Rec 08: Red-Black Trees

Audrey Yuan: ady8@cornell.edu

s/o Anissa (sec 207) for animations!

What's wrong with BSTs?

- standard way of defining a BST:

```
type ‘a tree =
| Leaf
| Node of ‘a * ‘a tree * ‘a tree
```

- BST invariant w/ the following statements
  - all nodes must be larger than any node in its left subtree
  - all nodes must be smaller than any node in its right subtree

What's wrong with BSTs?

Both are valid trees:

```
        5              0
       / \              \
      /   \              1
     /     \              \
    2       7              2
   / \     / \              \
  1   4   6   9              3
 /   /       / \              \
0   3       8   10            ...
```

If a client puts nodes in increasing order, then we have a linked list!

- what we want to be `O(log(n))` becomes `O(n)` time!

Red Black Tree Invariant

- We say a node can either be red or black
  - no adjacent red nodes along any path
  - number of black nodes from root to any leaf is the same
    - number of black nodes is called black height BH
  - convention: root is black
- Example:

```
        B
       / \
      /   \
     R     R
    / \   / \
   B   B B   B
```

Why??

worst case:

```
           B
          / \
         /   \
        B     R
       / \   / \
      B ... ... B
     / \       / \
    B  ...   ...  R
   /               \
...                 ...
```

relative length of two paths:

- longest path alternates between R/B
- shortest path has B nodes
- both have same number of B nodes

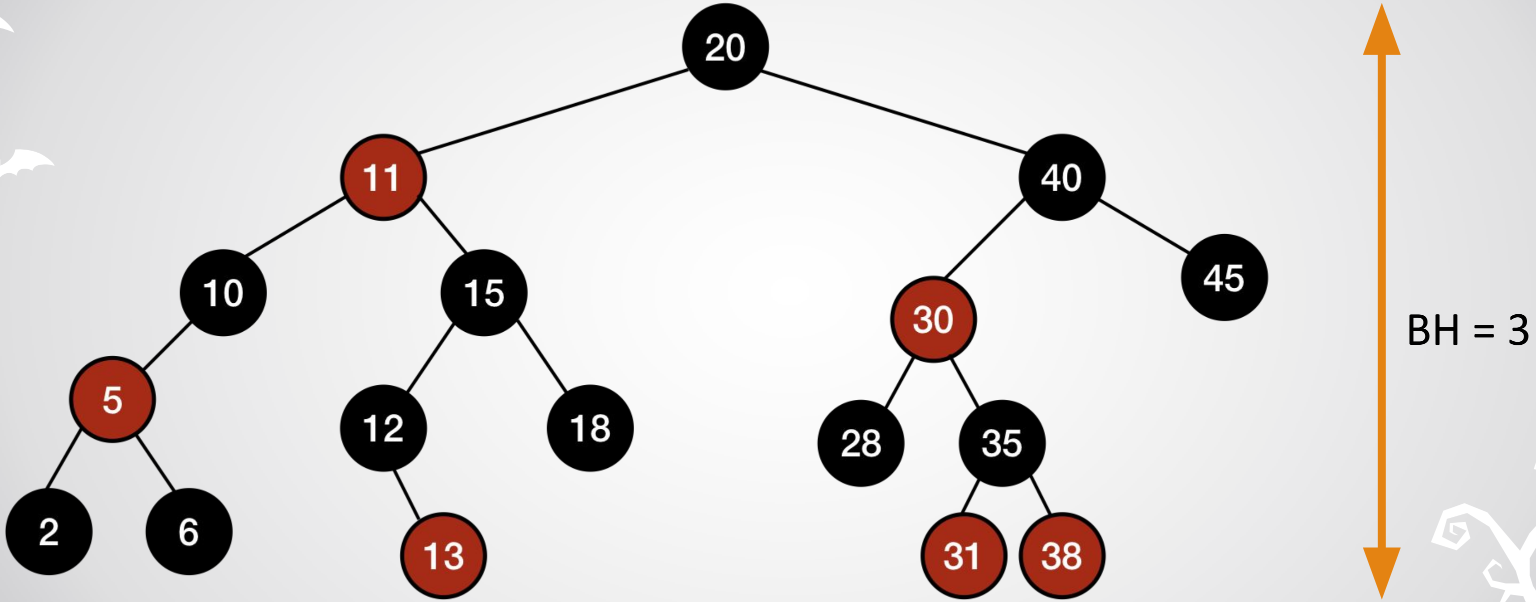
path length differs by factor of 2!

yay! `O(logn)` for operations

What are Red Black Trees?

- Balanced binary search tree
  - All nodes in the left subtree of a node are less than that node and all nodes in the right subtree are larger
  - If has n nodes, the height is O(logn)
- satisfies Red Black Tree Invariant

Red Black Tree Example



Note: a lot of times leaf nodes aren't actually drawn out

RB Trees in OCaml

Type Definition:

```
type color = Red | Black
type ‘a rb_tree =
| Leaf
| Node of color * ‘a * ‘a rb_tree * ‘a rb_tree
```

RB Trees in OCaml

Membership:

(same as BSTs!)

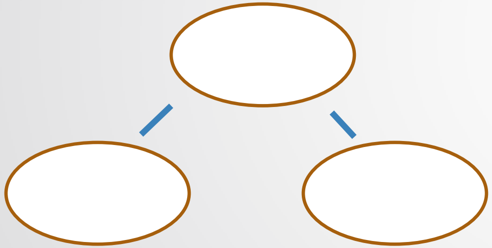
```
type rec mem x = function
   | Leaf -> false
   | Node (_, y, left, right) ->
       x = y || (x < y && mem x left) || (x > y && mem x right)
```

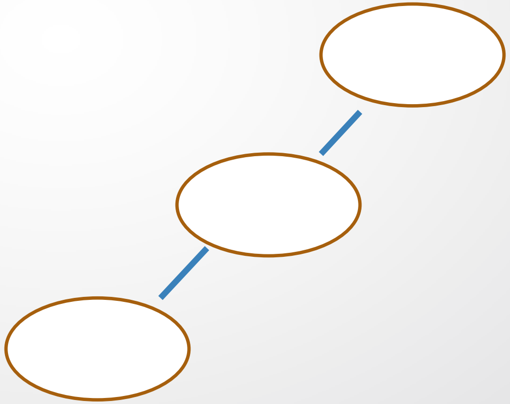
RB Trees Exercise

Exercise: given 3 nodes (not including leaves), what possible BST can you construct and how would you could you color them to satisfy the RB tree invariant?

RB Trees Exercise

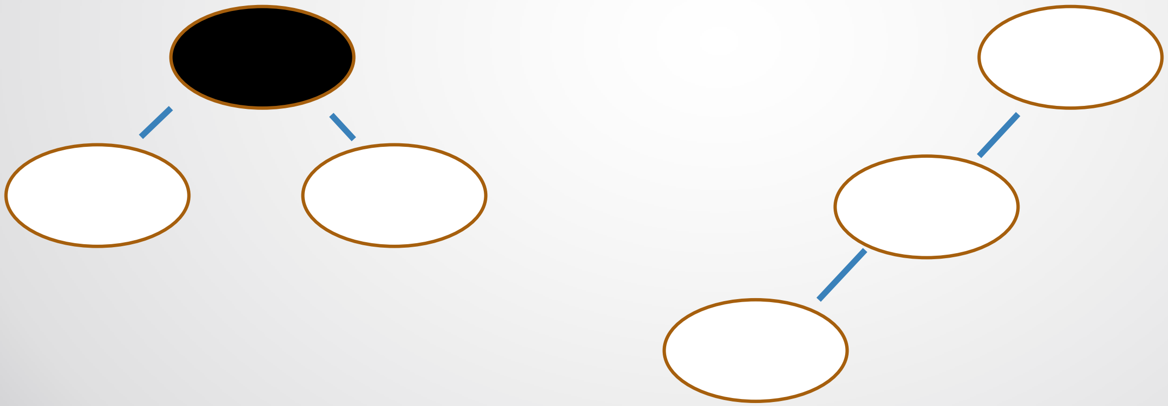
Exercise: given 3 nodes (not including leaves), what possible BST can you construct and how would you could you color them to satisfy the RB tree invariant?





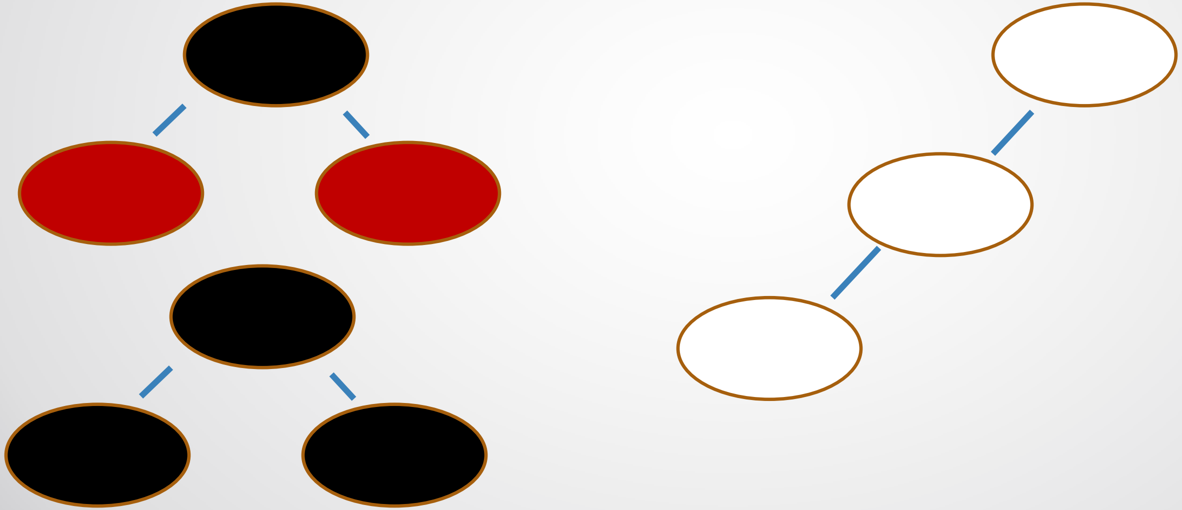
RB Trees Exercise

Exercise: given 3 nodes (not including leaves), what possible BST can you construct and how would you could you color them to satisfy the RB tree invariant?



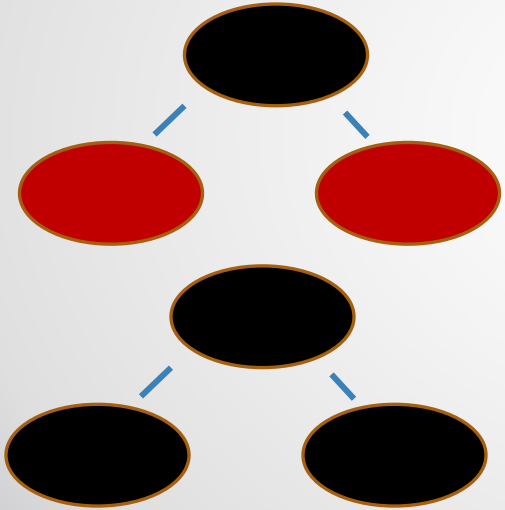
RB Trees Exercise

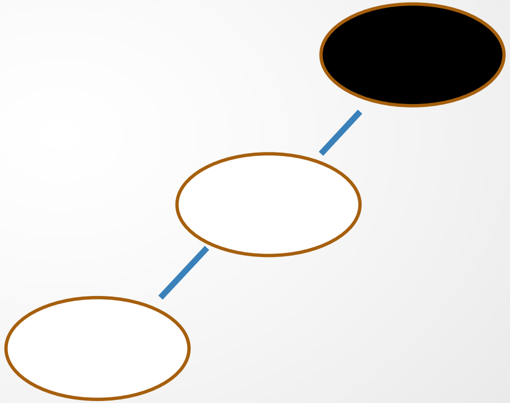
Exercise: given 3 nodes (not including leaves), what possible BST can you construct and how would you could you color them to satisfy the RB tree invariant?



RB Trees Exercise

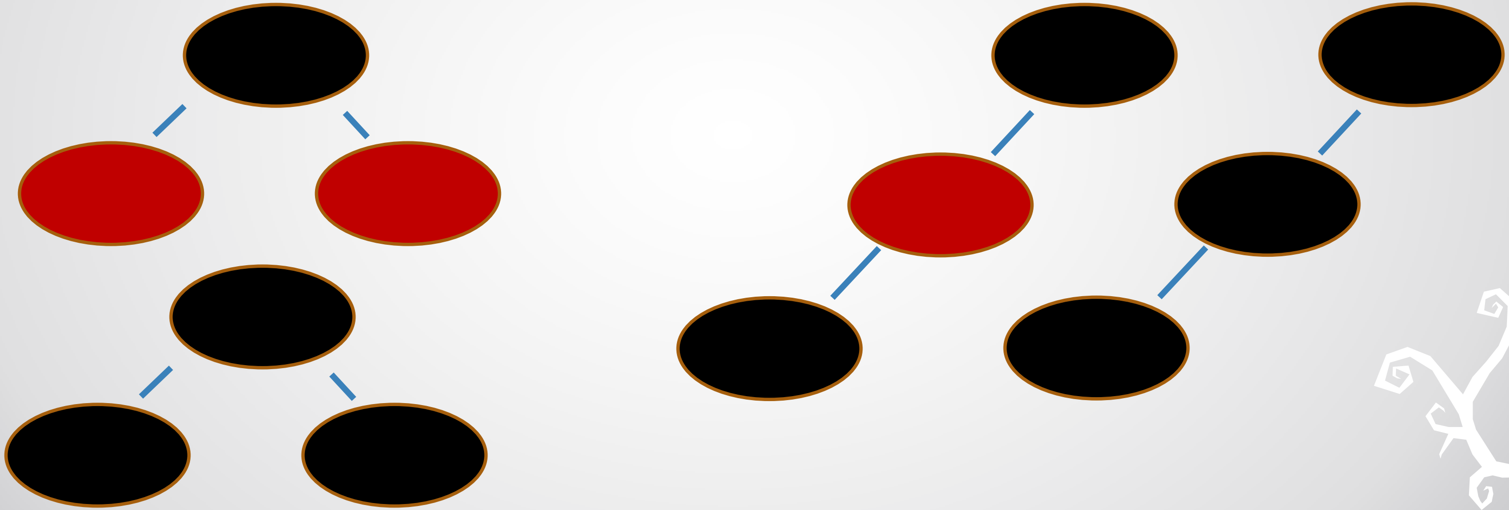
Exercise: given 3 nodes (not including leaves), what possible BST can you construct and how would you could you color them to satisfy the RB tree invariant?





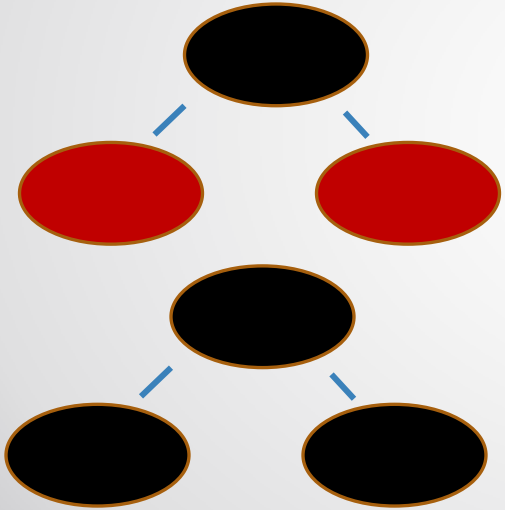
RB Trees Exercise

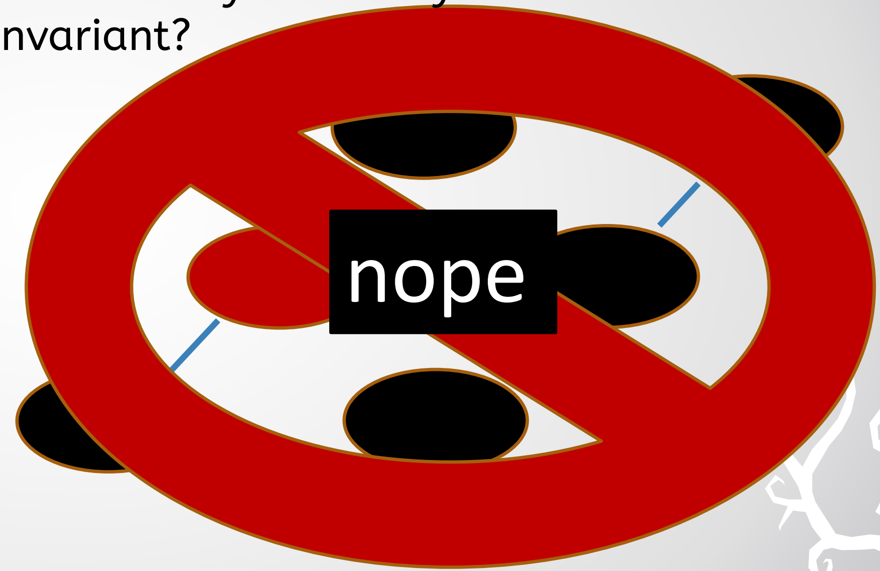
Exercise: given 3 nodes (not including leaves), what possible BST can you construct and how would you could you color them to satisfy the RB tree invariant?



RB Trees Exercise

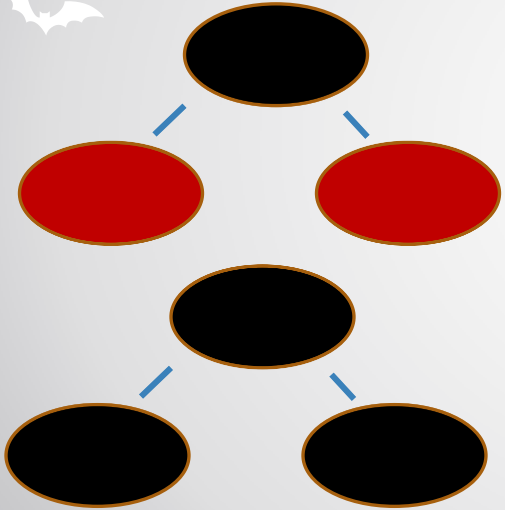
Exercise: given 3 nodes (not including leaves), what possible BST can you construct and how would you could you color them to satisfy the RB tree invariant?

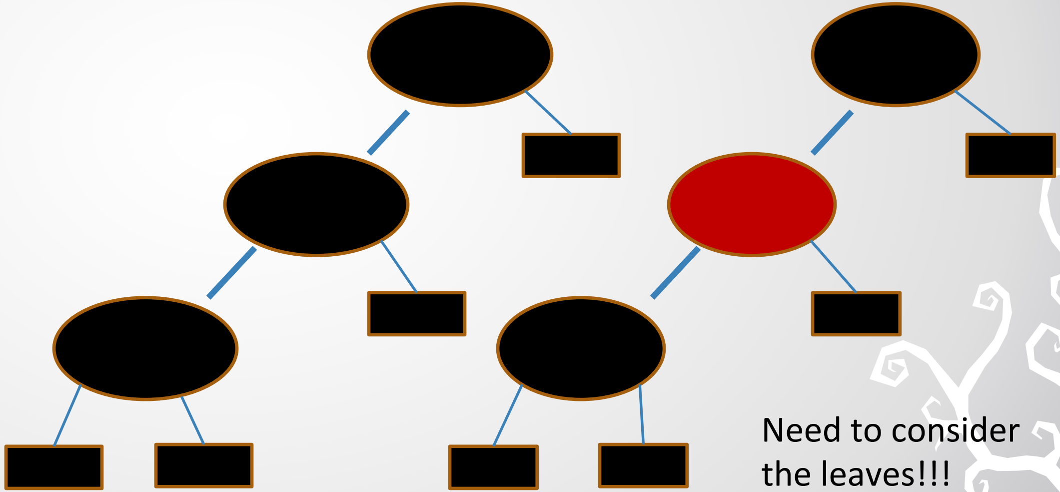




RB Trees Exercise

Exercise: given 3 nodes (not including leaves), what possible BST can you construct and how would you could you color them to satisfy the RB tree invariant?

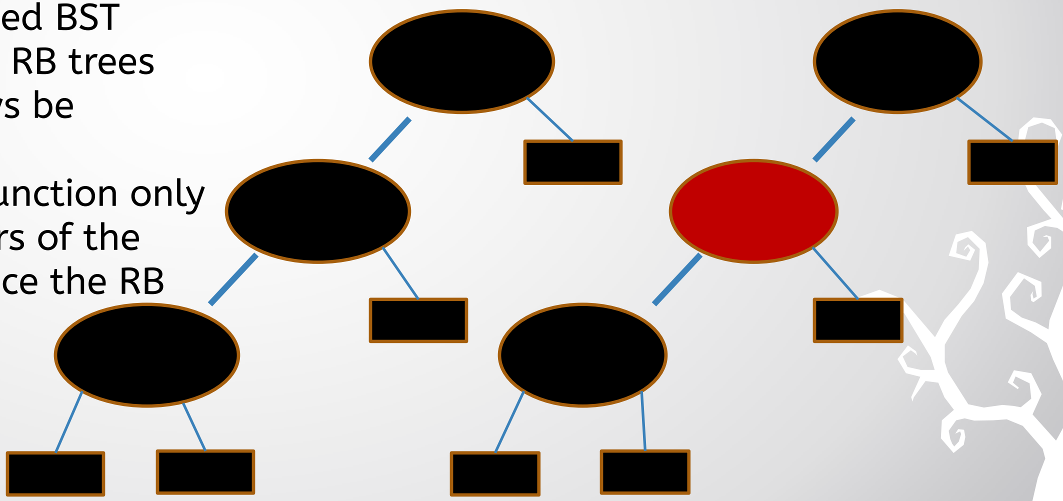




Need to consider the leaves!!!

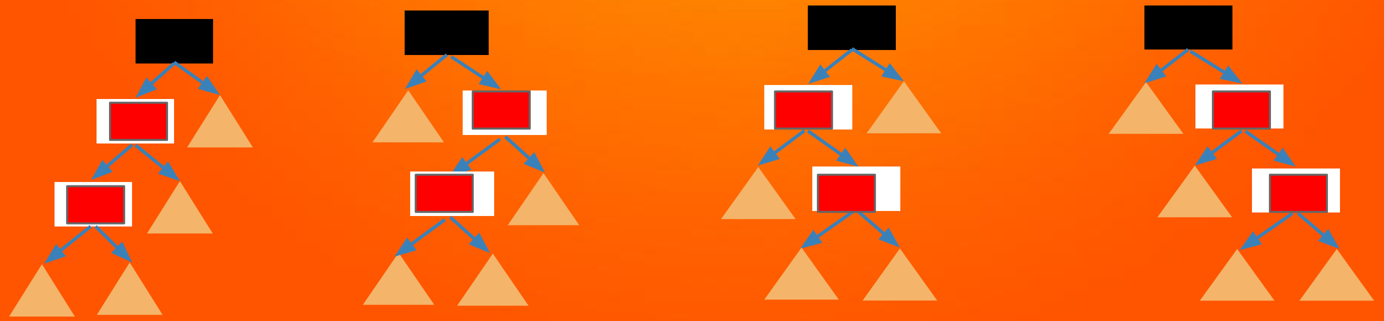
RB Trees Exercise

- To satisfy **global invariant** you could color both non-root nodes red
- However, this violates the **local invariant**
- Note: this BST is not a balanced BST
  - The coloring invariant for RB trees ensures that it will always be balanced!
  - This is why the balance function only needs to look at the colors of the nodes in order to rebalance the RB tree



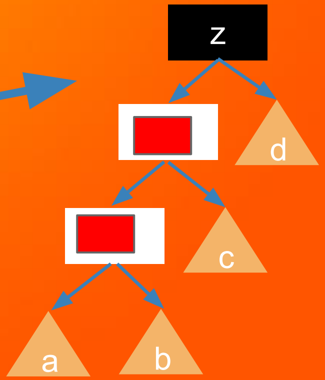
RB Trees Balance

- Okasaki's Algorithm: newly inserted node will always be red
- Creates 4 possible local invariant violations:



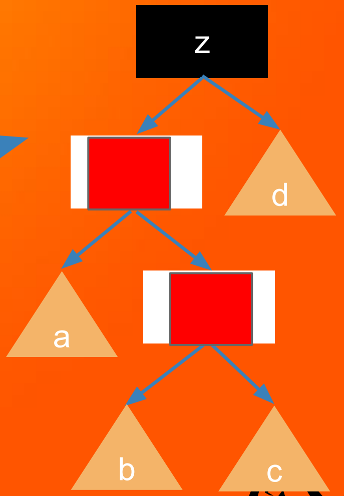
RB Trees Balance

```
let balance = function
  | Black, z, Node (Red, y, Node (Red, x, a, b), c), d
  | Black, z, Node (Red, x, a, Node (Red, y, b, c)), d
  | Black, x, a, Node (Red, z, Node (Red, y, b, c), d)
  | Black, x, a, Node (Red, y, b, Node (Red, z, c, d)) ->
    Node (Red, y, Node (Black, x, a, b), Node (Black, z, c, d))
  | a, b, c, d -> Node (a, b, c, d)
```



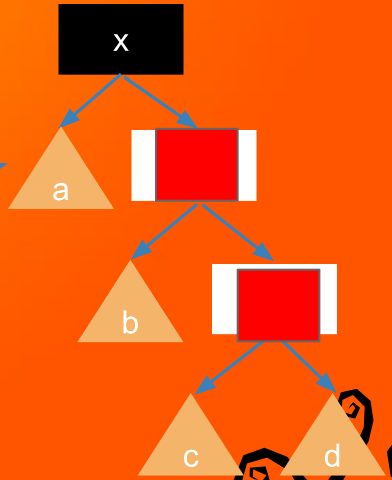
RB Trees Balance

```
let balance = function
  | Black, z, Node (Red, y, Node (Red, x, a, b), c), d
  | Black, z, Node (Red, x, a, Node (Red, y, b, c)), d
  | Black, x, a, Node (Red, z, Node (Red, y, b, c), d)
  | Black, x, a, Node (Red, y, b, Node (Red, z, c, d)) ->
     Node (Red, y, Node (Black, x, a, b), Node (Black, z, c, d))
  | a, b, c, d -> Node (a, b, c, d)
```



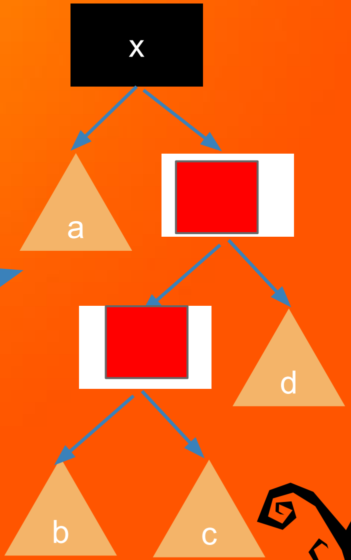
RB Trees Balance

```
let balance = function
  | Black, z, Node (Red, y, Node (Red, x, a, b), c), d
  | Black, z, Node (Red, x, a, Node (Red, y, b, c)), d
  | Black, x, a, Node (Red, z, Node (Red, y, b, c), d)
  | Black, x, a, Node (Red, y, b, Node (Red, z, c, d)) ->
     Node (Red, y, Node (Black, x, a, b), Node (Black, z, c, d))
  | a, b, c, d -> Node (a, b, c, d)
```



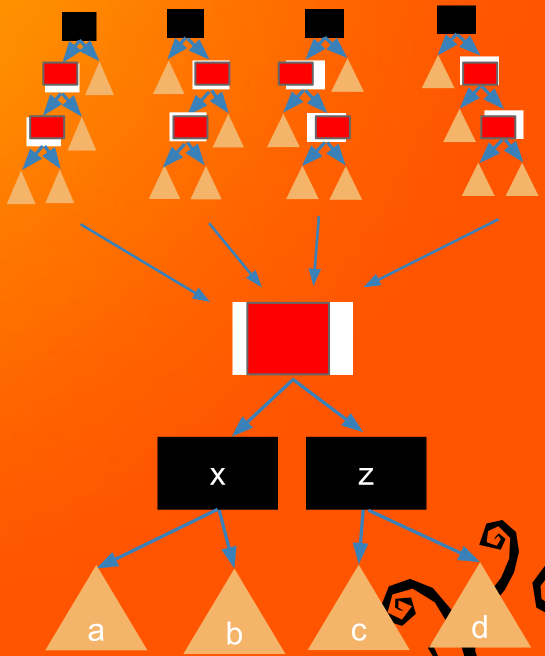
RB Trees Balance

```
let balance = function
  | Black, z, Node (Red, y, Node (Red, x, a, b), c), d
  | Black, z, Node (Red, x, a, Node (Red, y, b, c)), d
  | Black, x, a, Node (Red, z, Node (Red, y, b, c), d)
  | Black, x, a, Node (Red, y, b, Node (Red, z, c, d)) ->
      Node (Red, y, Node (Black, x, a, b), Node (Black, z, c, d))
  | a, b, c, d -> Node (a, b, c, d)
```



RB Trees Balance

```
let balance = function
  | Black, z, Node (Red, y, Node (Red, x, a, b), c), d
  | Black, z, Node (Red, x, a, Node (Red, y, b, c)), d
  | Black, x, a, Node (Red, z, Node (Red, y, b, c), d)
  | Black, x, a, Node (Red, y, b, Node (Red, z, c, d)) ->
     Node (Red, y, Node (Black, x, a, b), Node (Black, z, c, d))
  | a, b, c, d -> Node (a, b, c, d)
```



tree is already balanced!

RB Insert

- Insertion is a bit more tricky
- ex:

```
      B                   B
     / \     ==>         / \
    /   \    (B)        /   \
   R     R             R     R
                      /
                     B

      B                   B
     / \     ==>         / \
    /   \    (R)        /   \
   R     R             R     R
                      /
                     R
```

invariant broken!

Red-Black Trees - Insert

1. If the tree is empty, insert element at root as a red node
2. If the tree is non-empty:
   - If element is less than root node, recurse down the left subtree until reach a leaf and insert as red node
     - Rebalance tree
   - If element is greater than root, recurse down right subtree until you reach a leaf node and insert there as red node
     - Rebalance tree
3. Set root node to black

Red Black Trees Insert Example

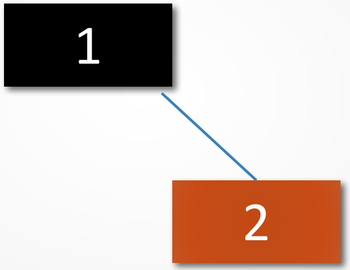
- 🎃 Exercise: given an empty Red-Black Tree, insert (by hand) the numbers 1 through 7 into the tree following the insert algorithm

Red Black Trees Insert Example

- Exercise: given an empty Red-Black Tree, insert (by hand) the numbers 1 through 7 into the tree following the insert algorithm

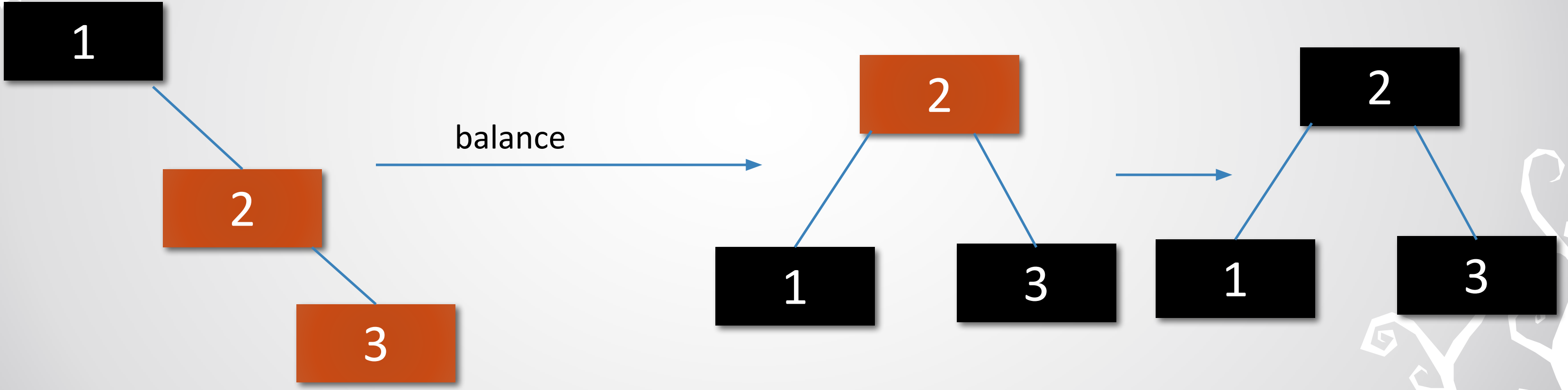
Red Black Trees Insert Example

- Exercise: given an empty Red-Black Tree, insert (by hand) the numbers 1 through 7 into the tree following the insert algorithm



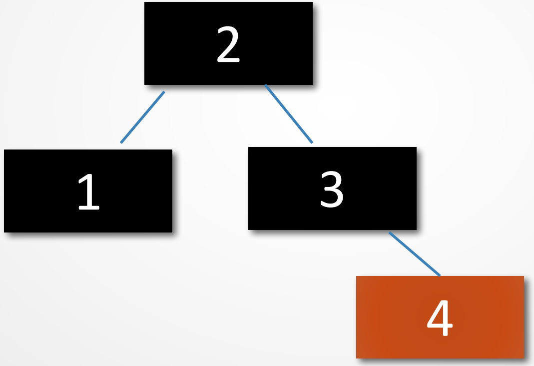
Red Black Trees Insert Example

- Exercise: given an empty Red-Black Tree, insert (by hand) the numbers 1 through 7 into the tree following the insert algorithm



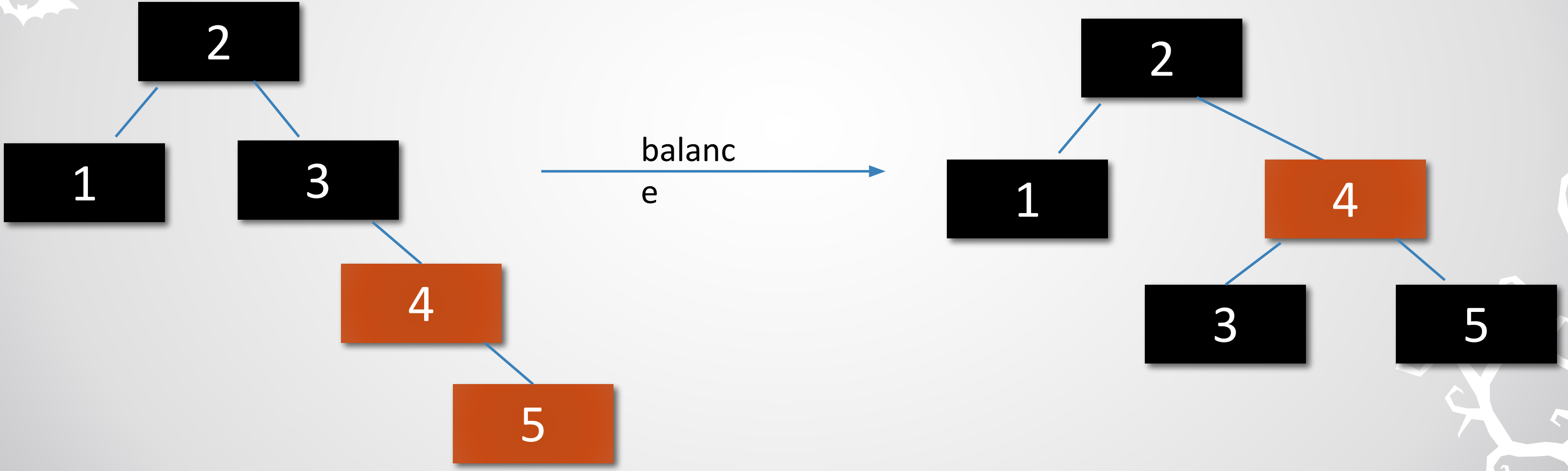
Red Black Trees Insert Example

- Exercise: given an empty Red-Black Tree, insert (by hand) the numbers 1 through 7 into the tree following the insert algorithm



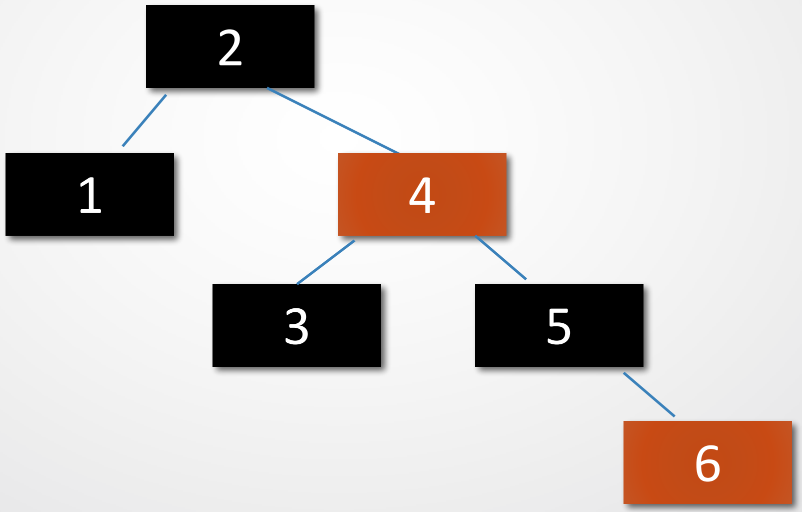
Red Black Trees Insert Example

- Exercise: given an empty Red-Black Tree, insert (by hand) the numbers 1 through 7 into the tree following the insert algorithm

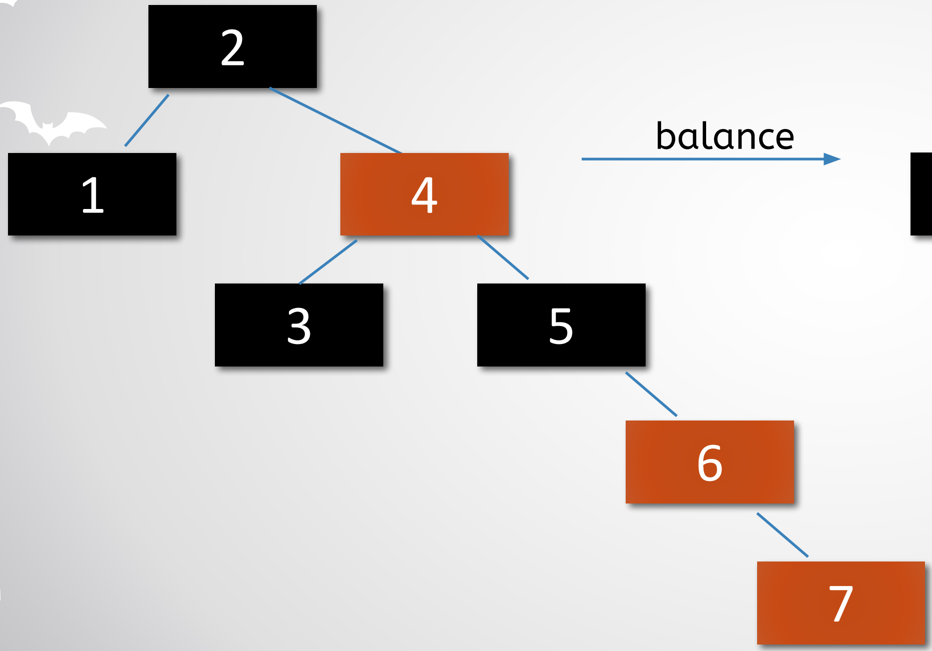


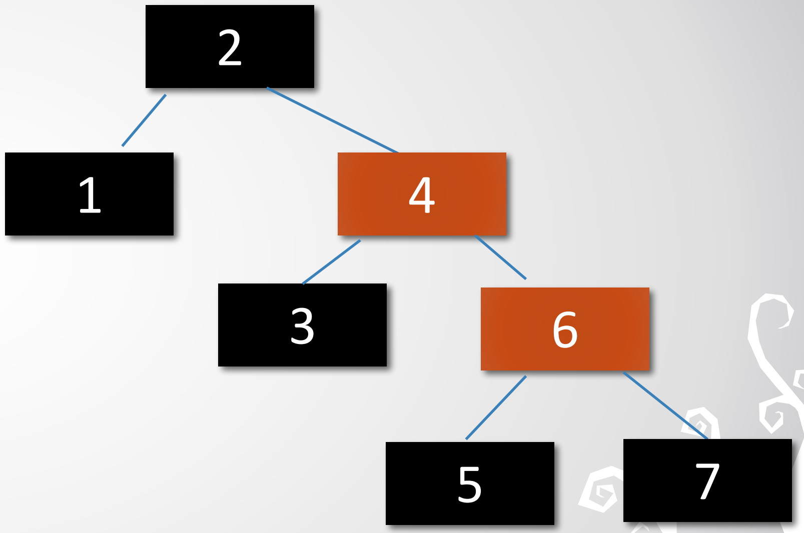
Red Black Trees Insert Example

- Exercise: given an empty Red-Black Tree, insert (by hand) the numbers 1 through 7 into the tree following the insert algorithm



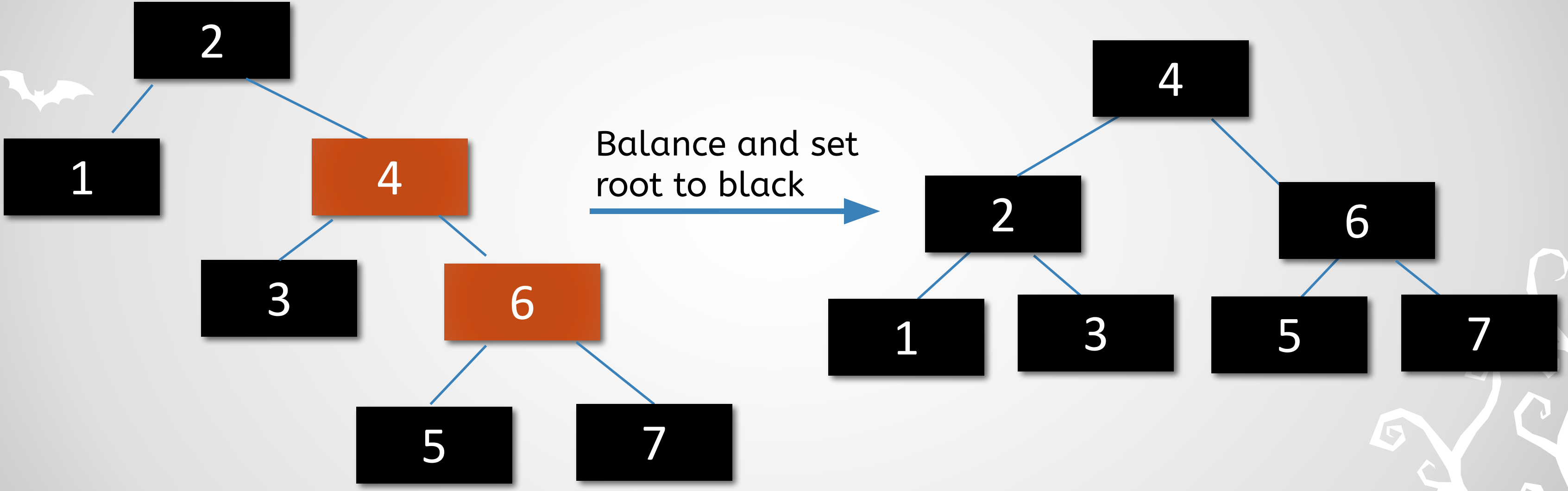
Red Black Trees Insert Example



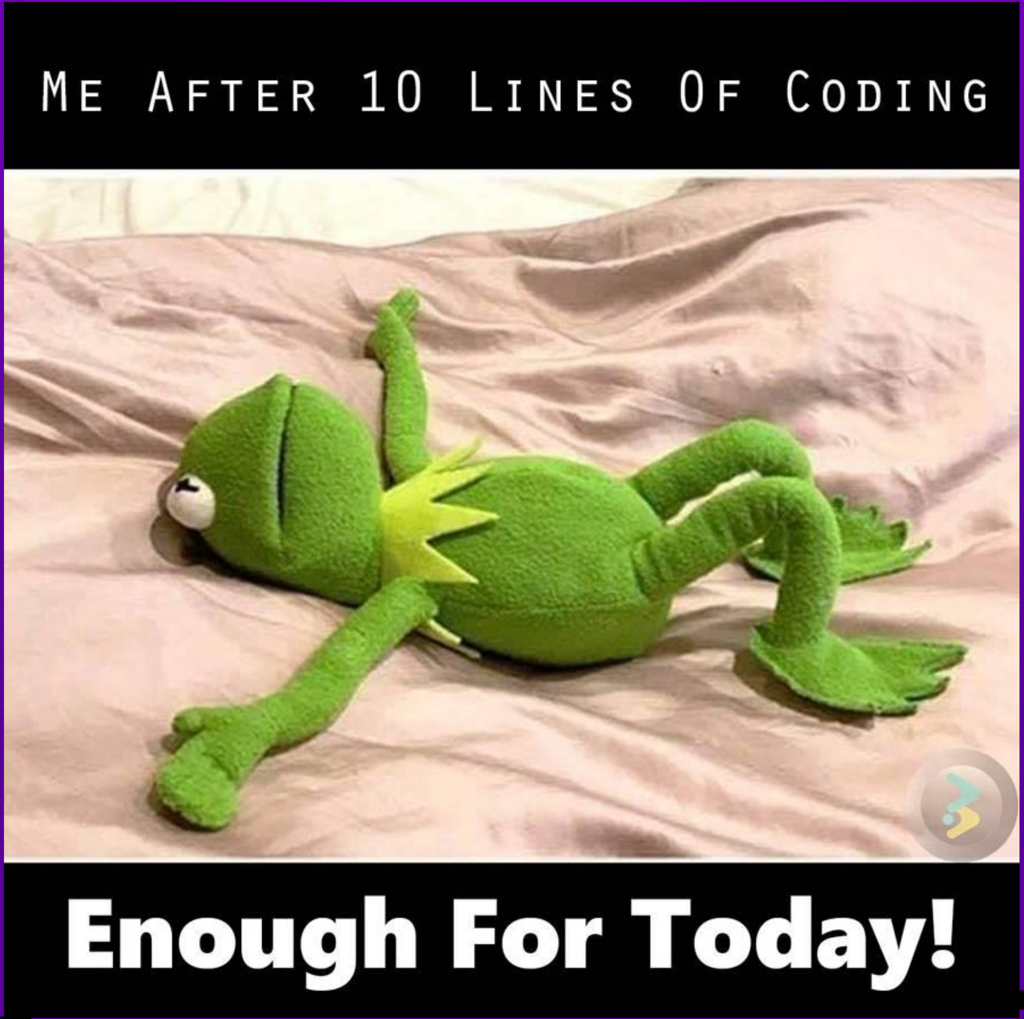


Not done yet!!

Red Black Trees Insert Example




Bernie
I am once again asking
you to let me know if you have any questions
imgflip.com


Me After 10 Lines Of Coding
Enough For Today!


Happy Halloween!
Stay safe!