Red-Black Tree Task Descriptions

Red-Black-Tree-Definition:

A red-black tree has the same restrictions as binary search trees:
- A Binary-Search-Tree's node has at most two children.
- The left child and all its descendants have lower values compared to the parent's value.
- The right child and all its descendants have higher or equal values than the parent's value.

Additionally, it must fulfill the following red-black tree properties:
- Every node in a red-black tree is either red or black
- The root must always be black
- Every leaf node is black (**please view the important clarification at the end of the document!)**
- If a node is red, then both its children must be black
- For every single node, the path from that node to all the leaf nodes must contain the exact same amount of block nodes on the way.

Red-Black-Tree-Tasks

1. Red-Black Tree Insertion

   The student solving the task is asked to add the given values to a given Red-Black Tree. If none is provided, then the values must be inserted into an initially empty red-black tree. The values must be inserted in the given order following the rules of a red-black tree until the created red-black tree includes all values. After inserting a new value, rebalancing may have to be performed.

2. Red-Black Tree Repair

   The student will receive a red-black tree, but it will be incorrect, meaning that at least one of the red-black tree properties is violated. The task is to fix it. What went wrong? Another student has previously inserted a value (you do not know which one) and either completely forgot to do any rebalancing (just inserted the node and that's it) or stopped rebalancing before it is complete (before all red-black tree properties are fulfilled).

   The task is to complete the rebalancing and therefore correct/finish the previous student's task.

   Assumptions to make the task clearer follow on the next page.

You may assume this for Red-Black Tree Repair:

- The received tree will always be incorrect. At least one red-black property will be violated. More specifically, there will either be a red-red violation or a red-root violation.
- Each singular rebalancing step will be executed atomically. You can imagine it like this: Pretend that on each call of rebalance() the program can halt and stop rebalancing. But it cannot stop in the middle of a rebalance() call.
- E.g. in one example, four total calls to rebalance() are necessary, but rebalance() was only executed fully twice before crashing.

Important clarification:

The nodes visible in the red-black trees are never the real leaf nodes.

Every single node, which has one or both children missing must be thought of having a black “NIL” node as one or both children.

Therefore, implicitly, rule 3 (every leaf node is black) will always be fulfilled.

Though, when commenting about black height, the NIL nodes are not counted towards the black height. So, the black height of the trees below would be two, since each path to the leaves passed through the root (black) and exactly one other black node apart from NIL.

- (10) → (3), (10) → (7), (10) → (15)