CIS 5520
Advanced Programming

Fall 2024

Today: Wed Oct 9th

- HW #3 due one week from Thursday (Sat, Queue & AVL)
- Today: **discussion on RedBlack module & Hughes video**
- **Next week**: GADTs discussion (Monday) and demo (Wed)
  - Code in github repo (07-GADTs), notes on website, don't forget about the quiz

Project Proposal

- Due: Thursday, October 24th
- Be creative! This is just a rough draft
  - See write up for project topics to avoid
  - Happy to talk about project ideas after class/during OH
- Teams of two
  - If you propose a more significant project, you may form a group of three, but your proposal should describe how to divide the work

Discussion plan

- Quiz discussion
- Hughes' video

[5 min break]

- Persistent Data Structures
- Red/Black Trees

Where do properties come from?

- Hughes' talk gives several kinds of properties that we can define for QuickCheck
  - Validity – make sure that representation invariants are maintained
  - Postcondition – make sure that operations do what they should
  - Metamorphic – think about all combinations of operations
  - Model-based – compare this implementation with a less buggy one
- All code is a source of potential bugs, including *testing* code
  - Buggy test: finds a bug that doesn't exist
  - Buggy test: misses a bug that it should catch

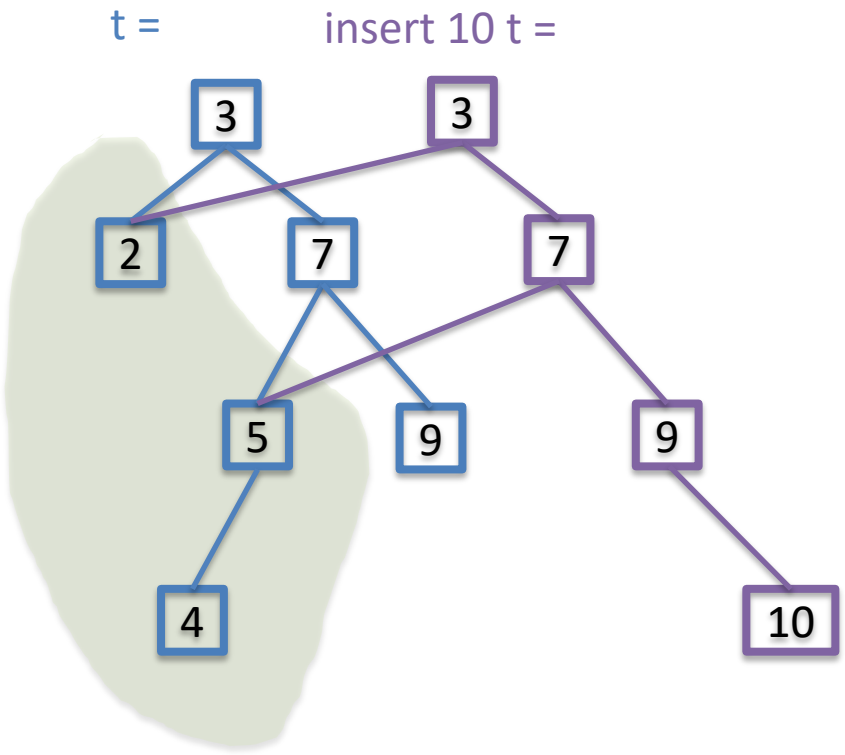
Model-based testing

- Where do we get another implementation?
  - Existing implementations: Data.Set for RBTs
  - Simpler implementations: ordered lists w/o duplicates for RBTs
  - Other examples? Really depends on the situation…
- When is model-based testing not appropriate
  - No model available, or existing models are buggy
  - Model over-specifies desired properties
- Model-based testing is not always practical, that's why other kinds of properties are important

Persistent Data structures

- What do you mean by a 'persistent' data structure?
- How efficient is this implementation compared to a mutable data structure?
- How do RedBlack trees really work?

Persistent Binary-Search Tree



- insertion returns new tree
- Recreates the path from new leaf to root
- In a *balanced* tree, this path has at most length O(log n)
- Rule of thumb: persistent structures cost at most O(log n) more than mutable structures

RED BLACK TREES

Insertion

```
data Color = R | B
data T a   = E | N Color (T a) a (T a)
newtype RBT a = Root (T a)

insert :: Ord a => a -> RBT a -> RBT a
insert s (Root x) = Root (ins s)
   where ins E = N R E x E
         ins s@(N color a y b)
             | x < y     =          N color (ins a) y b
             | x > y     =          N color a y (ins b)
             | otherwise = s
```

Temporarily suspend invariant: Result of ins may create a red root or a red node with a red child.

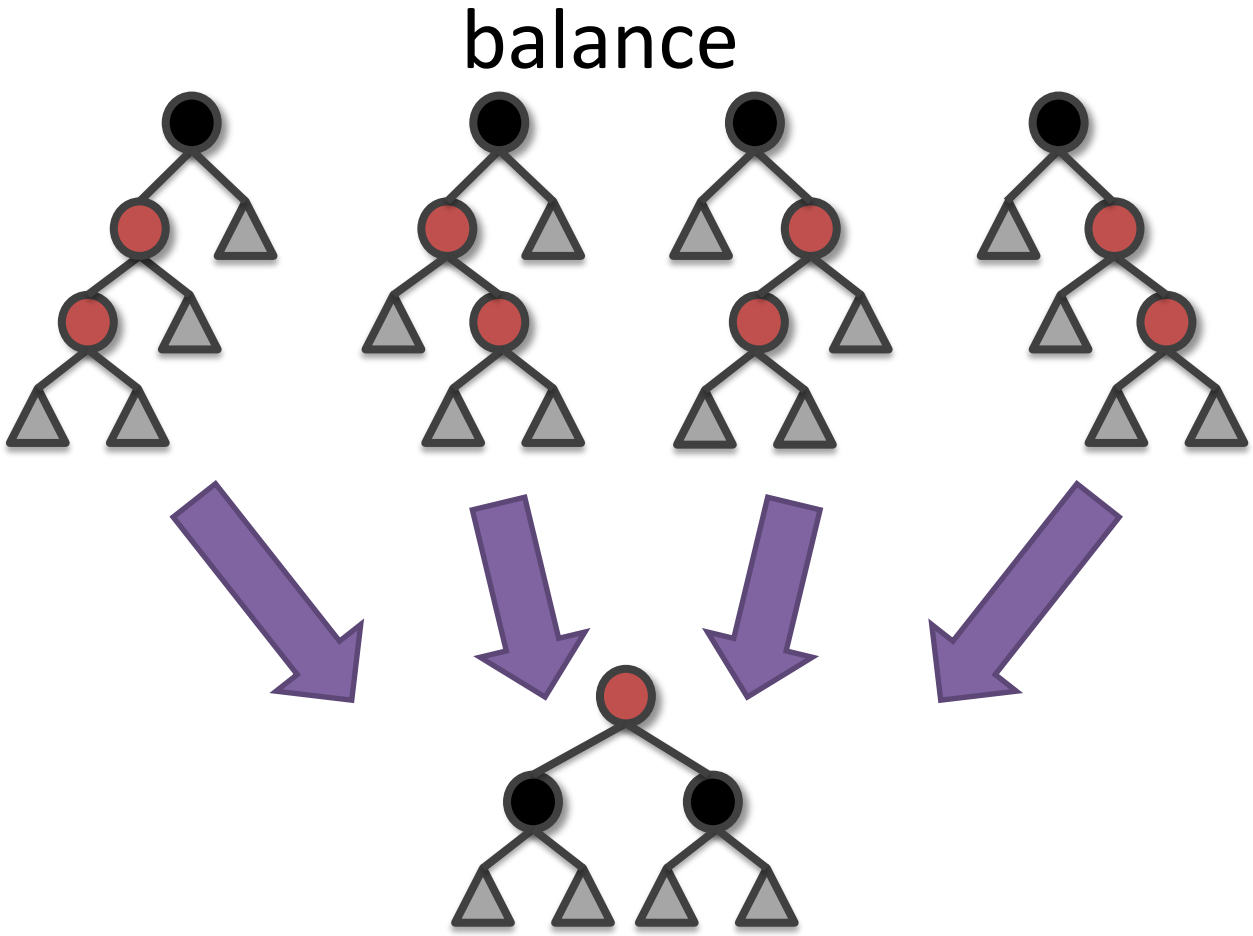
Insertion

```
data Color = R | B
data T a   = E | N Color (T a) a (T a)
newtype RBT a = Root (T a)

insert :: Ord a => a -> RBT a -> RBT a
insert s (Root x) = blacken (ins s)
   where ins E = N R E x E
         ins s@(N color a y b)
             | x < y     = balance (N color (ins a) y b)
             | x > y     = balance (N color a y (ins b))
             | otherwise = s
         blacken (N _ a x b) = Root (N B a x b)
```

Temporarily suspend invariant: Result of ins may create a red root or a red node with a red child.

Two fixes:
- blacken if root is red at the end
- rebalance two internal reds


balance

balance

```
balance :: T a -> T a
balance (N B (N R (N R a x b) y c) z d) =
         N R (N B a x b) y (N B c z d)
balance (N B (N R a x (N R b y c)) z d) =
         N R (N B a x b) y (N B c z d)
balance (N B a x (N R (N R b y c) z d)) =
         N R (N B a x b) y (N B c z d)
balance (N B a x (N R b y (N R c z d))) =
         N R (N B a x b) y (N B c z d)
balance (N color a x b) = N color a x b
```