

CSE 100:

TERNARY TRIES

RED-BLACK TREES

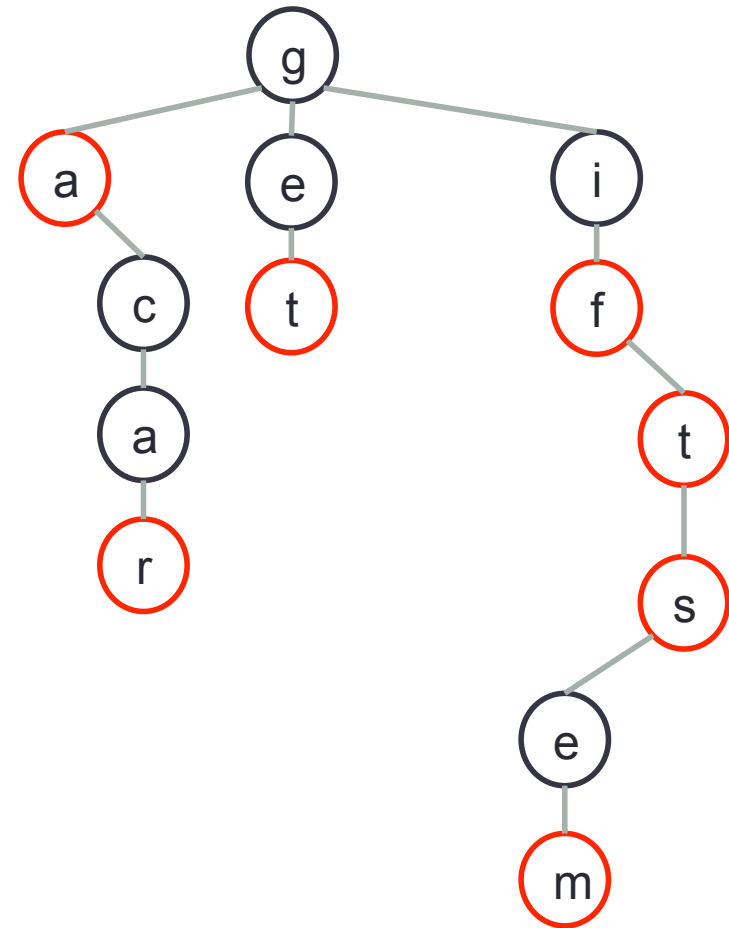
Ternary search trees to the rescue!

- Tries combine binary search trees with tries.
- Each node contains the following:
 - A key **digit** for search comparison
 - Three pointers:
 - **left** and **right**: for when the digit being considered is less than and greater than (respectively) the digit stored in the node (the BST part)
 - **middle**: for when the digit being considered is equal to the digit stored in the node (the trie part)
 - An **end** bit to indicate we've completed a key stored in the tree.

List all the words (strings) you can find in this TST

Are the following in the tree?
(A=yes, B=no)

- get
- if
- gif
- its
- gacar
- tsem



Draw the ternary tree for the following (in this order)

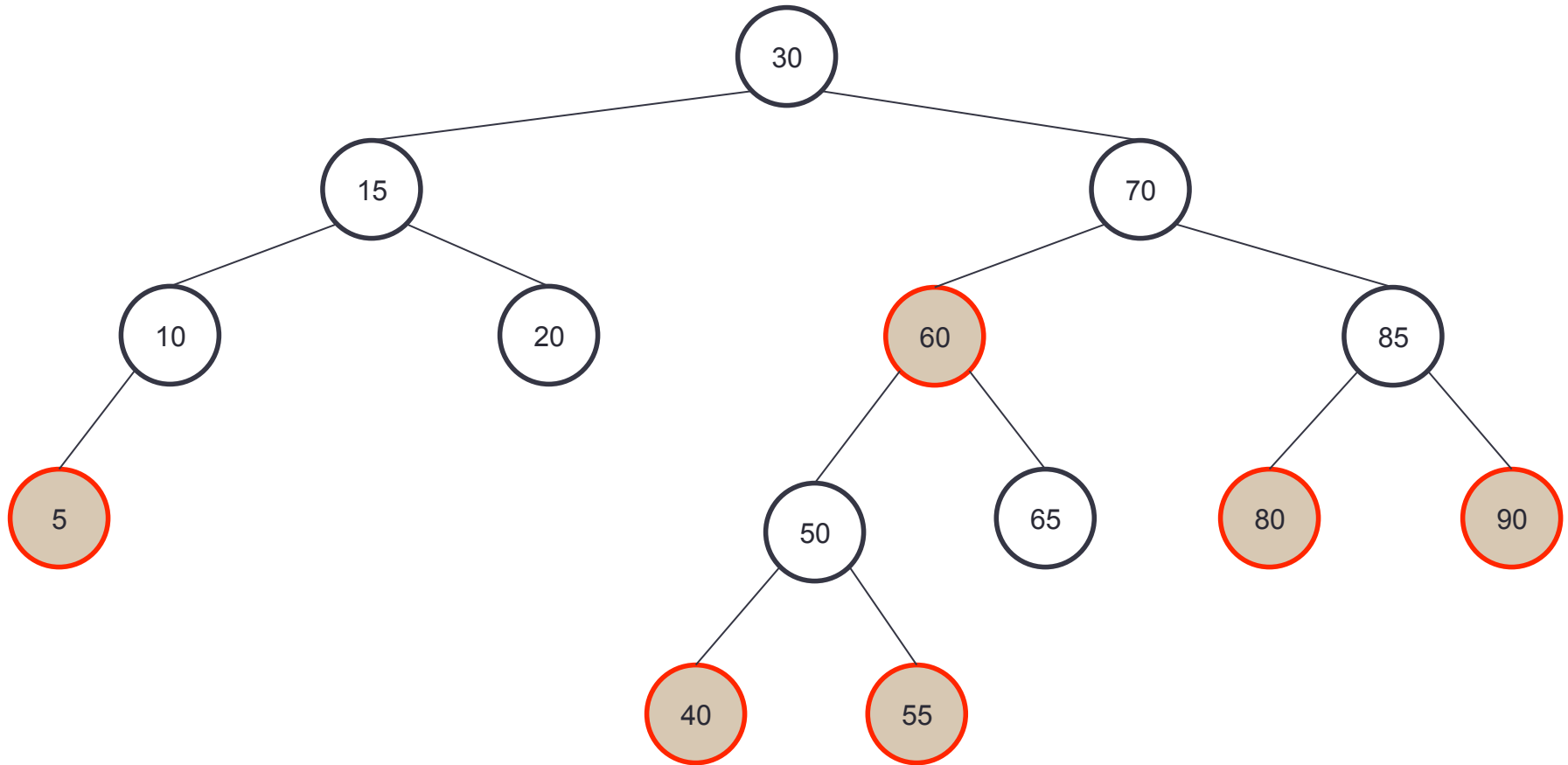
i
just
met
this
is
crazy
call
me
maybe

Does the structure of the tree depend on the order in which keys were inserted? A. Yes B. No

Algorithms for insert and find (in TSTs and MWTs)

- In your reading and/or in Paul Kube's slides

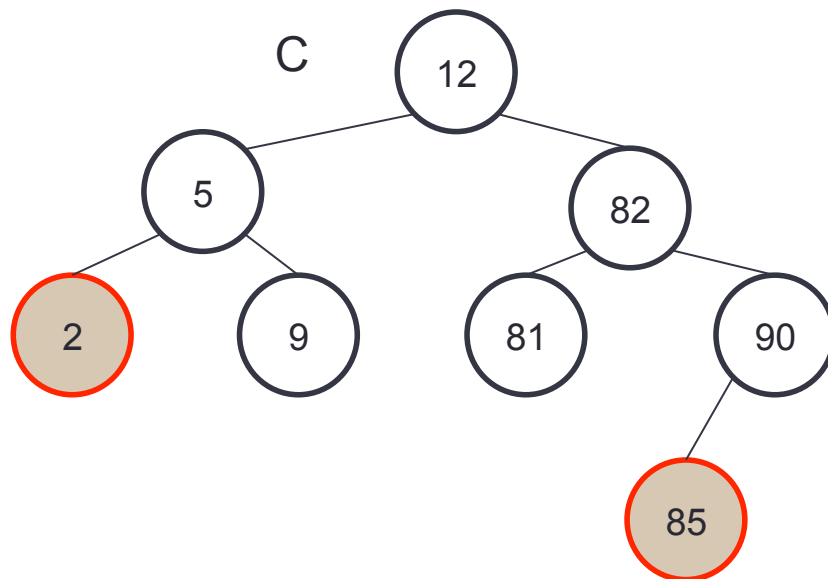
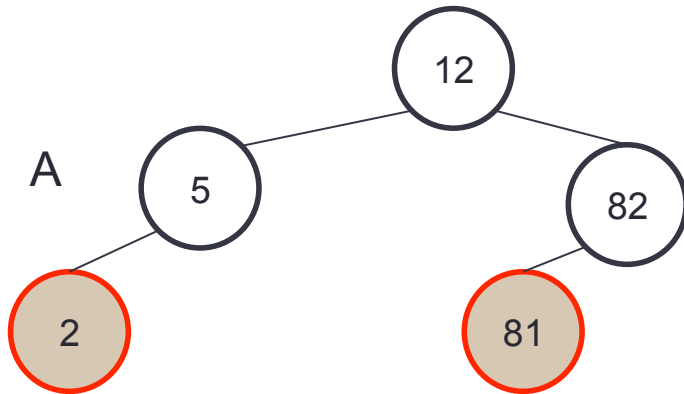
Red-Black Trees



1. Nodes are either red or black
2. Root is always black
3. If a node is red, all its children must be black
4. For every node X, every path from X to a null reference must contain the same number of black nodes

Which of the following are legal red-black trees?

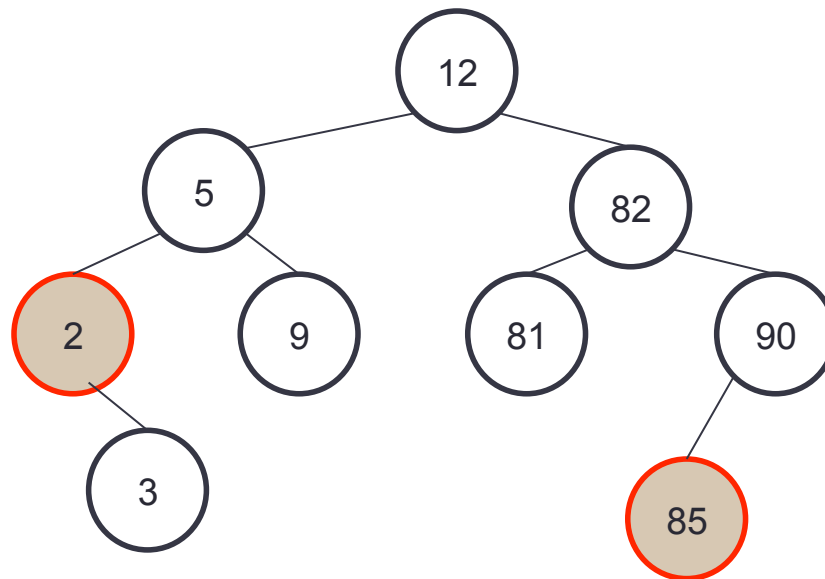
Red Black Trees



D. A&B

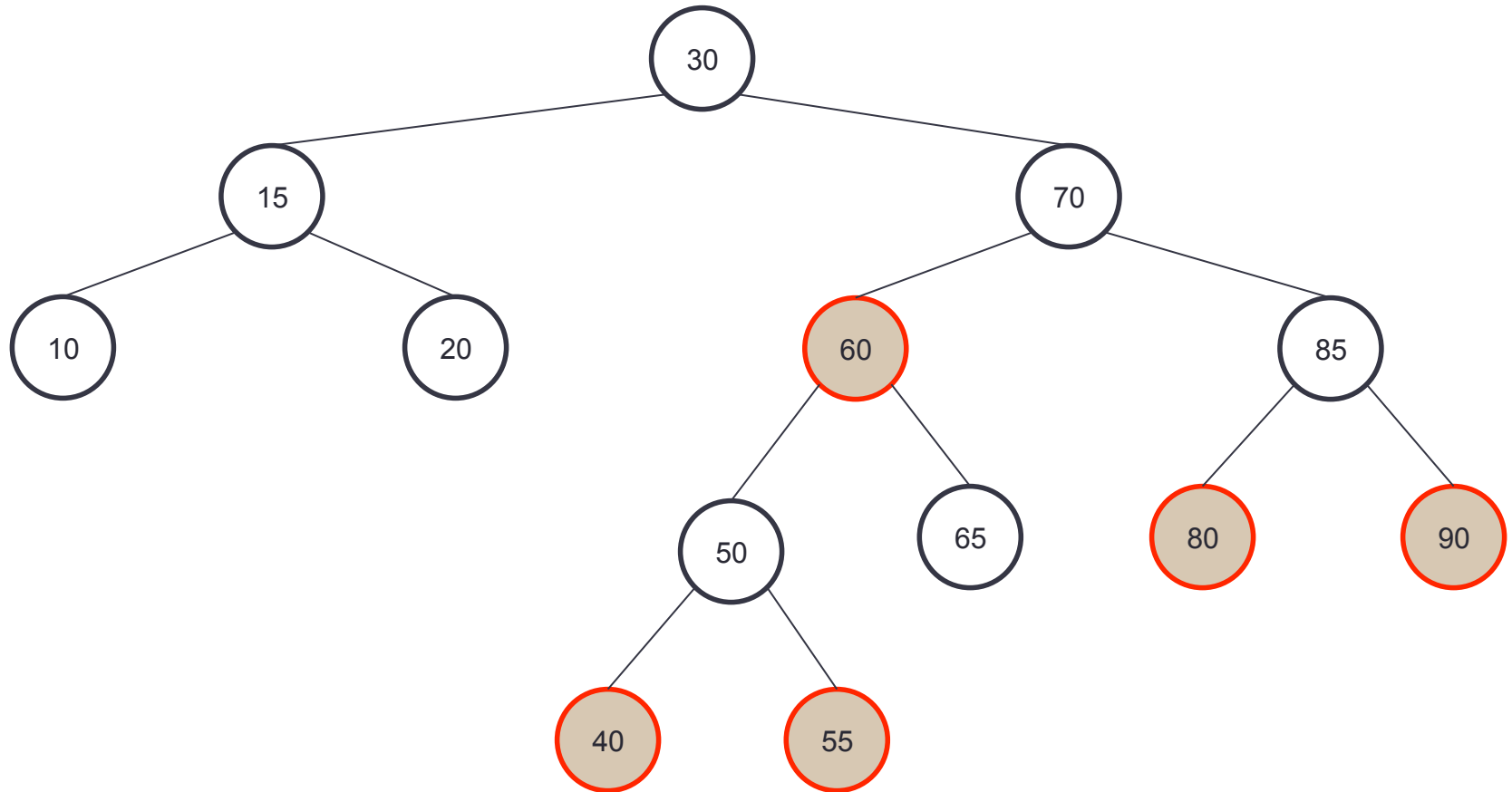
E. A&B&C

Is this a legal Red-Black tree?



- A. Yes
- B. No

Red-Black Trees

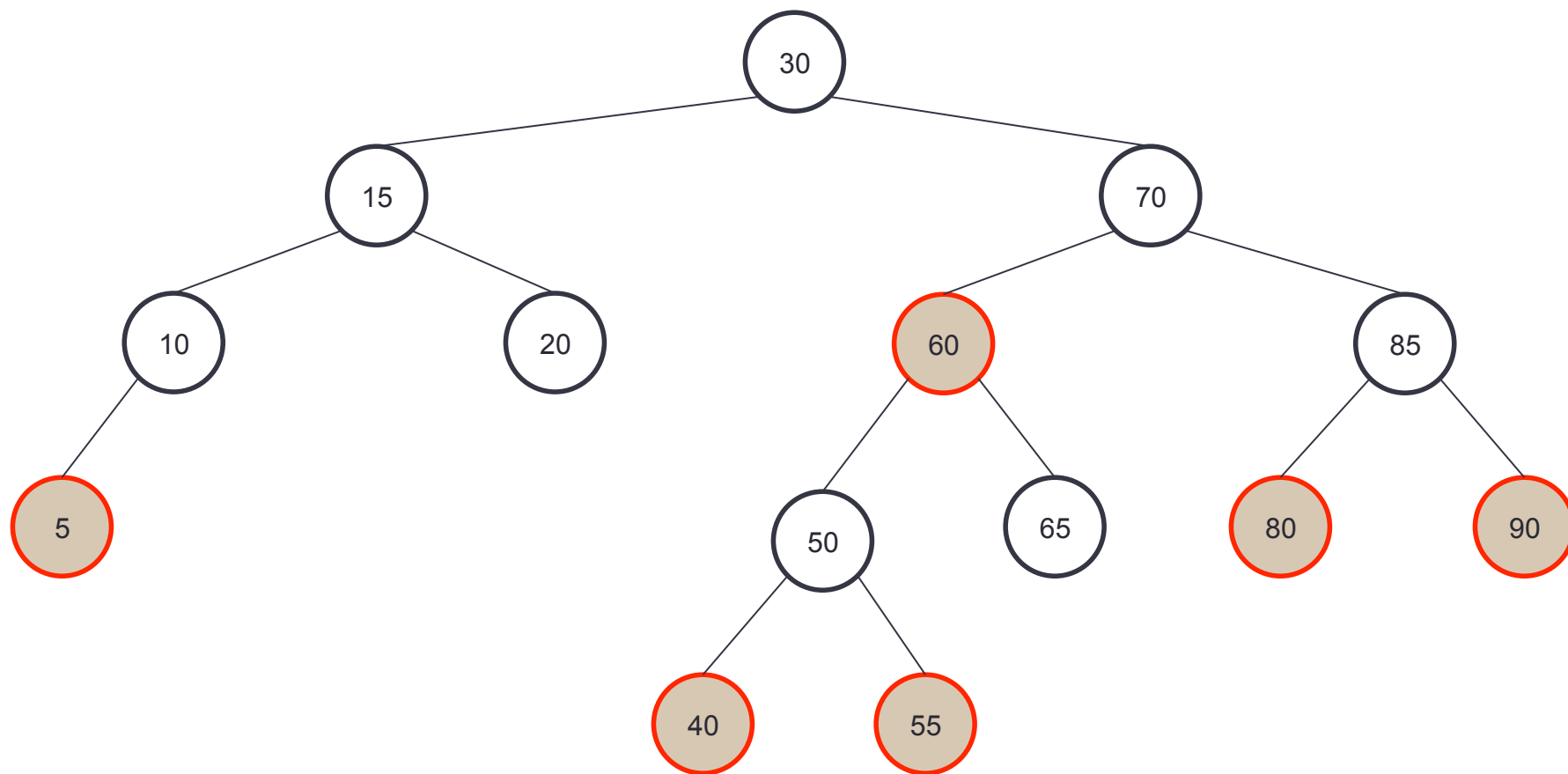


This is a red-black tree but not an AVL tree

Are all red black trees AVL trees?

No

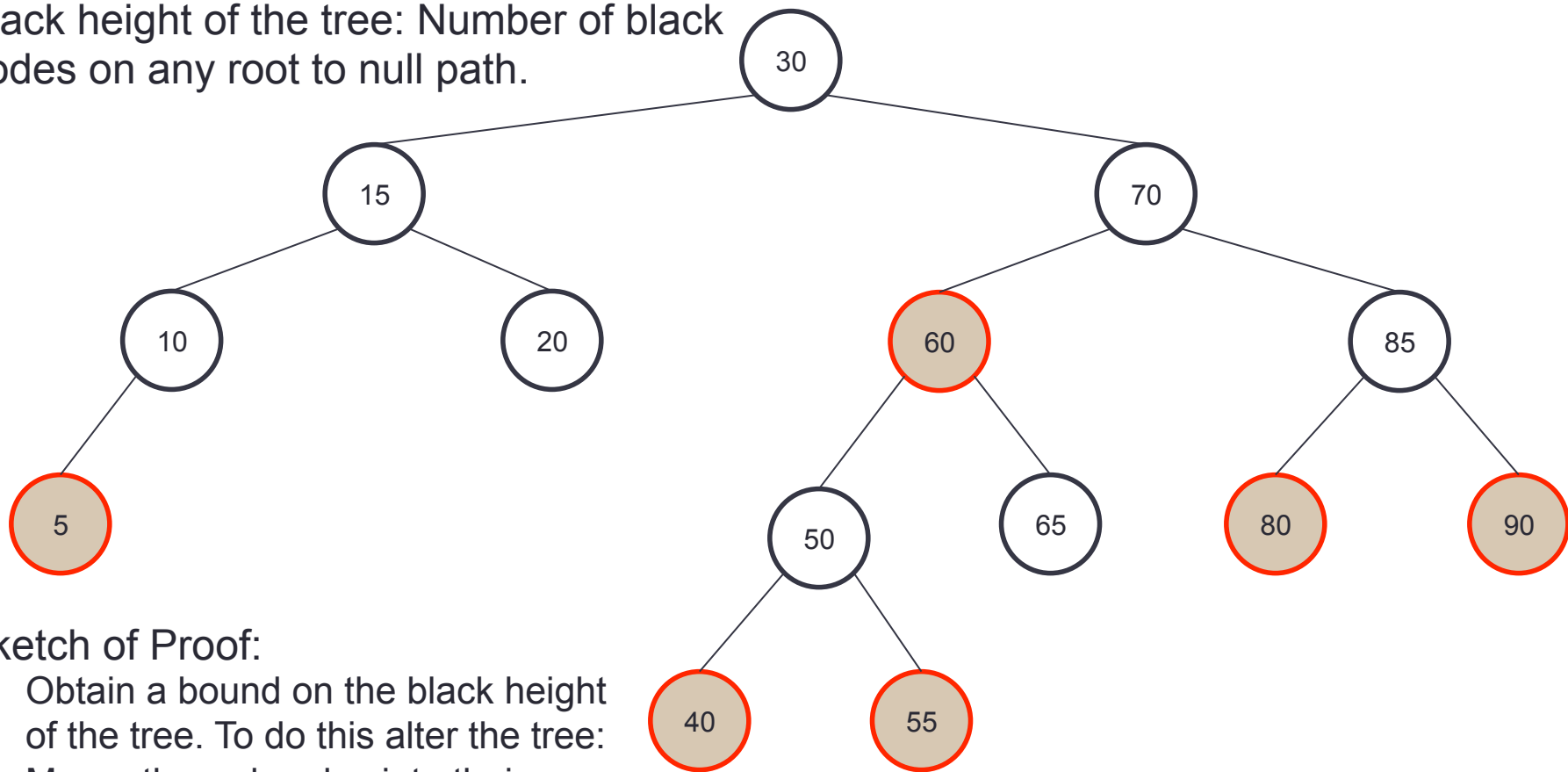
Red-Black trees have height $O(\log N)$



Proof idea: We will alter this tree so that it is in a form that allows us to make assertions about the height of the altered tree (which will NOT be a red-black tree). Then we will relate the height of the real tree to the height of the altered tree, in the worst case.

Height of a red-black tree is always $O(\log N)$.

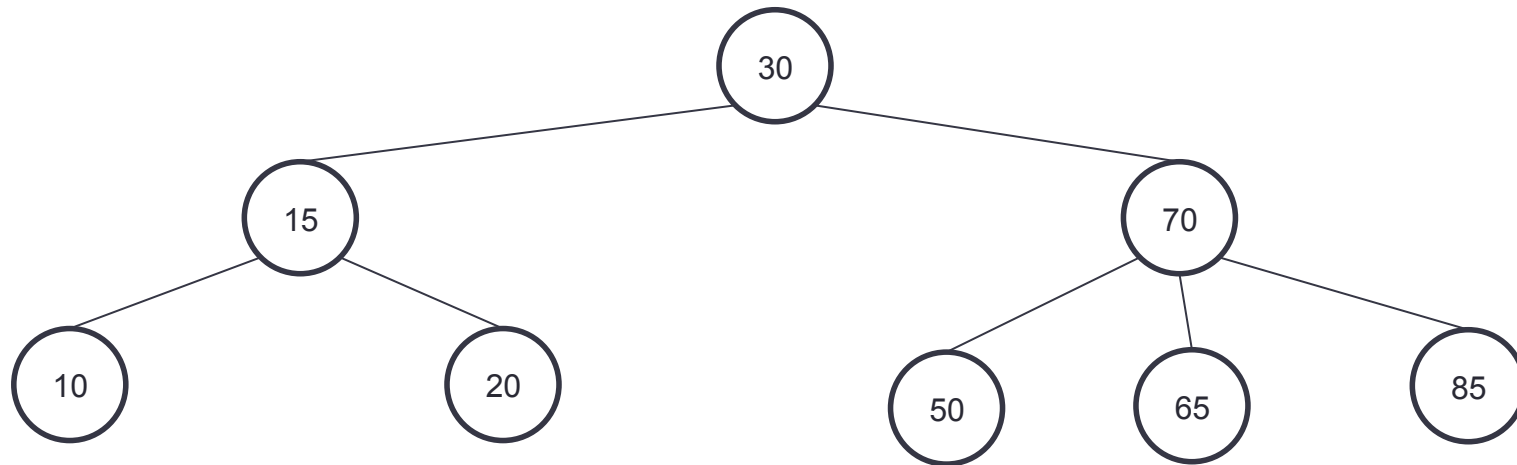
Black height of the tree: Number of black nodes on any root to null path.



Sketch of Proof:

1. Obtain a bound on the black height of the tree. To do this alter the tree: Merge the red nodes into their black parents

Red-nodes Merged into black parents



Sketch of Proof:

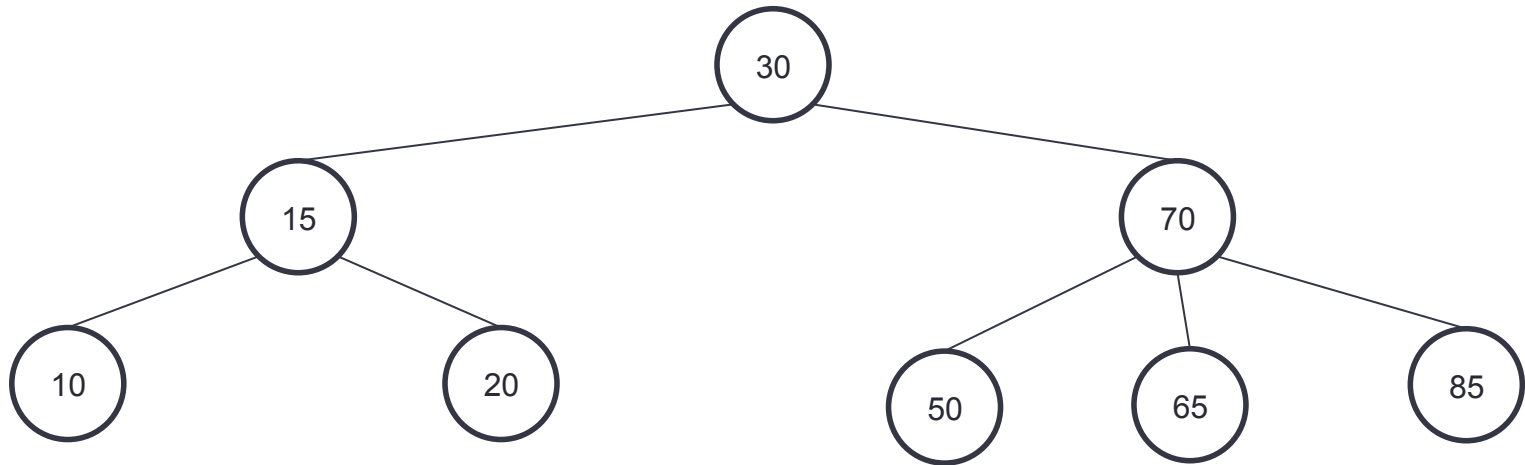
1. Obtain a bound on the **black height** of the tree. To do this alter the tree: Merge the red nodes into their black parents

Observe the following:

Leaves are all at the same level
Each internal node has at least 2 children

Black height of the tree: Number of black nodes from any root to null path, minus 1

What is the tightest upper bound on the height of this tree, where N_{black} is the number of nodes in *this* tree?

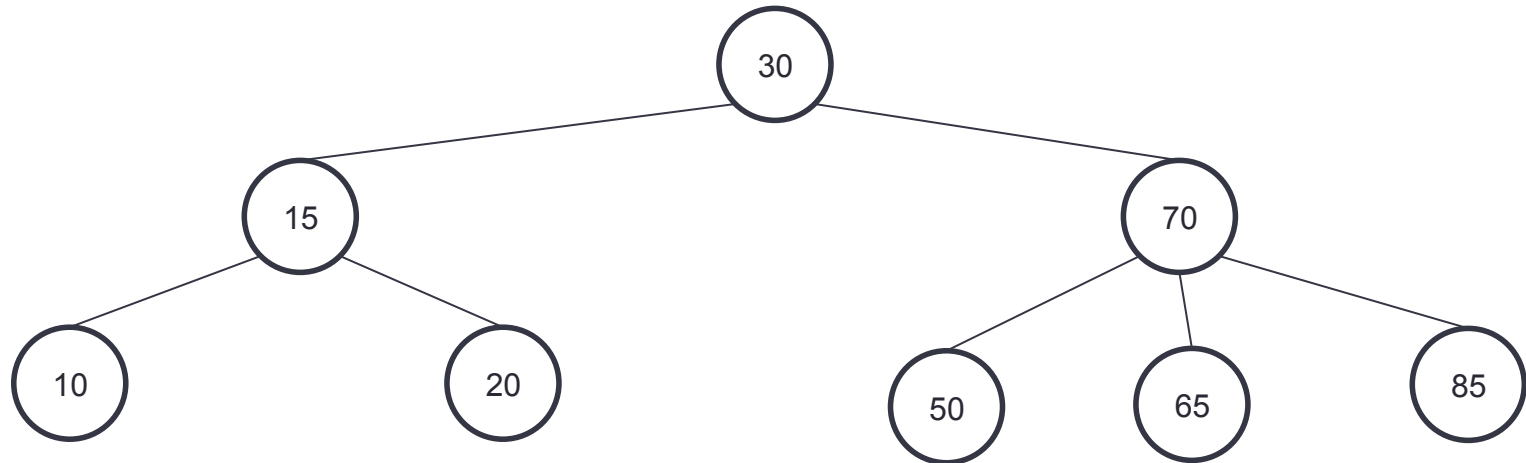


A. 2

B. $\log_2(N_{\text{black}} + 1)$

C. N_{black}

Red-Black invariants imply balance



Leaves are all at the same level
Each internal node has at least 2 children

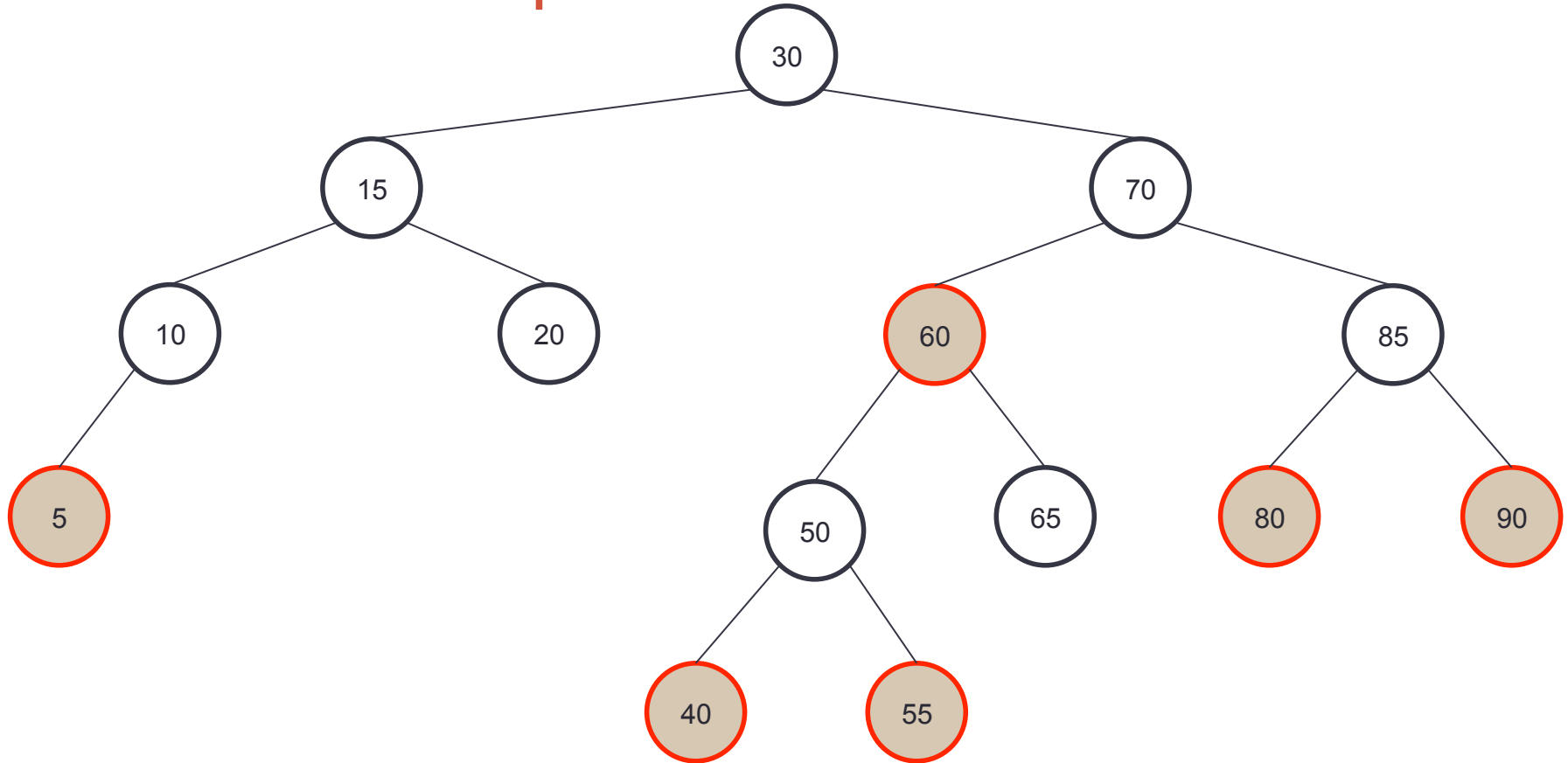


Height is at most $\log_2(N+1)$

If you put the red nodes back, this can increase the height by at most a factor of 2

Height is at most $2 \cdot \log_2(N+1)$

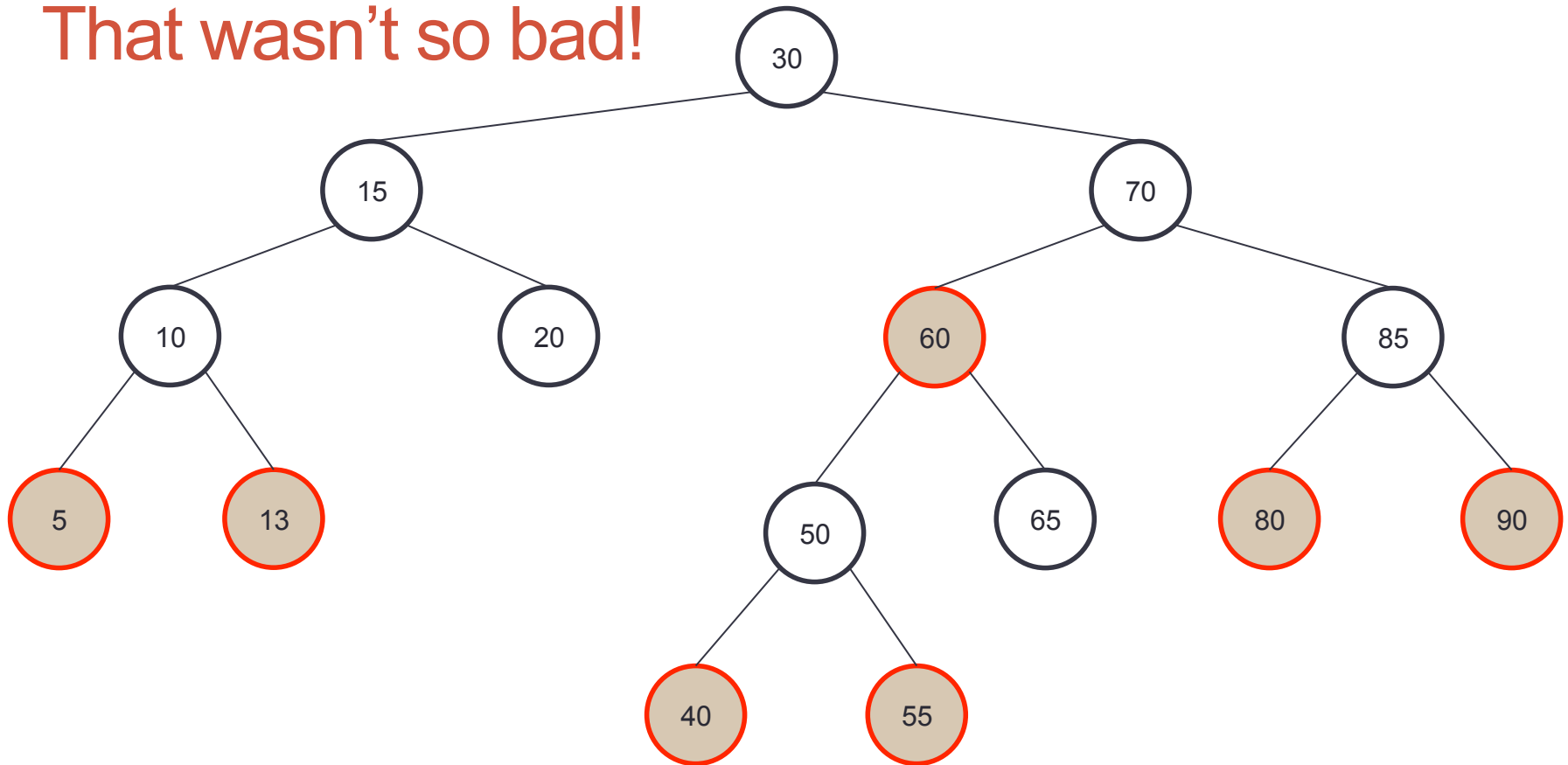
Now for the fun part... insertions



Non-root insertions will always be red

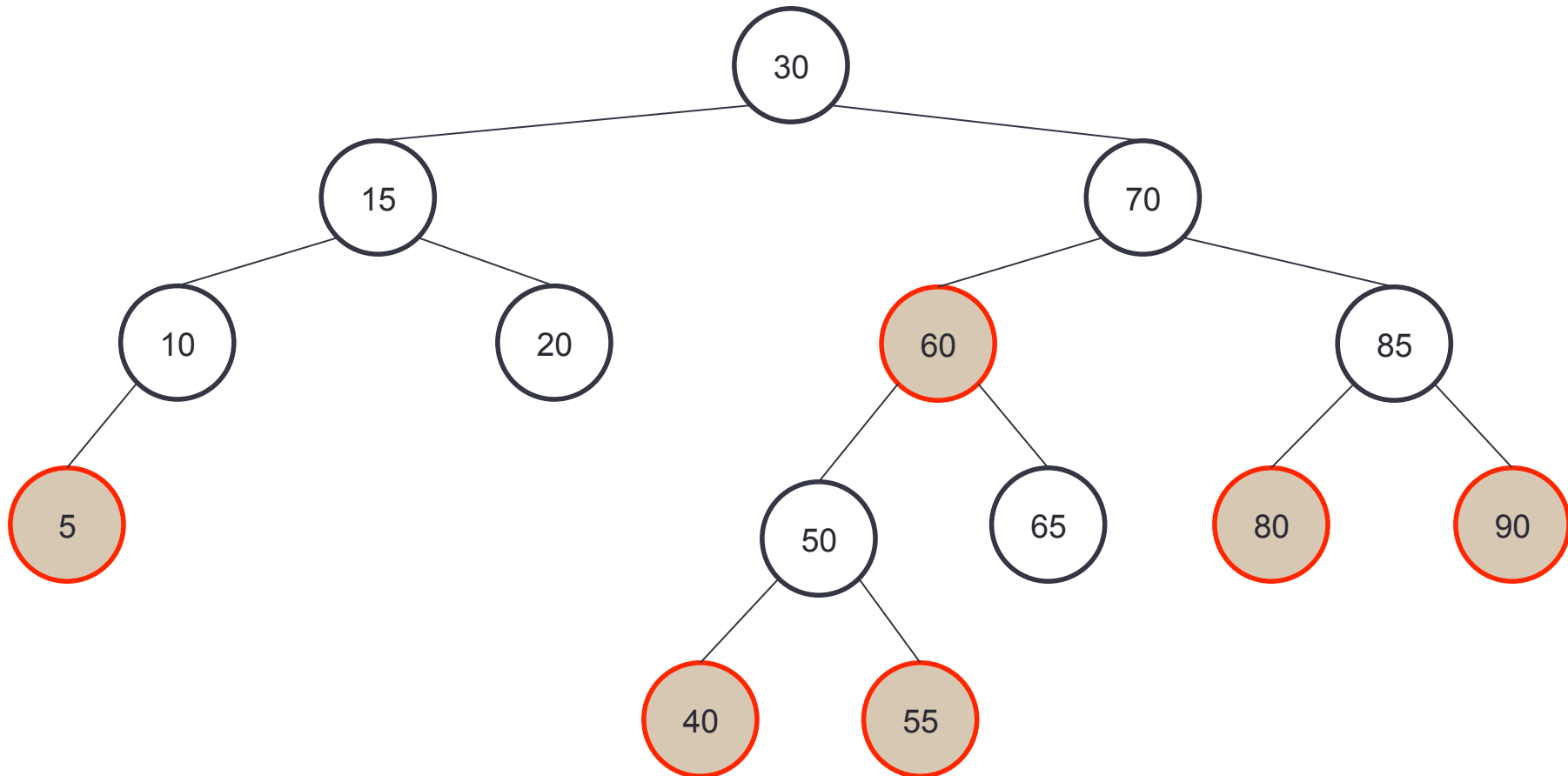
Try inserting 13

That wasn't so bad!



Case 0: Parent was black. Insert new leaf node (red) and you're done.

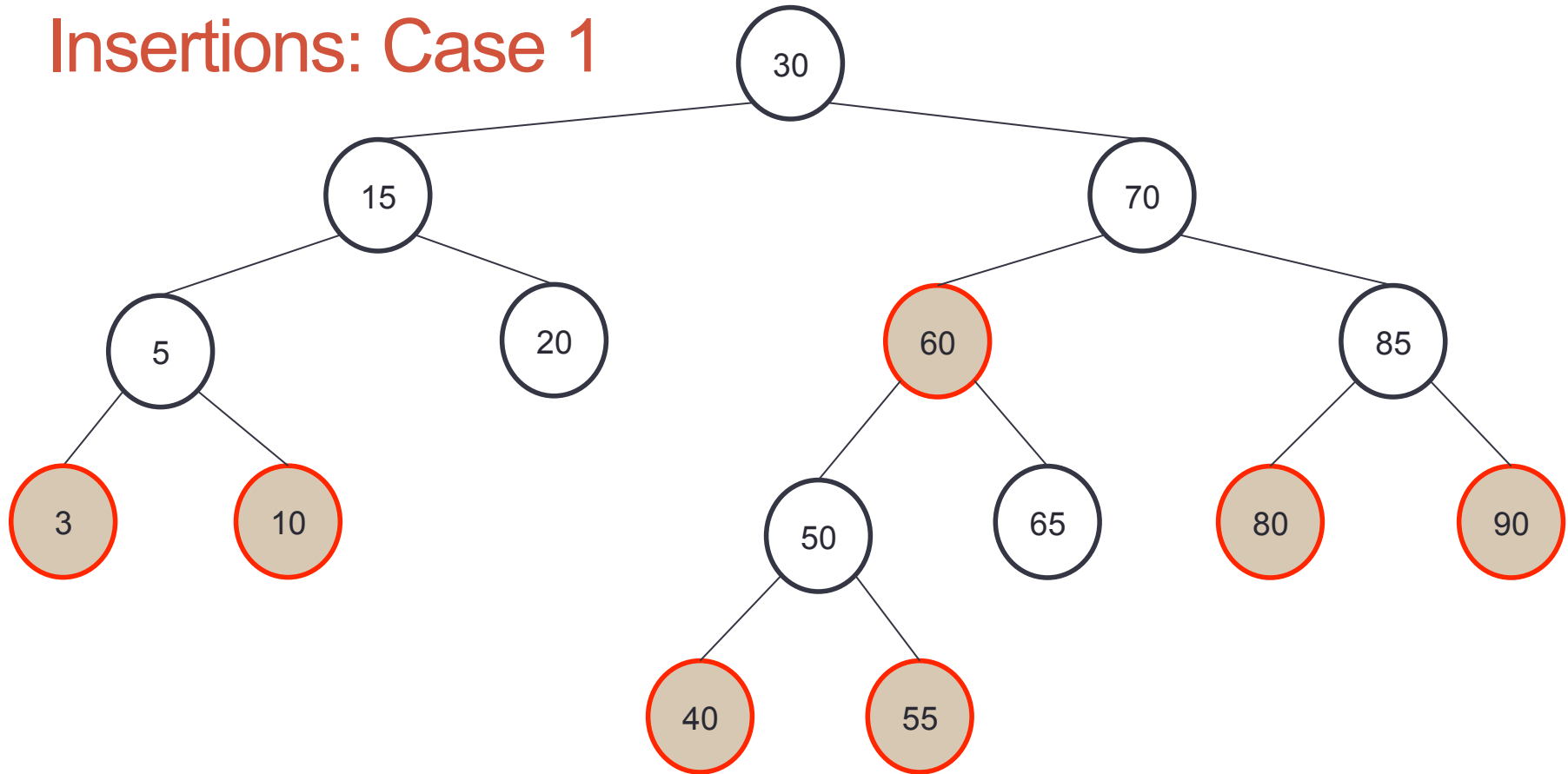
Insertions: More complicated case



Try inserting 3

Case 1: Parent of leaf is red, parent is left child of grandparent,
leaf is left child of parent, (& sibling of parent is black)

Insertions: Case 1



Right AVL rotation, and recolor