Red-Black Trees

- Insertion
- Deletion
Beyond (2,4) Trees

What do we know about (2,4)Trees?

- Balanced
- $O(\log n)$ search time
- Different node structures

Can we get the (2,4) tree advantages in a binary tree format???

Welcome to the world of Red-Black Trees!!!
A **red-black tree** is a binary search tree with the following properties:

- edges are colored **red** or **black**
- **no two consecutive red edges on any root-leaf path**
- **same number of black edges on any root-leaf path** *(black height)*
- **edges connecting leaves are black**
(2,4) Tree Evolution

Note how (2,4) trees relate to red-black trees

Now we see red-black trees are just a way of representing 2-3-4 trees!
Red-Black Tree Properties

N  # of internal nodes
L  # leaves (= N + 1)
H  height
B  black height

Property 1:  \(2^B \leq N + 1 \leq 4^B\)

Property 2: \(\frac{1}{2} \log (N + 1) \leq B \leq \log (N + 1)\)

Property 3: \(\log (N + 1) \leq H \leq 2 \log (N + 1)\)

This implies that searches take time \(O(\log N)\)!
Insertion into Red-Black

1. Perform a standard search to find the leaf where the key should be added
2. Replace the leaf with an internal node with the new key
3. Color the incoming edge of the new node red
4. Add two new leaves, and color their incoming edges black
5. If the parent had an incoming red edge, we now have two consecutive red edges! We must reorganize tree to remove that violation. What must be done depends on the sibling of the parent.
Insertion - Plain and Simple

Let:
- \( n \) be the new node
- \( p \) be its parent
- \( g \) be its grandparent

Case 1: Incoming edge of \( p \) is black

No violation

STOP!

Pretty easy, huh?

Well... it gets messier...
Restructuring

Case 2: Incoming edge of $p$ is red, and its sibling is black

We call this a “rotation”
- No further work necessary
- Inorder remains unchanged
- Black depth is preserved for all leaves
- No more consecutive red edges!
- Corrects “malformed” 4-node in the associated (2,4) tree
More Rotations

(2,4) Trees
Promotion

Case 3: Incoming edge of $p$ is red and its sibling is also red

- We call this a “recoloring”
- The black depth remains unchanged for all the descendants of $g$
- This process will continue upward beyond $g$ if necessary: rename $g$ as $n$ and repeat.
- Splits 5-node of the associated (2,4) tree
Summary of Insertion

• If two red edges are present, we do either
  • a restructuring (with a simple or double rotation) and stop, or
  • a recoloring and continue

• A restructuring takes constant time and is performed at most once. It reorganizes an off-balanced section of the tree.

• Recolorings may continue up the tree and are executed $O(\log N)$ times.

• The time complexity of an insertion is $O(\log N)$. 
An Example

Start by inserting “REDSOX” into an empty tree

Now, let’s insert “CUBS”...
A Cool Example

(2,4) Trees
An Unbelievable Example

What should we do?

Oh No!

(2,4) Trees
Rotation
A Beautiful Example

What now?
Rotation

(2,4) Trees
A Super Example

Holy Consecutive Red Edges, Batman!

We could’ve placed it on either side
The SUN lab and Red-*Bat* trees are safe... ...for now!!!
Cut/Link Restructure Algorithm

- Remember the cut/link restructure algorithm from AVL tree lecture? We can use it to implement rotation.

- We use an inorder traversal to restructure the tree as before

- For example, below we have a subtree with two consecutive red edges.
But there is one more consideration in the case of a red-black tree: recoloring.

In this case, the root of the subtree should be the same color as the former root was, and both of its children should be colored red. This is the only recoloring case for Insertion.

For deletion, you will need to perform “color compensation” (you’ll hear about it in a minute) on the grandchildren.