- For lab 8 we will be creating a collection of JUnit 5 test that should do a good job of testing our binary search tree functionality
  - There are 28 tests in all
  - Passing all of the tests does not guarentee that your binary search tree code is completely correct, but does get you started
    - The insert and delete methods need more testing
- Unfortunately we haven't really had a chance to get much if anything implemented in our binary search tree code yet, but once you get started, these tests should help ensure things are working well
- For the tests that return a value, we will compare the return value with the expected return value
  - Such as getHeight(), getRoot(), getMin(), getMax(), etc
- For the tests that do not return a value, we will use a modified version of the inOrderWalk to verify that all nodes are in their proper places
  - Such as insertNode(), deleteNode()

- Insert node tests
  - Insert key 20
  - Insert keys 20, 10
  - Insert keys 20, 30
  - Insert keys 20, 10, 30
  - Insert keys 20, 10, 30, 5, 15, 25, 35
    - Check height
    - Check successor of the root
    - Check the predecessor of the root

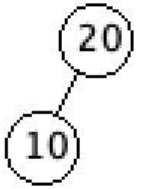
- Insert node tests (cont)
  - Insert keys 50, 25, 75, 15, 40, 60, 90, 10, 20, 30, 45, 55, 70, 80, 95, 8, 12, 28, 4, 9, 11, 13, 27, 29, 53, 57, 54, 56, 58, 93, 91
    - Check height
    - Check height for subtree rooted at 60
    - Check max
    - Check min
    - Check root
    - Check get node found
    - Check get node not found
    - Check predecessor on node with no left child
    - Check predecessor on node with left child
    - Check no predecessor for node with min key

- Insert node tests (cont)
  - Insert keys 50, … (cont)
    - Check succecessor on node with no right child
    - Check succecessor on node with right child
    - Check no succecessor for node with max key
    - Delete node 45
    - Delete nodes 45, 95
    - Delete nodes 45, 95, 60
    - Delete nodes 45, 95, 60, 50

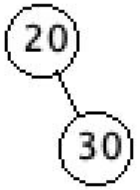
- Insert node tests (cont)
  - Insert keys {20, 10, 30, 35, 5, 35, 15, 35, 25, 35} and verify that the node with key 35 has a count of 4
  - Insert keys {20, 10, 30, 35, 5, 35, 15, 35, 25, 35}, delete two copies of the node with key 35 and verify that the node with key 35 has a count of 2

- After inserting key 20

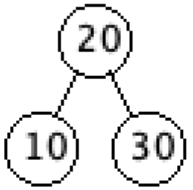
- After inserting keys 20 10



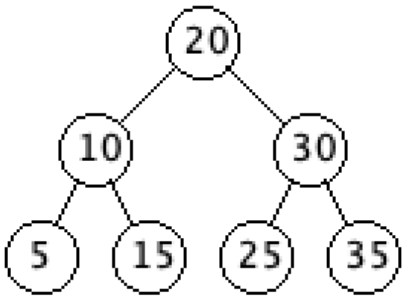
- After inserting key 20 30



- After inserting 20 10 30

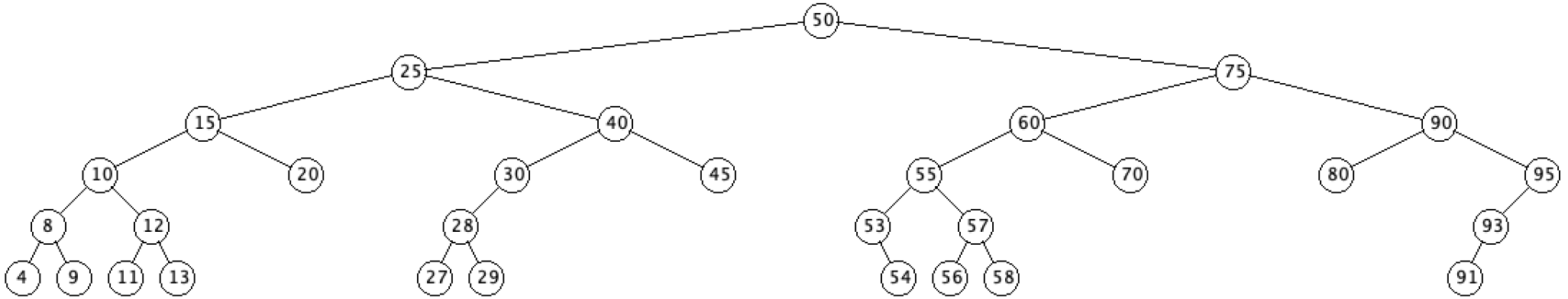


- After insert keys 20, 10, 30, 5, 15, 25, 35



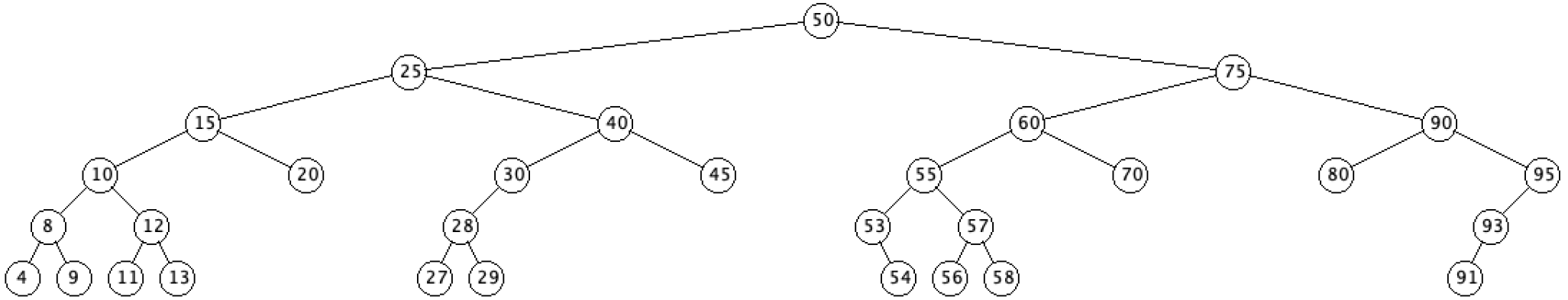
  - The height is 2
  - Successor of the root is (25,30,,,1)
  - Prececcessor of the root is (15,10,,,1)

- After inserting keys 50, 25, 75, 15, 40, 60, 90, 10, 20, 30, 45, 55, 70, 80, 95, 8, 12, 28, 4, 9, 11, 13, 27, 29, 53, 57, 54, 56, 58, 93, 91



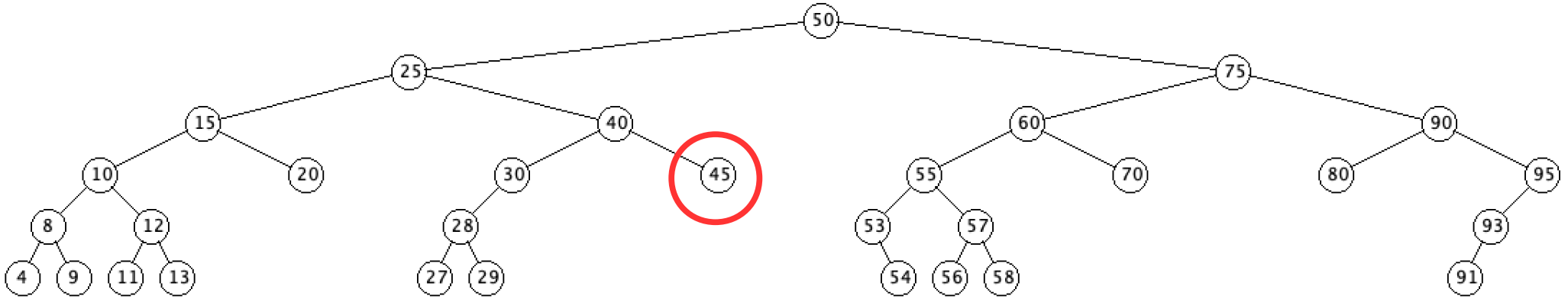
  - The height is 5
  - The height of the subtree rooted at 60 is 3
  - The max is (95,90,93,,1)
  - The min is (4,8,,,1)
  - The root is (50,,25,75,1)
  - The node with key 91 is (91,93,,,1)
  - If we search for a node with key 99 the result is null

- After inserting keys 50, … (cont)

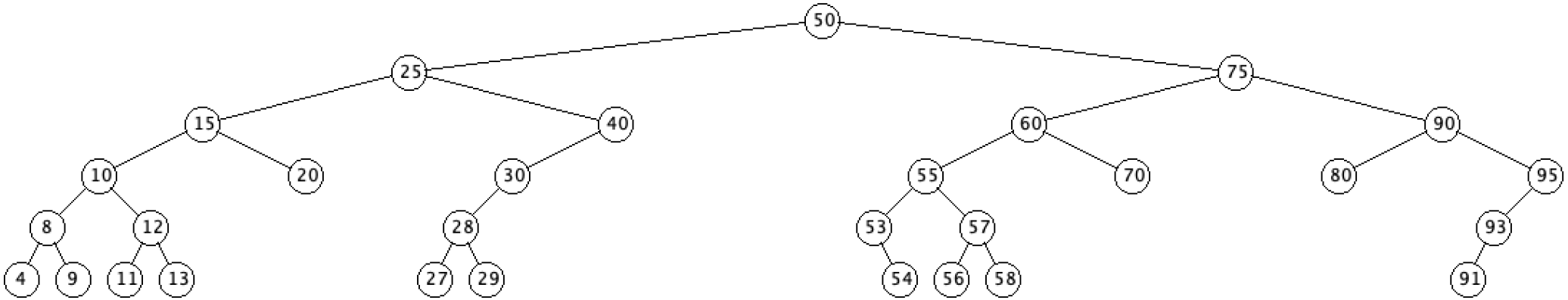


  - The predecessor of the node with key 91 is (90,75,80,95,1)
  - The predecessor of the node with key 10 is (9,8,,,1)
  - The predecessor of the node with key 4 is null
  - The successor of the node with key 45 is (50,,25,75,1)
  - The successor of the node with key 50 is (53,55,,54,1)
  - The successor of the node with key 95 is null

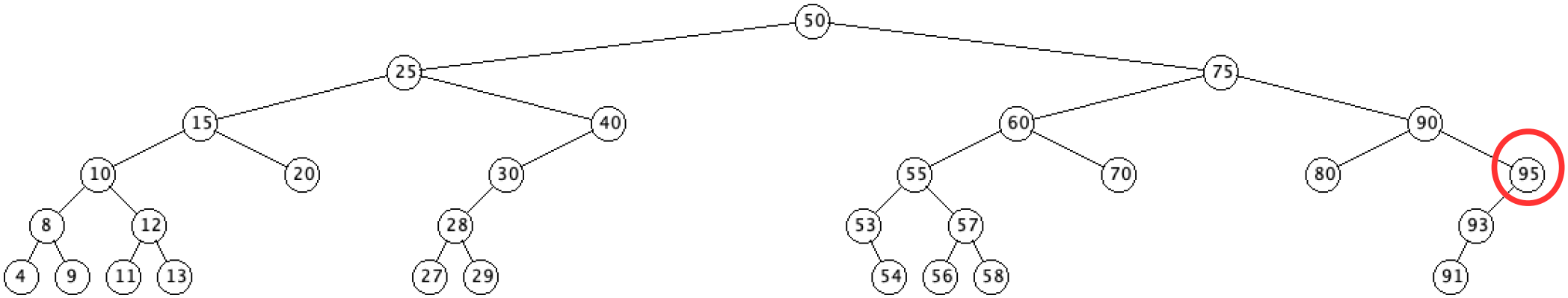
- After inserting keys 50, … (cont)



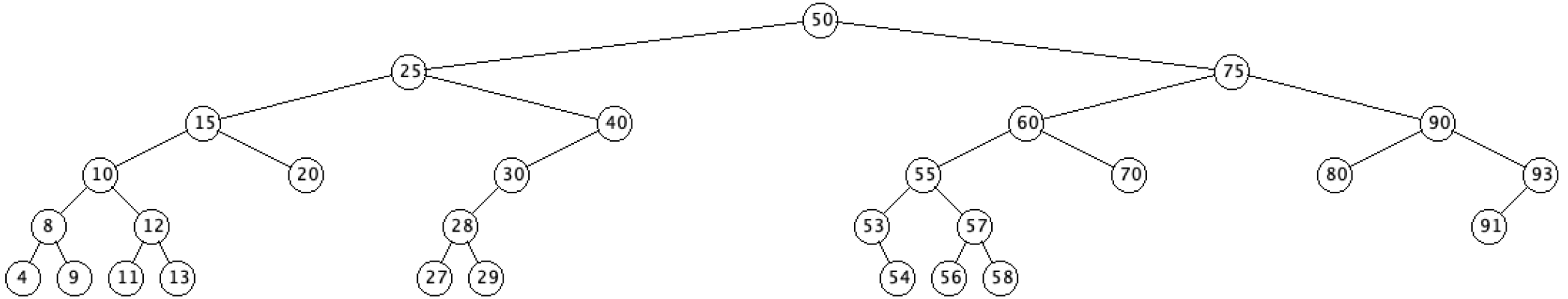
- Deleting 45 gives us



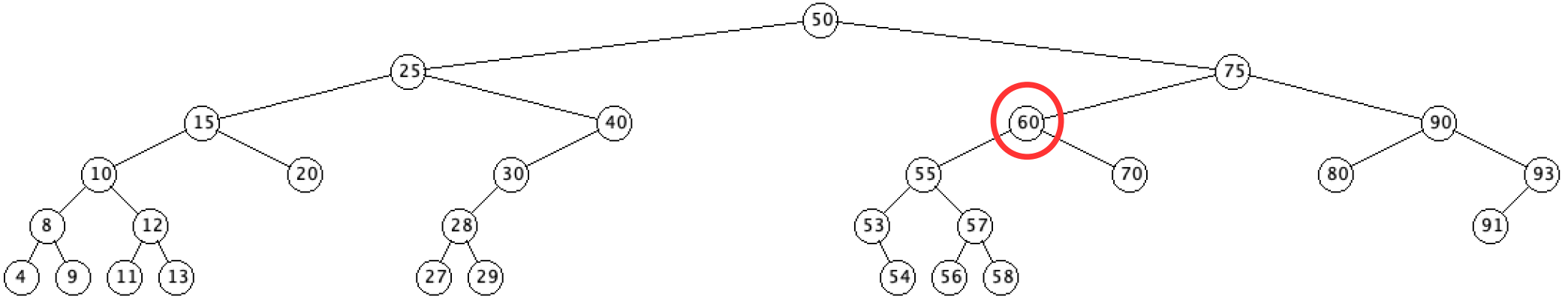
- After inserting keys 50, … (cont) and deleting 45



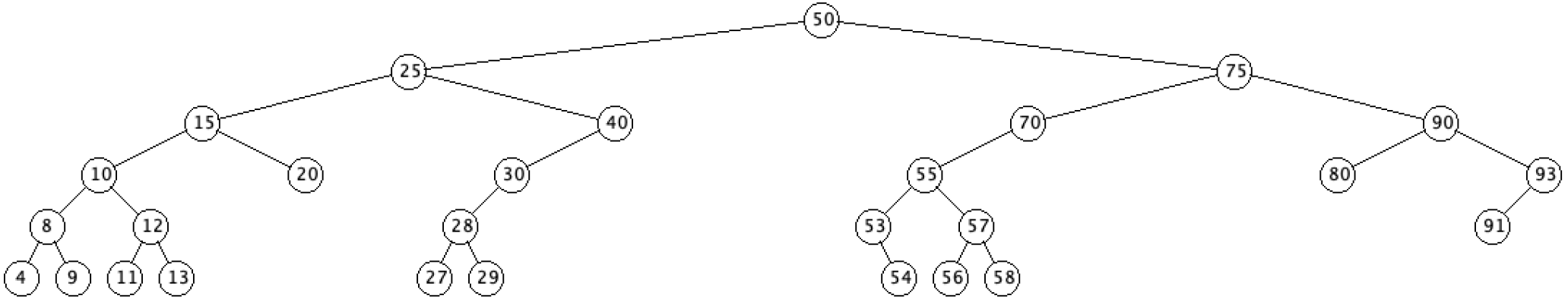
- Deleting 95 gives us



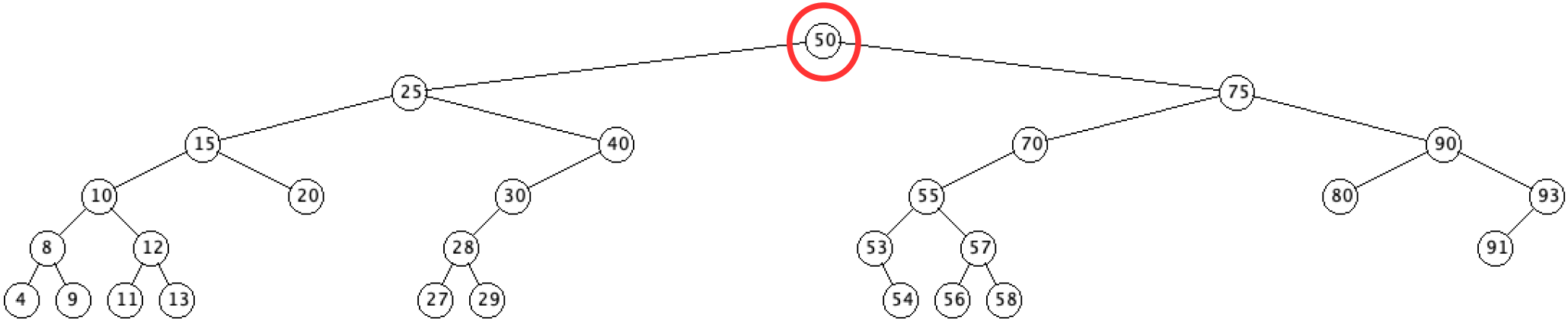
- After inserting keys 50, … (cont) and deleting 45 & 95



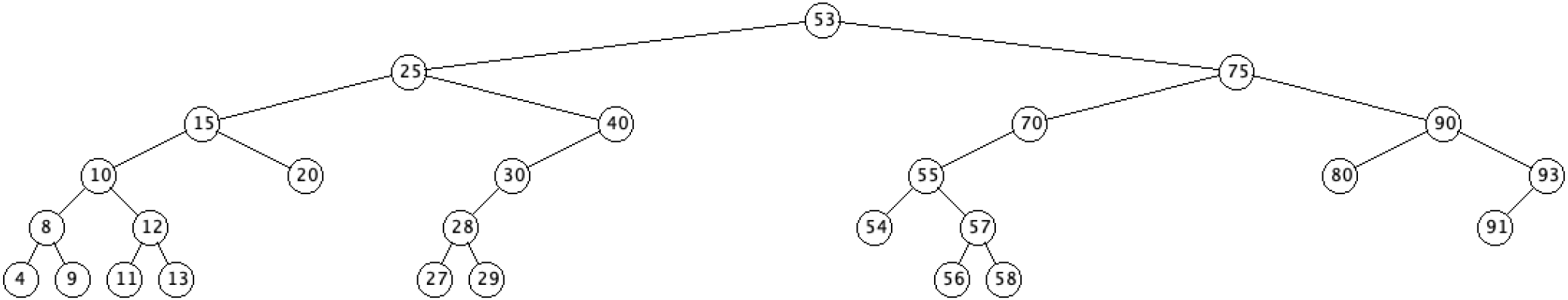
- Deleting 60 gives us



- After inserting keys 50, … (cont) and deleting 45, 95, & 60



- Deleting 50 gives us



- My test file is approximately 660 lines long
  - Around 250 lines is just declaring and instantiating the test result data
  - And more than a few of the tests are very similar
- I haven't finalized the lab yet, but it will be something like
  - I will provide the JUnit test program with some of the tests removed or tests functionality removed
  - I will provide a list of which tests need to be either added or updated and what needs to be done
- The dream for me is that once we have completed the lab
  - We will all have a better understanding of using JUnit test
  - We will all have a good collection of tests to help us while working on program 5

- When you are implementing your binary search tree the three methods that are most challenging are
  - insertNode
  - deleteNode
  - shiftNode
- The tests related to inserting 1, 2, 3, 7, and 31 nodes should help out when implementing insertNode
- And the tests related to deleting nodes 45, 95, 60, and 50 should help out when implementing deleteNode and shiftNode
- If you take a class in software engineering, you may learn about a software development concept called “test driven software development” in which you develop the tests and test data prior to developing the software
  - And use the tests and test data to drive the software development
  - We might claim that is what we are doing for program 5

- Here are the methods in the test file
- static void setUpBeforeClass() throws Exception
  - Initialization of stuff prior to any test
- void setUp() throws Exception
  - Initialization prior to each test
  - Instantiate two binary search trees, one for the small trees (tree) and one for the 31 key tree (t)
  - Add the nodes associated with the 31 key tree to t
- void testInsertRoot()
  - Insert a node with key 20 to tree and verify the tree is correct by verifying the inOrderWalk returns the correct list of nodes
  - Since there is only one node in tree, it is the root

- Here are the methods in the test (cont)
- void testInsertTwoNodes()
  - Insert nodes with keys 20 and 10 to tree and verify the tree is correct by verifying the inOrderWalk returns the correct list of nodes
- void testInsertTwoNodesB()
  - Insert nodes with keys 20 and 30 to tree and verify the tree is correct by verifying the inOrderWalk returns the correct list of nodes
- void testInsertThreeNodes()
  - Insert nodes with keys 20, 10, and 30 to tree and verify the tree is correct by verifying the inOrderWalk returns the correct list of nodes

- Here are the methods in the test (cont)
- void testInsertSevenNodes()
  - Insert nodes with keys 20, 10, 30, 5, 15, 25, and 35 to tree and verify the tree is correct by verifying the inOrderWalk returns the correct list of nodes
- void testGetHeightForSevenNodesTree()
  - Insert nodes with keys 20, 10, 30, 5, 15, 25, and 35 to tree and verify the height is 2
- void testGetSuccessorOfRootForSevenNodesTree()
  - Insert nodes with keys 20, 10, 30, 5, 15, 25, and 35 to tree and verify the successor of the root is (25,30,,)
- void testGetPredecessorOfRootForSevenNodesTree()
  - Insert nodes with keys 20, 10, 30, 5, 15, 25, and 35 to tree and verify the predecessor of the root is (15,10,,)

- Here are the methods in the test (cont)
- void testInsertThirtyOneNodes()
  - Get an in order walk for the 31 node tree t, and verify that that all of the nodes are correct
- void testGetHeight()
  - Verify that the height of the 31 node tree t is 5
- void testGetHeightForSubtreeRootedAt60()
  - Verify that the height of the subtree rooted at the node with key 60 of the 31 node tree t is 3
- void testGetMax()
  - Verify the max node of the 31 node tree t is (95,90,93,)
- void testGetMin()
  - Verify the min node of the 31 node tree t is (4,8,,)

- Here are the methods in the test (cont)
- void testGetRoot()
  - Verify the root of the 31 node tree t is (50,,25,75)
- void testGetNode()
  - Verify the getNode method returns (91,93,,) as the node with key 91 when searching the tree starting at the root
- void testGetNodeNotFound()
  - Verify the getNode method returns null when searching for the node with key 99
  - The tree does not have a node with key 99
- void testPredecessorWithNoLeftChild()
  - Verify the getPredecessor method returns (90,75,80,95) when searching for the predecessor of the node with key 91

- Here are the methods in the test (cont)
- void testPredecessorWithLefthild()
  - Verify the getPredecessor method returns (9,8,,) when searching for the predecessor of the node with key 10
- void testPredecessorOnMin()
  - Verify the getPredecessor method returns null when searching for the predecessor of the node with key 4
  - The key 4 is the smallest key in the tree t
- void testSuccessorWithNoRightChild()
  - Verify the getSuccessor method returns (50,,25,75) when searching for the successor of the node with key 45
- void testSuccessorWithRightChild()
  - Verify the getSuccessor method returns (53,55,,54) when searching for the successor of the node with key 50

- Here are the methods in the test (cont)
- void testSuccessorOnMax()
  - Verify the getSuccessor method returns null when searching for the successor of the node with key 95
  - The key 95 is the largest key in the tree t
- void testDeleteNode45()
  - Verify the tree with 31 nodes is correct after deleting the node with key 45 using the results of inOrderWalk
- void testDeleteNode45And95()
  - Verify the tree with 31 nodes is correct after deleting the nodes with keys 45 and 95 using the results of inOrderWalk

- Here are the methods in the test (cont)
- void testDeleteNode45And95And60()
  - Verify the tree with 31 nodes is correct after deleting the nodes with keys 45, 95, and 60 using the results of inOrderWalk
- void testDeleteNode45And95And60And50()
  - Verify the tree with 31 nodes is correct after deleting the nodes with keys 45, 95, 60, and 50 using the results of inOrderWalk
- void testCountForDuplicateNode()
  - Insert nodes with keys 20, 10, 30, 35, 5, 35, 15, 35, 25, and 35 to tree and verify the count for the node with key 35 is 4
- void testCountForDuplicateNodeWithDelete()
  - Insert nodes with keys 20, 10, 30, 35, 5, 35, 15, 35, 25, and 35 to tree, delete two nodes with key 35, and verify the count for the node with key 35 is 4