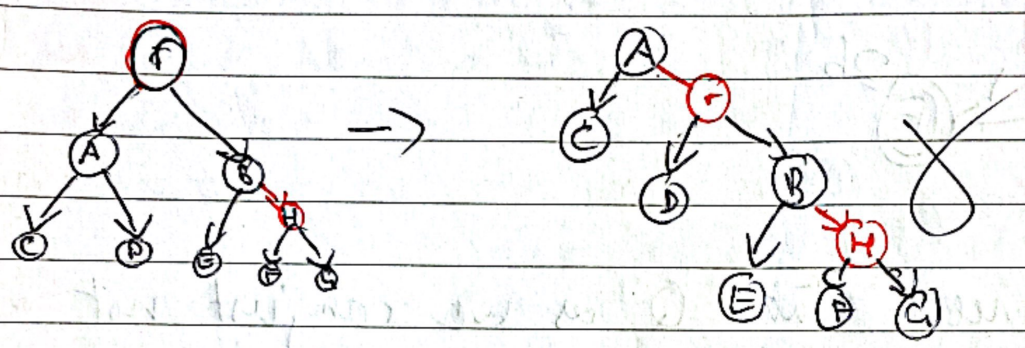
Algorithms Supo 2 work

1. The root must be black
   - If a node is red then both children black
   - From a node, all paths from that node to descendent leaves contain the same number of black nodes.

2.



So we want a right rotation of rA edge.

We can just change the root colour from red to black.

```
let root_colour = function
    | Leaf -> Leaf
    | Node (Red, a, t1, t2) -> Node (Black, a, t1, t2)
    | t -> t;
```

Assuming we get passed a red-black tree that is well formed except for having a red root we'll show root_colour will return a well-formed r-b tree with the same data but a black root.

We'll show all the rb tree invariants remain after the function.

2. Since a r-b tree with a red root is passed it will go to the second pattern match and return the same tree with no changes except the node colour will now be Black.
Let's check each r-b tree property.

i) Every node is red or black.
This still holds since it held before and we just changed a red to a black

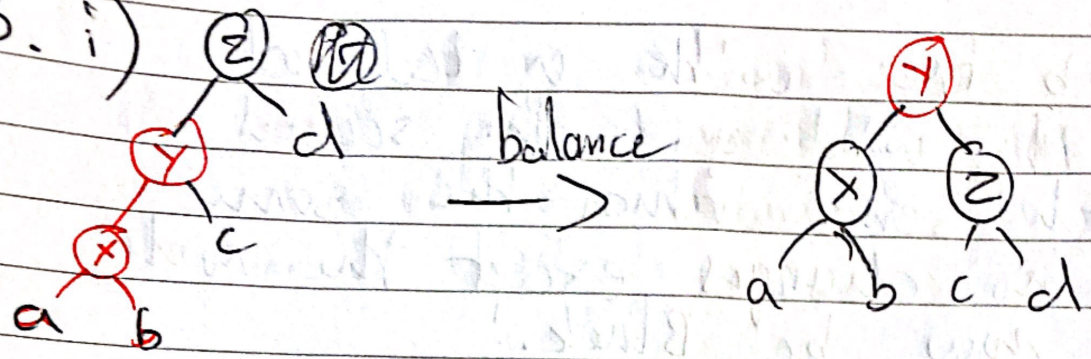
ii) The root is black.
Yes because we just made it black

iii) All leaves are black.
We didn't change anything about any leaf so this still holds.

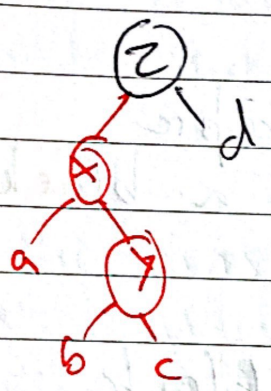
iv) If a node is red, both children are black.
We haven't added any new red nodes or touched the children of any red node so this still holds.

v) For each node, all paths from that node to descendent leaves contain the same number of black nodes.
The only paths we've changed are ones starting at the root node. We've added 1 black to all these paths so they still have the equal number of blacks.

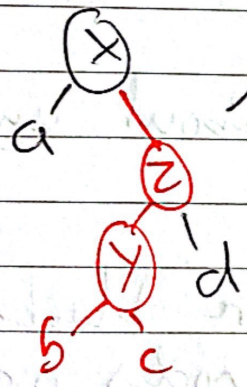
3. i)



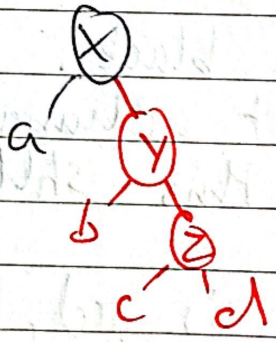
ii)



iii)



iv)



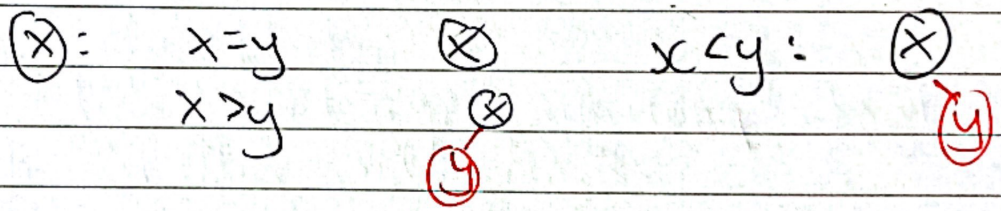
Balance fixes the problem of a red node having a red child node.

4. Firstly depth = 0.
We have a root with two leaves.

we call ~~insert x Node~~
insert x Node (Black, y, leaf, leaf)
-> ins Node (Black, y, leaf, leaf)

if x=y it's already in tree so it returns properly
if x>y we insert x into left leaf
if x<y we insert x into right leaf

When we insert into a leaf we make the new leaf a red node. The cases are as follows.



I realised the above is for depth 1. Depth 0 is just a leaf so we just make a red node with x and then make it black in the bottom pattern match.

For depth 2 it will ask either insert x as a child of a black node, in which case nothing needs done, or of a red node. If it's a red node then we have a red node as the child of a red node. However since we call balance this will fix this issue. It may create another red-red situation above but since balance is called on every recursion it will fix it up to potentially a red root.

4. If it gives a red root we see that the last pattern match makes it black and in (2) we showed this keeps the tree well formed.

Identical logic applies to any depth ≥ 2 so insert is correct for every well-formed red-black tree.

5. The ins s function always returns a node in every case so we'll never pattern match to a leaf.