Opening Questions

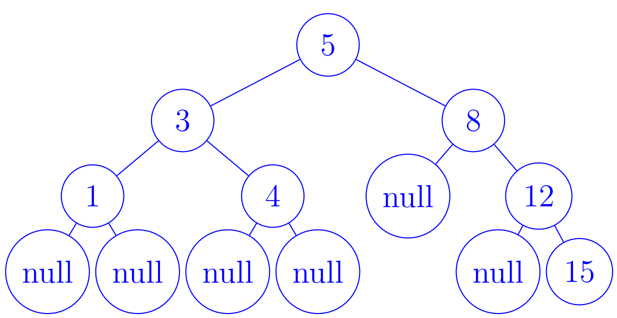
1. What is the main problem with Binary Search Trees that Red-Black Trees correct? Explain briefly (2–3 sentences) how Red-Black Trees correct this problem with Binary Search Trees. The biggest problem is that the tree may be a straight line that would cause the height to be $O(n)$. Red-Black trees maintain this balance by keeping track of the colors and allows the tree to be better balanced using a specific set of rules.

2. For the balanced binary search trees, why is it important that we can show that a rotation at a node is $O(1)$ (i.e. not dependent on the size of the BST) We want to make fixes that are not dependent on the size of a binary tree.

Red-Black Trees

Red-Black Properties

1. Every node is colored either red or black
2. The root is black
3. Every null pointer descending from a leaf is considered to be a null black leaf node
4. If a node is red, then both of its children are black
5. For each node, all paths from the node to descendant leaves contain the same number of black nodes (black height)



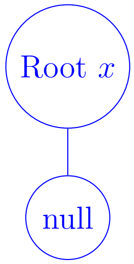
black-height of a node $bh(x)$ – The number of black nodes on any path from, but not including, a node x down to a null black leaf node. (Black height counts black nodes including null leaves, height doesn't count null nodes.)

1. If a node x has $bh(x) = 3$, what is its largest and smallest possible height (distance to the farthest leaf) in the BST? 2–5

2. Prove using induction and red-black tree properties. A red-black tree with n internal nodes (nodes with values), none null nodes (n key values) has height at most $2\lg(n+1)$

Part A) First show the sub-tree rooted at node x has at least $2^{bh(x)} - 1$ internal nodes. Use induction. Relating the black height of a node to how many values at least must be in that subtree.

Base Case: $bh(x) = 0$



This tree has $bh = 1$, with the only internal node being the root.

$$\begin{aligned} 2^{\text{bh(x)}} - 1 &= 2^0 - 1 \\ &= 1 - 1 \\ &= 0 \end{aligned}$$

Induction Step: Assuming at least $2^{bh(x)} - 1$ keys in some subtree rooted at x. If a node is red, $bh((parent)) = bh((child))$ else a node is black, then the $bh((parent)) = bh((child)) + 1$.

$$\left(2^{bh(x)} - 1\right)\left(2^{bh(x)} - 1\right) + 1 = 2 \times \left(2^{bh(x)} - 1\right) + 1$$

Part B) Let h be height of a Red-Black Tree, by property four, at least half of the nodes on path from root to leaf are black

$$bh(root) \geq \frac{h}{2}$$

Use that and Part A to show

$$h \leq 2\log(n+1)$$

$bh(root) = k$ at least $2^k - 1$.

$$\begin{aligned} 2^{bh(root)} - 1 &\leq n \\ 2^{h/2} - 1 &\leq n \\ 2^{h/2} &\leq n + 1 \\ \frac{h}{2} &\leq \lg(n+1) \\ h &\leq 2\lg(n+1) \\ h &= O(\lg(n+1)) \end{aligned}$$

3. Which BST operations change for a red-black tree and which do not change? What do the operations that change need to be aware of and why?

- Search – Won't change, you'd ignore the colors
- Insert – Update
- Delete – Update
- Predecessor – Won't change
- Successor – Won't change
- Minimum – Won't change
- Maximum – Won't change
- Rotations – Update

Red-Black Tree Insert

Similar to BST Insert, assume we start with a valid red-black tree.

1. Locate leaf position to insert new node
2. Color new node red and create 2 new black null leafs below newly inserted red node
3. If parent of new insert was __________ (fill in the blank, black or red), then done. ELSE procedure to recolor nodes and perform rotations to maintain red-black-properties.

There are three cases if Red-Black Property #4 when insert a red node Z (or changed color of a node to red) and its parent is also red.

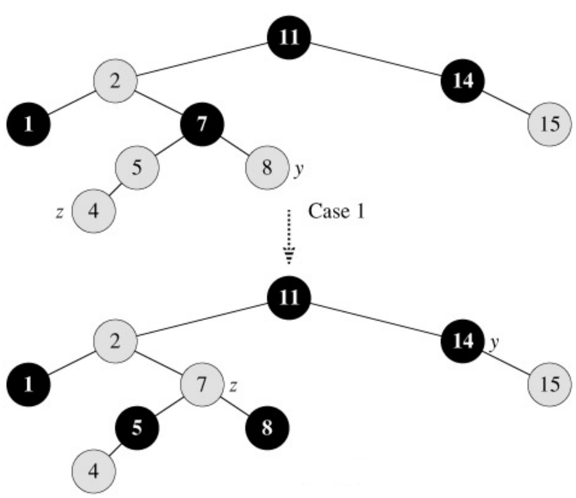


Figure 11.1: Broken Red-Black Property #4: Case #1

Node Z (red) is a left or right child and its parent is red and its uncle is red (the children of nodes value 4 5 8 must all be black, or null black).
Swap the colors of a parent node and both its children, preserving the black height property at all nodes.

- Change Z's parent and uncle to black
- Change Z's grandparent to red
- No effect on black height on any node
- Z's grandparent is now Z and check again for property #4 (two reds in a row) still broken at new node Z (possible non-terminal case, need loop or recursion)

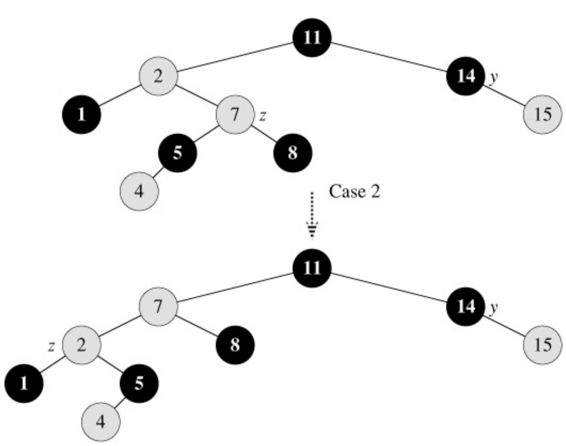


Figure 11.2: Broken Red-Black Property #4: Case #2

Node Z is a right child and its parent is red and its uncle is NOT red.
Do a single rotation, preserving the black height property at all nodes.

- Rotate left on parent of Z.
- Re-label old parent of Z as Z and continue to case #3.

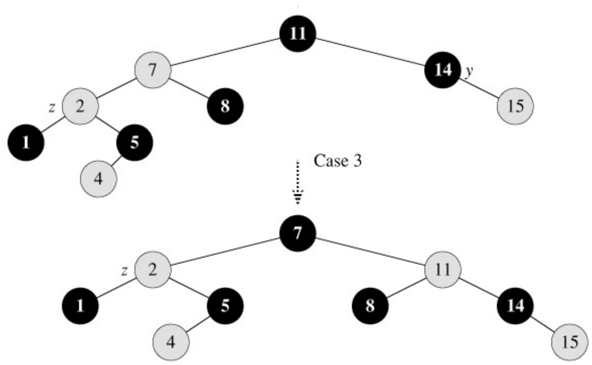


Figure 11.3: Broken Red-Black Property #4: Case #3

Node Z is a left child and its parent is red and its uncle is NOT red.
Do a single rotation and swap the colors of a parent node and both its children, preserving the black height property at all nodes.

- Rotate right on grandparent of Z
- Color old parent of Z black
- Color old grandparent of Z red