Testing Library Components: Experience with Red-Black Trees

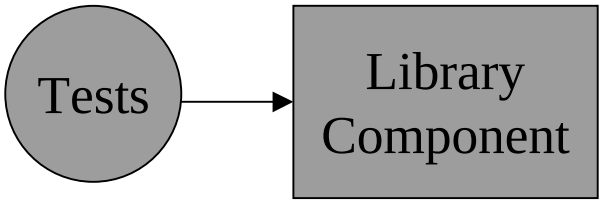
formalWARE Presentation

October 8, 1997

Lee White
Department of Computer Science
University of Victoria

Component Testing:

Types of Libraries: STL for C++,
Ada Library



- Specification Test (Black Box)
- Implementation-Based Test

Objective: This testing should be as thorough as possible in terms of detecting errors, but very efficient in terms of tester's time; use automated testing to the greatest extent possible

Implementation Based:

Given some information about the implementation, we want to take advantage of this in our testing.

Example: If container is known to be implemented as red-black tree, don’t build the trees, but specify the inputs to test these structures.

Implementation-Dependent:

Given the exact method of implementation, base tests on this; less desirable, because implementation details could change, in this library or from one library to another.

Specification Based? \ Implementation Dependent?	Y	N
Y	Both	Spec. Based
N	Impl. Based & Dep.	Impl. Based

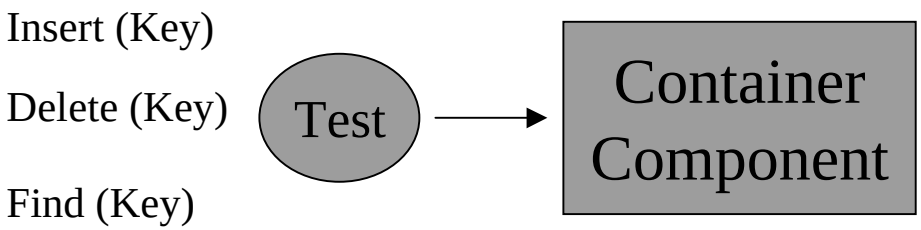
Look at case of Specification-Based and Implementation-Dependent: Ex: Tables of data of a certain kind will be used in solution, given as part of the specification.

Testing: Specification-Based vs Implementation-Based: Detected by motivation.

Testing: Implementation-Dependent: Detected by specification.

We wanted to do Implementation-Based Testing for Containers:

Containers: Red-Black Trees
Deques
Doubly-Linked Lists
Vectors (Arrays)

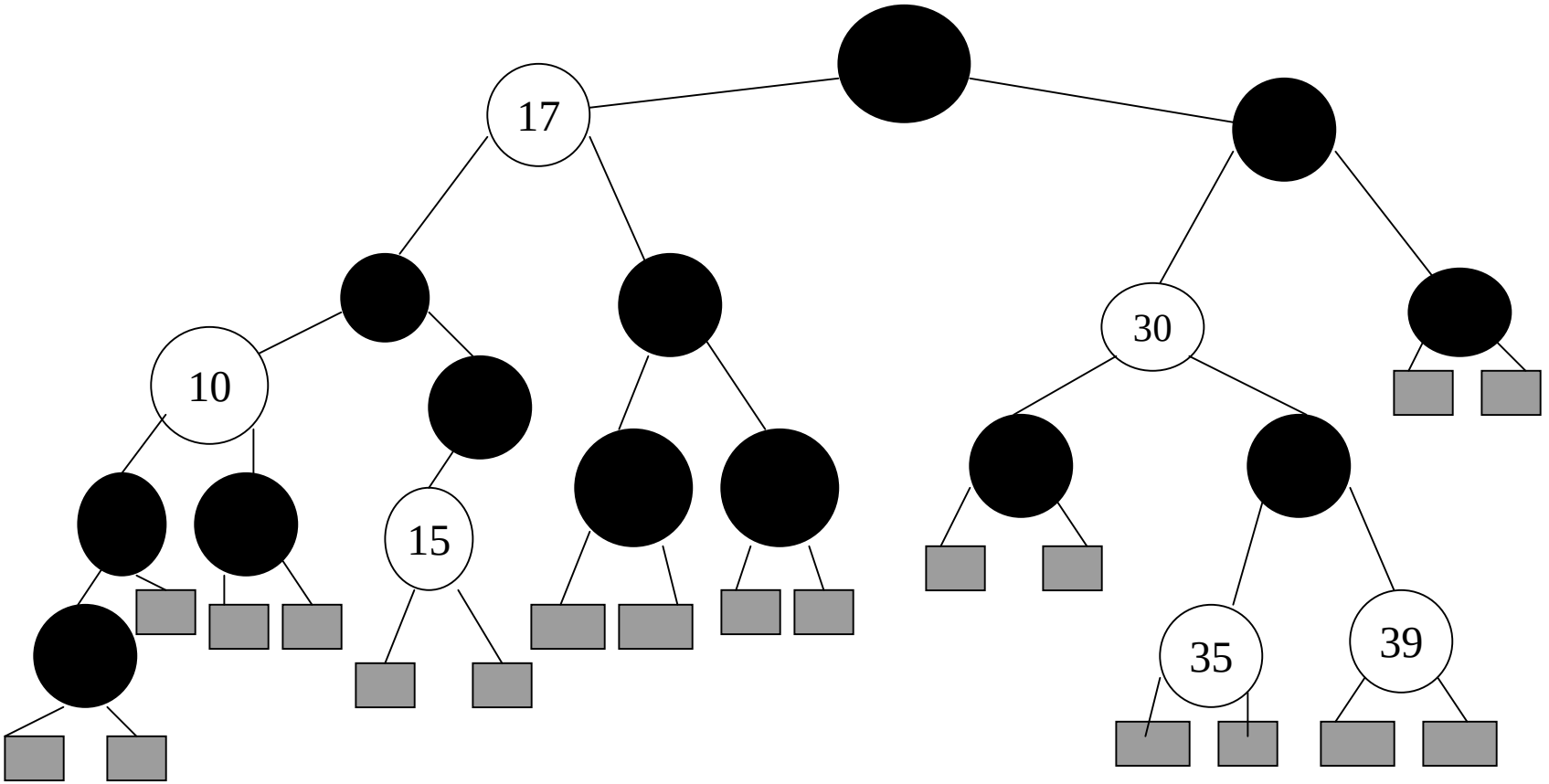


Problems of Observability and Controllability

A Red-Black Tree:

Is a Binary Search Tree (BST) in which every node has either the color red or black, and which satisfies:

- Every leaf (nil) is black.
- If anode is red, then both its children are black.
- Every simple path from a node to a descendant leaf contains the same number of black nodes.

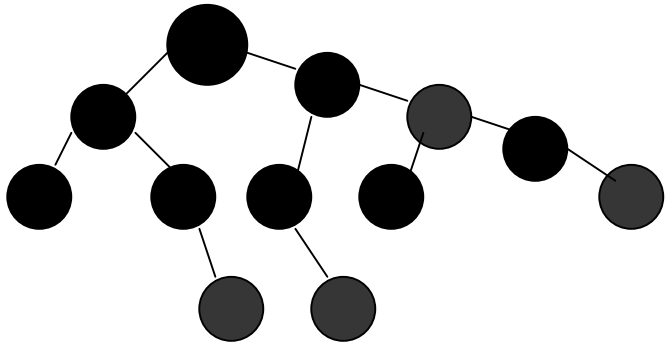

17
10
15
30
35
39

An AVL Tree (or Balanced BST) is a BST which satisfies the following condition:

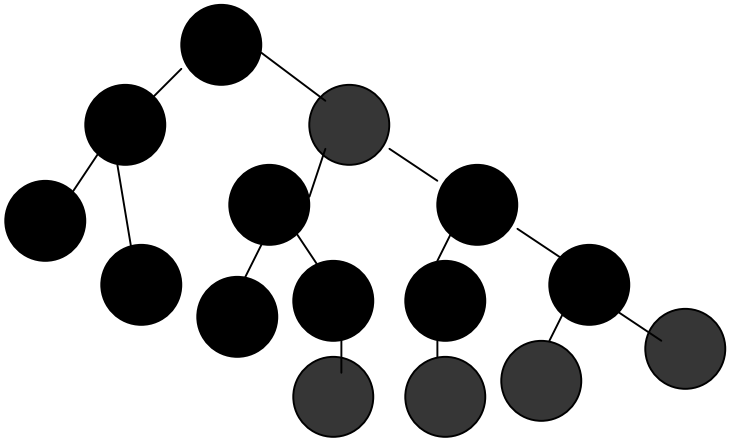
- *For every node in the BST, the maximum path lengths in the left subtree and right subtree differ by no more than one.*

Red-Black Trees vs AVL Trees

An AVL Tree can be colored as a Red-Black Tree (actually a Fibonacci Tree):



A Red-Black Tree which cannot be an AVL Tree:



Approach:

Implementation-Based Testing of Red-Black Trees:

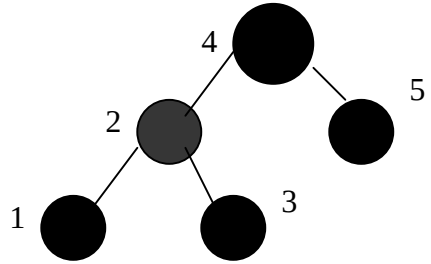
- Construct all red-black trees automatically with <= N nodes, and then perform insert and delete operations on them.

Generation Technique:

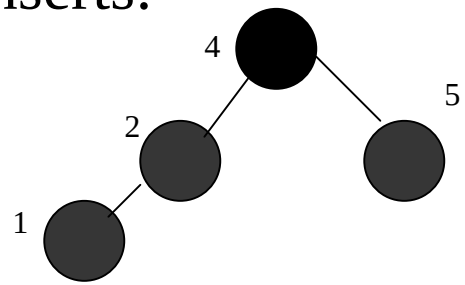
•Breadth-first generation of red-black trees.

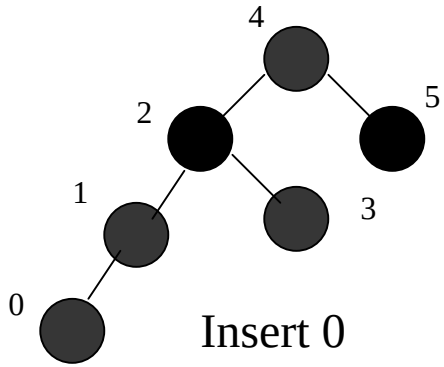
•We quickly found that some red-black trees could not be generated in this way, even though their shape as BST’s were as targeted.

Initial RB-Tree



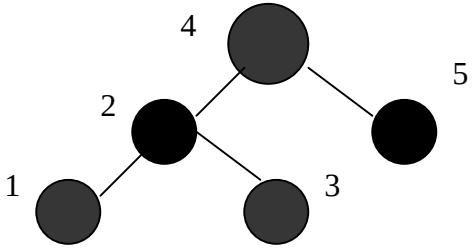
Inserts:

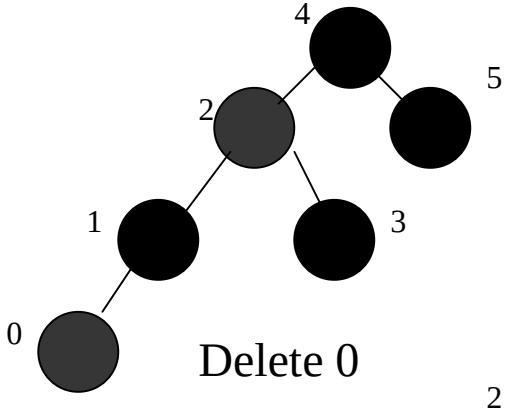




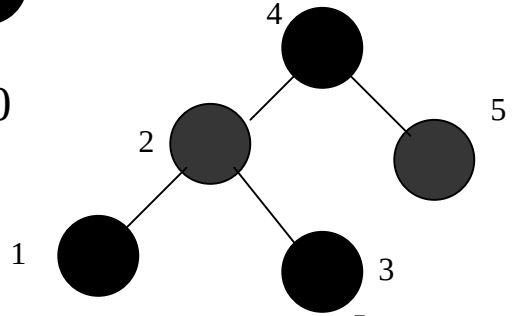
Insert 0

RB-Tree Obtained:





Delete 0



Desired RB-Tree

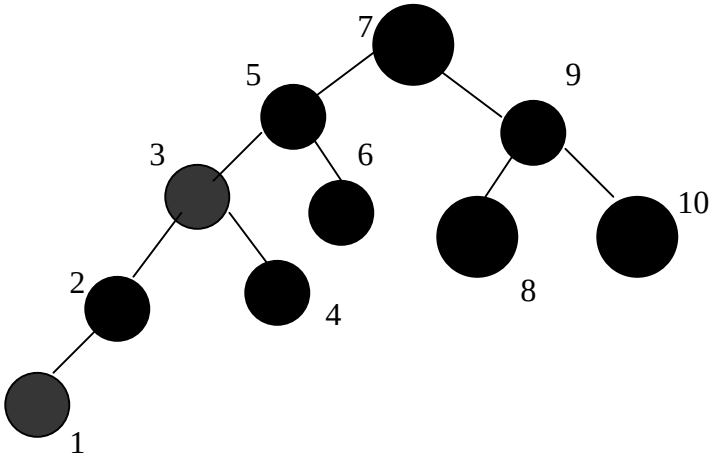
•The next objective as implementation-based testing was to generate all BST's using breadth-first generation.

•Breadth-first generation worked OK until N=10 node counter example.

•Could not generate this BST by breadth-first generation method.

•Could not generate it by **any** permutation of input keys.

•Could not generate it by inputs alone.



Counter example for N=10

Why? Because adding node 1 will require rotations, which will change the shape of the BST.

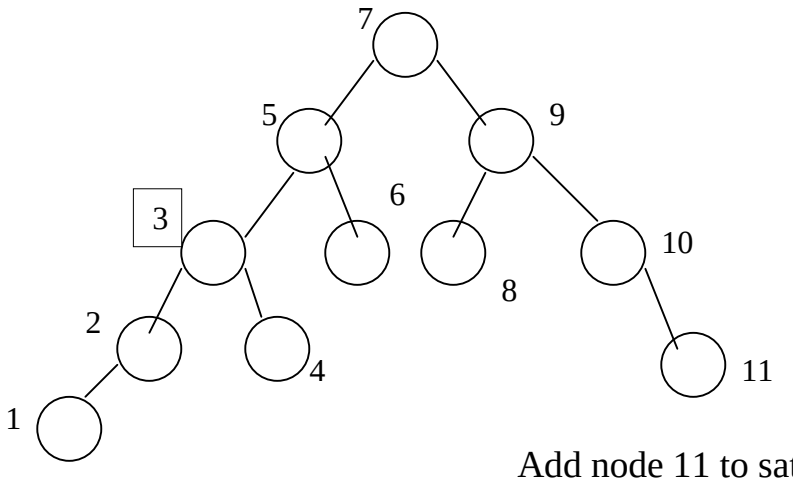
Necessary and Sufficient Conditions for BST's given as red-black trees to be generated by the breadth-first method using the insert and elete algorithms of Cormen, Leiserson & Rivest:

For every subtree of the given BST, compute the height of the left subtree, HL, and of the right subtree, HR.

Then for every subtree, either

1) | HL - HR | <= 1, or

2) Max (HL, HR) = k (even) and
   Min (HL, HR) = (k - 2).



BST Counterexample

Add node 11 to satisfy necessary
and sufficient conditions at every node.

Now breadth-first algorithm can be used to generate this BST, and then node 11 can be deleted.

We are now working on the conditions to generate all red-black trees with N nodes.