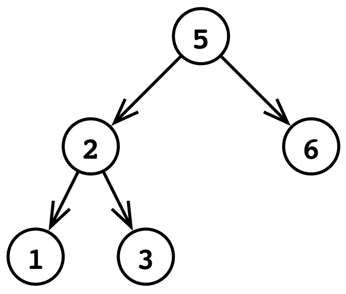
CMPUT 275 — Tangible Computing II Winter 2018
AVL Trees

Problems

1. Insert key 4 into the following AVL tree. Don't forget to perform rotations to fix the AVL property after you create the new node (one of the four cases from the slides).



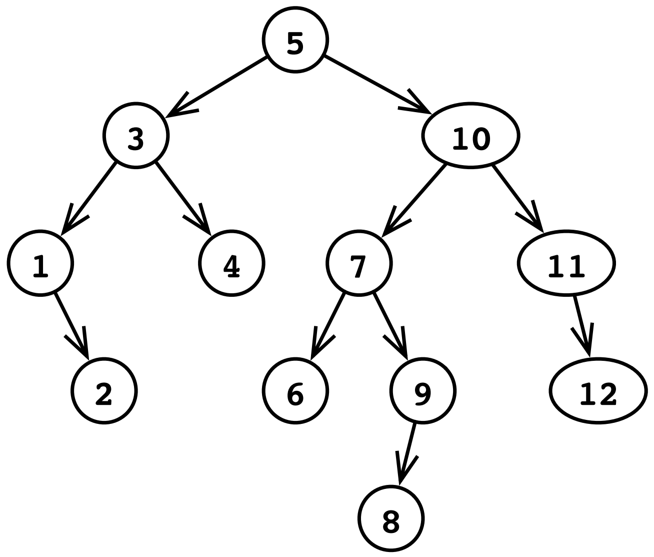
2. The following **red/black** property is another way to keep a binary search tree balanced. Suppose T is a binary search tree and every node is coloured either red or black. We call such a tree a **red/black** tree if the following properties hold:

   - If a node is red then both children are black.
   - Each leaf node (i.e. has no children) is black.
   - For every node v, all paths from v to some leaf node in the subtree under v contains the same number of black nodes. Call this value $b(v)$.

   Feel free to look up pictures online. We won't discuss how to maintain the red/black property in class: regard it as an alternative to AVL trees you can look up if you are curious. The point of this question is to give another example of how a somewhat subtle property can be used to guarantee useful properties in data structures.

   Argue that a red/black tree has height $O(\log n)$.
   Hint: Show how the first two properties imply the height is at most $2 \cdot b(v) - 1$ and the last two properties imply the number of nodes in the subtree under any node v is at least $2^{b(v)} - 1$.

3. Now remove key 4 into the following AVL tree. Don't forget to perform rotations to fix the AVL property after you create the new node. Recall, the rule is to repeatedly pick the deepest node where the AVL height property is violated and fix it using rotations.



5
3
10
1
4
7
11
2
6
9
12
8