red/red problem, but because 30's sibling is black, we can rotate to fix it.
Why it Works

• Red/red problems arise in top down:
  - When creating a new leaf below a red node
    • Can rotate to fix
  - When bubbling red up (turns the current node red)

• Let's re-examine the red/red fixup cases (same as bottom-up)
Fixing Red/Red Problems

These 3 cases are all handled without pushing any more red up. They are fixed **locally**, without recursion.
Why it Works

- Red/red problems arise in top down:
  - When creating a new leaf below a red node
    - Can rotate to fix
  - **When pushing black down (turns the current node red)**

- Let's re-examine the red/red fixup cases (same as bottom-up)
Fixing Red/Red Problems

Single level recursion is possible.

We did not push up into 50 – it is red, but its sibling (70) is black.

However, we have to push up into 45.
Now we have a red/red problem. But this can be fixed with a rotation.
Fixing Red/Red Problems

The rotation can never recurse further, because it doesn't create a red node at the top.
Fixing Red/Red Problems

This case is impossible. We would have bubbled the red up into 60.

This is the only case which requires indefinite recursion (in bottom-up) – so indefinite recursion never happens in top-down insertion.
Red-Black Trees

- Red-Black as 2-3-4
- Red-Black as BST
- Bottom-Up Insertion
- Top-Down Insertion

Summary