Data Structures - Assignment 4

Name - Gaurang Shukla
Rollno - CS2415
Course - Mtech CS

Q1. Prove or disprove that all height-balanced trees are red-black trees.

First let us define the Height Balanced and Red Black Trees.

Height Balanced Tree : A height-balanced binary tree is defined as a binary tree in which the height of the left and the right subtree of any node differ by not more than

Red Black Tree : A red black tree should satisfy these five properties:

A. A node is either red or black.
B. The root is black. This rule is sometimes omitted. Since the root can always be changed from red to black, but not necessarily vice versa,
C. All leaves (NULL) are black.
D. If a node is red, then both its children are black.
E. Every path from a given node to any of its descendant NULL nodes contains the same number of black nodes.

Algorithm :

```
color_black(x):
  x.color = black;
  if x is a node:
    color_children(x.left, x.right)
```

```
color_red(x):  // height(x) must be even
  x.color = red
  color_children(x.left, x.right)  // x will always be a node

color_children(a,b):
  if height(a) < height(b) or height(a) is odd:
    color_black(a)
  else:
    color_red(a)
  if height(b) < height(a) or height(b) is odd:
    color_black(b)
  else:
    color_red(b)
```

Proof :

Lets see if each property of red black is satisfied or not using above algorithm

A. We will use depth first order to traverse all the nodes in the tree which will ensure that every node is coloured either red or black.
B. Obviously, the root node is coloured as black.
C. NULL nodes have height 1, which is odd. So they will always be coloured as black.
D. Children of red nodes can have either odd height or shorter than sibling, in both cases they will be coloured as black.

To prove that the Black node height is the same from every node to NULL node, we will use induction.

Use induction on n>=1 to prove:

for odd height = 2*n-1,

color_black() creates a red-black tree, with depth n

for even height = 2*n,

color_red() sets all paths to depth n

color_black() creates a red-black tree with depth n+1

Base case, for n = 1:

for odd height = 1, the tree is a leaf;

color_black() sets the leaf to black; the sole path has depth 1,

for even height = 2, the root is a node, and both children are leaves, marked black as above;

color_red() sets node to red; both paths have depth 1

color_black() sets node to black; both paths have depth 2

The induction step is where we use the AVL invariant: sibling trees can differ in height by at most 1. For a node with a given height:

subcase A: both subtrees are (h-1)

subcase B: one subtree is (h-1), and the other is (h-2)

Induction step: given the hypothesis is true for n, show that it holds for n+1:

For odd height = 2*(n+1)-1 = 2*n+1,

subcase A: both subtrees have even height 2*n

color_children() calls color_red() for both children, via induction hypothesis, both children have depth n for parent, color_black() adds a black node, for depth n+1

subcase B: subtrees have heights 2*n and 2*n-1

color_children() calls color_red() and color_black(), resp;

for even height 2*n, color_red() yields depth n (induction hyp.)

for odd height 2*n-1, color_black() yields depth n (induction hyp.)

for parent, color_black() adds a black node, for depth n+1

For even height = 2*(n+1) = 2*n + 2

subcase A: both subtrees have odd height 2*n+1 = 2*(n+1)-1

color_children() calls color_black() for both children, for depth n+1

from odd height case above, both children have depth n+1

for parent, color_red() adds a red node, for unchanged depth n+1

for parent, color_black() adds a black node, for depth n+2

subcase B: subtrees have heights 2*n+1 = 2*(n+1)-1 and 2*n

color_children() calls color_black() for both children, for depth n+1

for odd height 2*n+1, color_black() yields depth n+1 (see above)

for even height 2*n, color_black() yields depth n+1 (induction hyp.)

for parent, color_red() adds a red node, for depth n+1

for parent, color_black() adds a black node, for depth n+2 = (n+1)+1