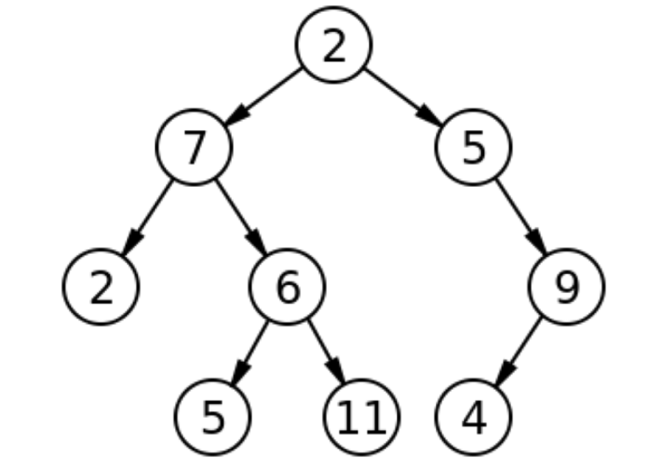
|  |
| --- |
| **CHAPTER 6 HOMEWORK** |

**PROBLEM 1:**

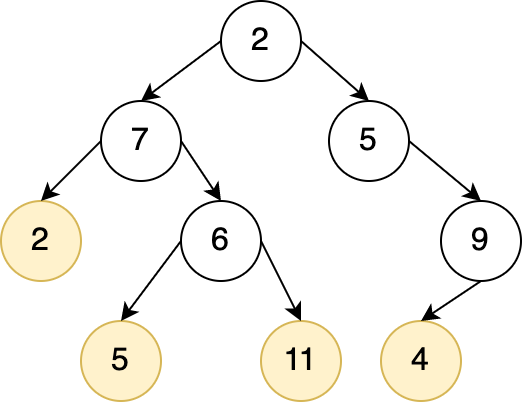
****

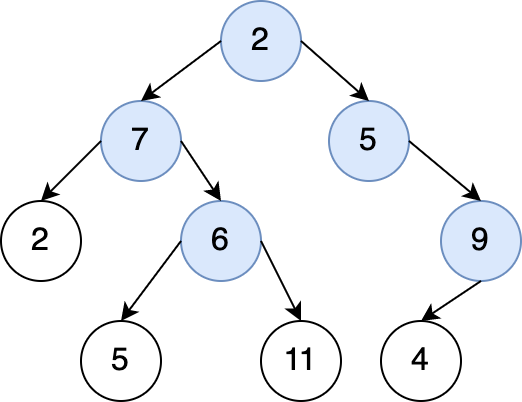
1. In a **full binary tree** all nodes have either 0 or 2 children. Both types of nodes can appear at all levels in the tree.

In a **complete binary tree**all levels except the last are completely filled, and in the last level all nodes are to the left as much as possible.

- T is not a complete binary tree, because level 1 aren’t completely filled.

- T is not a full binary tree, because node 5 and node 9 has only one children.



1. The degree of T is 2.
2. There are 4 leaves in T. Left nodes:
3.  There are 5 internal nodes in T. Internal nodes:
4. T is not a binary search tree.

Because 7 is a node in the left subtree of 2 but 7 > 2; 5 is a node in the right subtree of 2 but 5 < 2

1. The height of T is 3
2. The nodes at level 1 : 7, 5

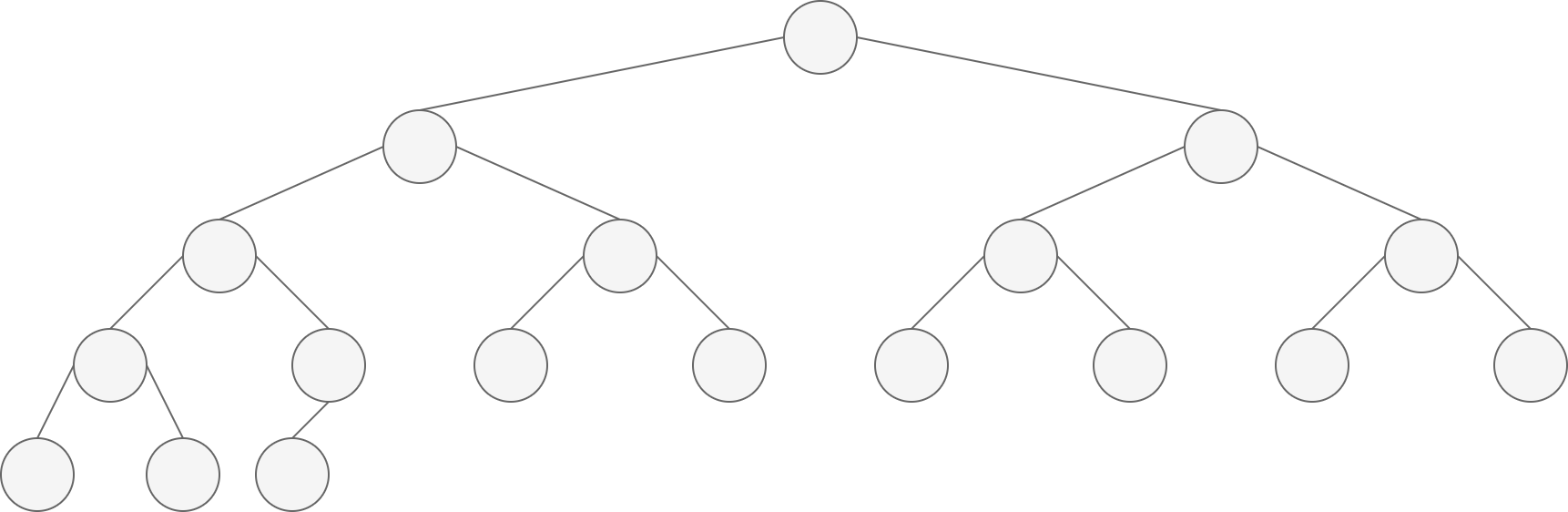
The nodes at level 2: 2, 6, 9

1. The path from root node to node 11: 2 7 6 11

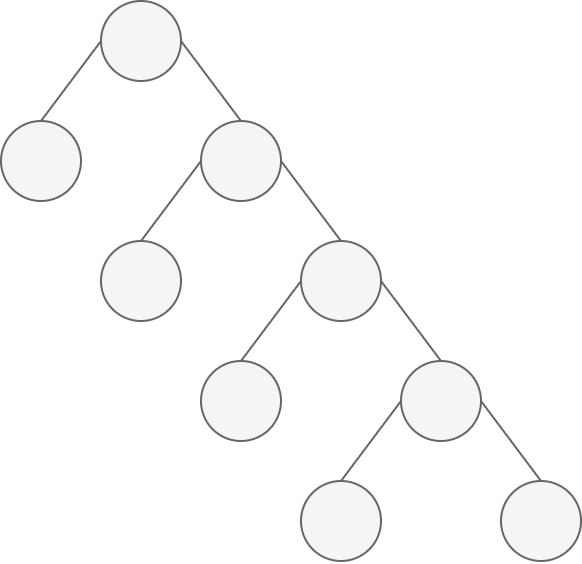
**PROBLEM 2:**

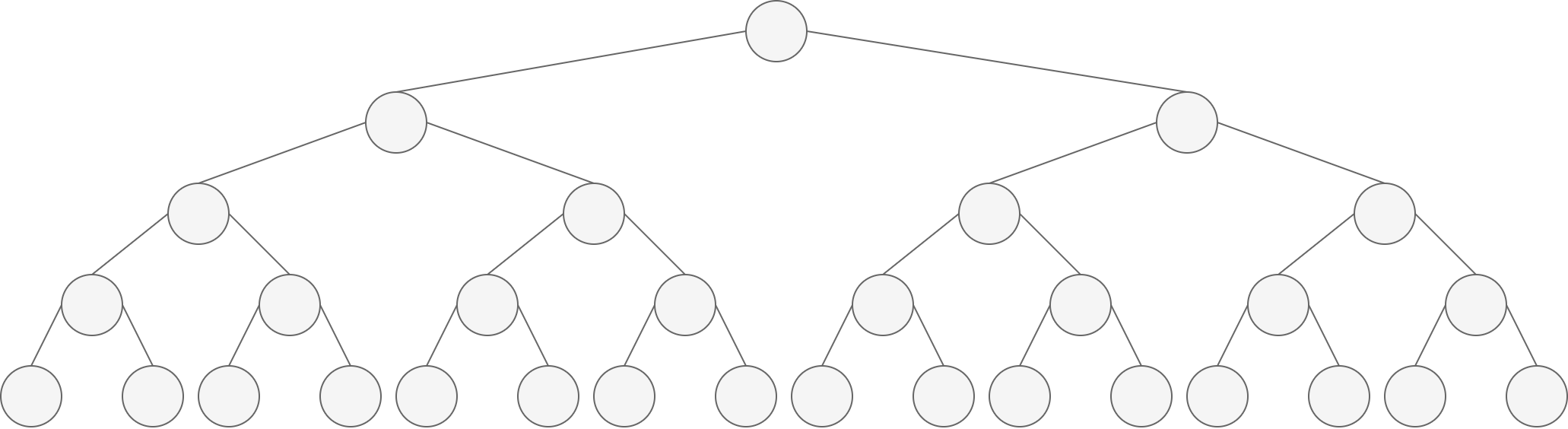
Height = 4

1. **T is complete but not full.**



1. **T is full but not complete.**



1. **T is a perfect binary tree**

**PROBLEM 3:**

A full binary tree T. N is the total number of nodes in T. L is the total leaves. I is the total internal nodes + root.

T has N nodes T has N – 1 edges (1)

T is a full binary tree Every internal node of T has 2 edges.

T has I nodes which are total internal nodes + root T has 2I edges (2)

1. **L = I + 1**
2. and (2) N – 1 = 2I

L + I – 1 = 2I

L = I + 1

1. **N = 2I + 1**
2. and (2) N – 1 = 2I N = 2I + 1
3. **I = (N – 1) / 2**

(1) and (2) N – 1 = 2I I = (N – 1) / 2

1. **L = (N + 1)/2**

a/ L = I + 1

c/ I = (N – 1) / 2

L = (N + 1) / 2

1. **N = 2L – 1**

d/ L = (N + 1) / 2 N = 2L – 1

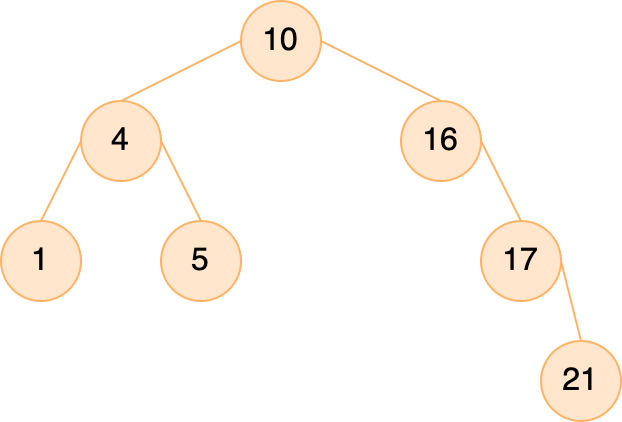
1. **I = L – 1**

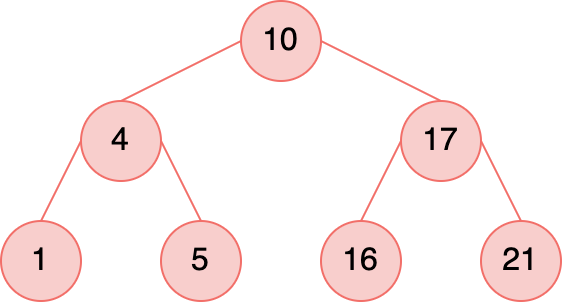
a/ L = I + 1 I = L – 1

**PROBLEM 4:**

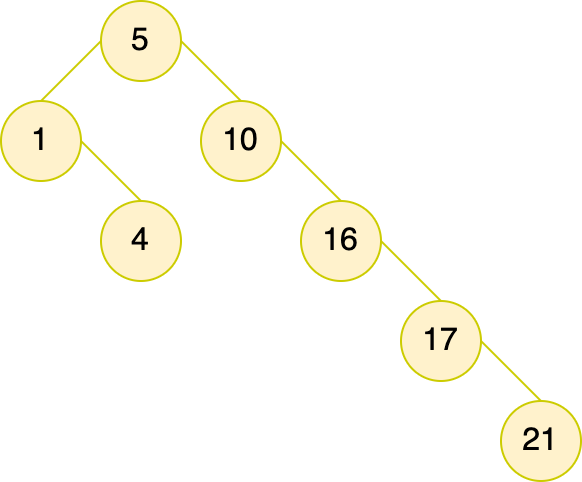
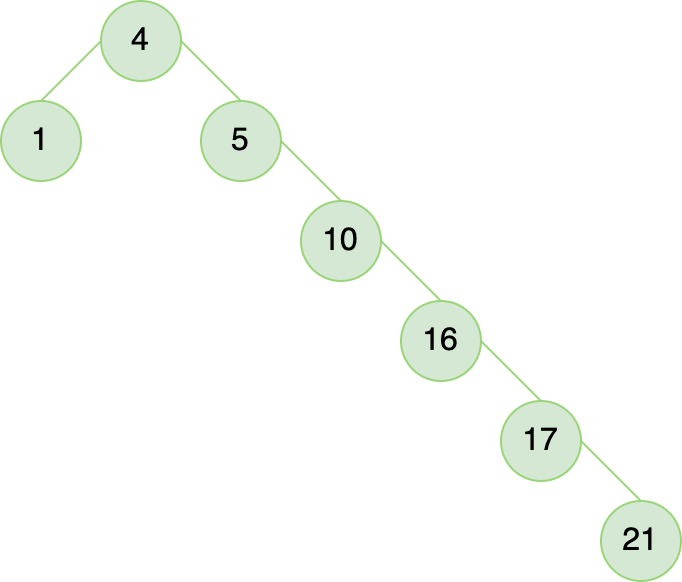
Set of keys {1, 4, 5, 10, 16, 17, 21}

**Height = 2 Height = 3**

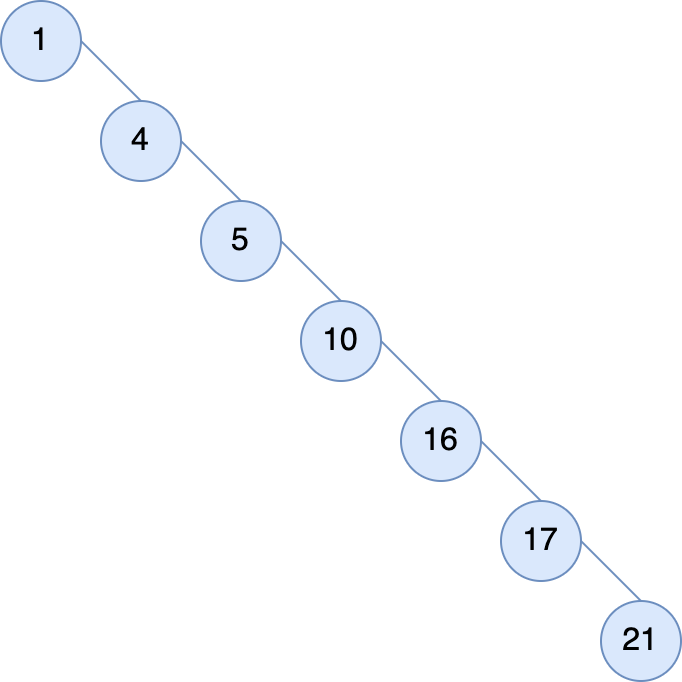
****



**Height = 4 Height = 5**

****

**Height = 6**

****

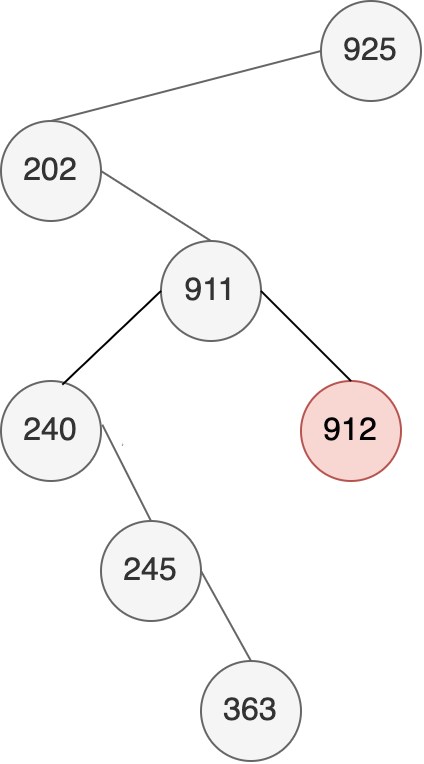
**PROBLEM 5:**

* The binary-search-tree property guarantees that all nodes in the left subtree are smaller, and all nodes in the right subtree are larger.
* The min-heap property only guarantees the general child-larger-than-parent relation, but doesn't distinguish between left and right children. For this reason, the min-heap property can't be used to print out the keys in sorted order in linear time because we have no way of knowing which subtree contains the next smallest element.

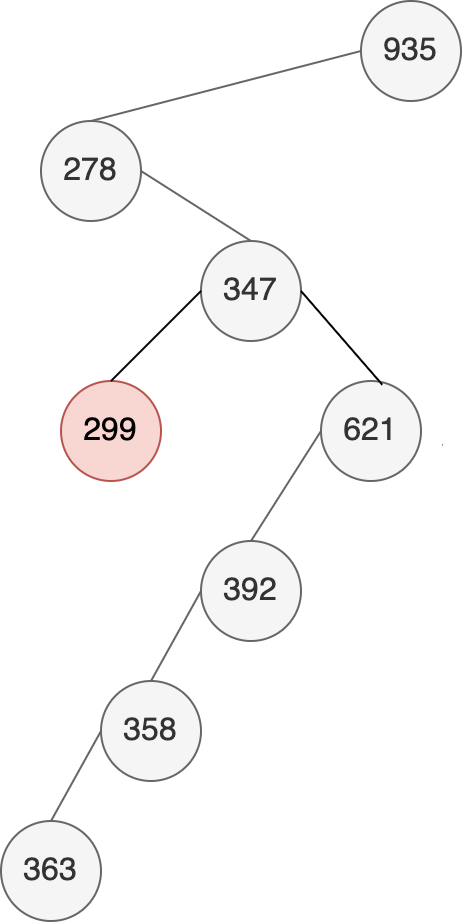
**PROBLEM 6:**

The sequence c and e could not be the sequence of nodes examined.

c) Because whilst searching for 363, the node 912 is not examined (363 < 911)



e) Because whilst searching for 363, the node 299 is not examined (363 > 299)



**PROBLEM 7:**

// Recursive versions of TREE-MINIMUM

TREE\_MINIMUM (NODE \*root)

{

**if** (root->left != **NULL**)

TREE\_MINIMUM(root->left);

**else** **return** root;

}

// Recursive versions of TREE-MAXIMUM

TREE\_MAXIMUM (NODE \*root)

{

**if** (root->right != **NULL**)

TREE\_MAXIMUM(root->right);

**else** **return** root;

}

**PROBLEM 8:**

TREE-PREDECESSOR(x)

**if** x.left != NIL

**return** TREE-MAXIMUM(x.left)

y = x.p

**while** y != NIL **and** x == y.left

x = y

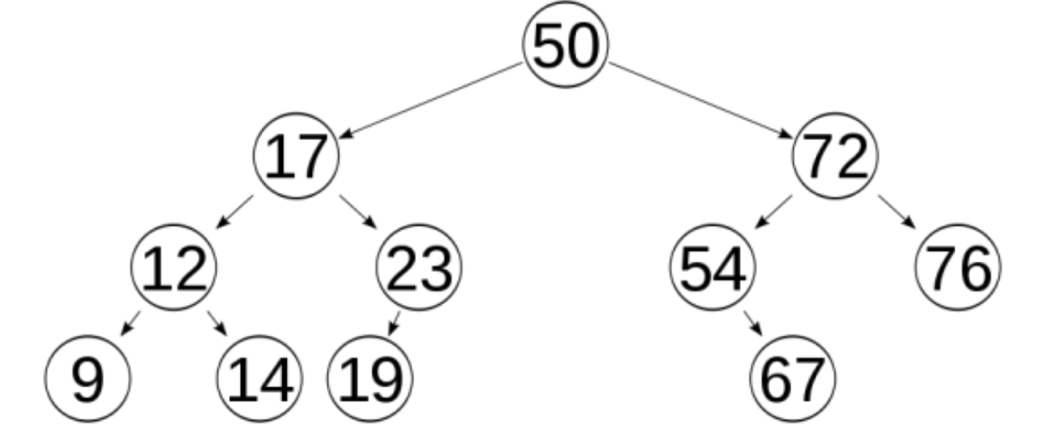
y = y.p

**return** y

**PROBLEM 9:**

Search for 10 in this tree. Then A = {6}, B = {4, 8, 10} and C = {}*.* So, since 6 > 4 it breaks professor's claim.

**PROBLEM 10:**

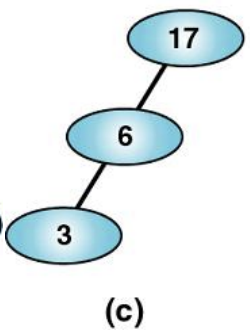
****

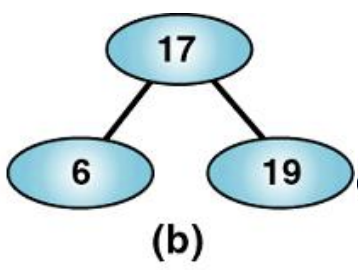
**a) Pre – order :** 50, 17, 12, 9, 14, 23, 19, 72, 54, 67, 76

**b) In – order :** 9, 12, 14, 17, 19, 23, 50, 54, 67, 72, 76

**c) Post – order :** 9, 14, 12, 19, 23, 17, 67, 54, 76, 72, 50

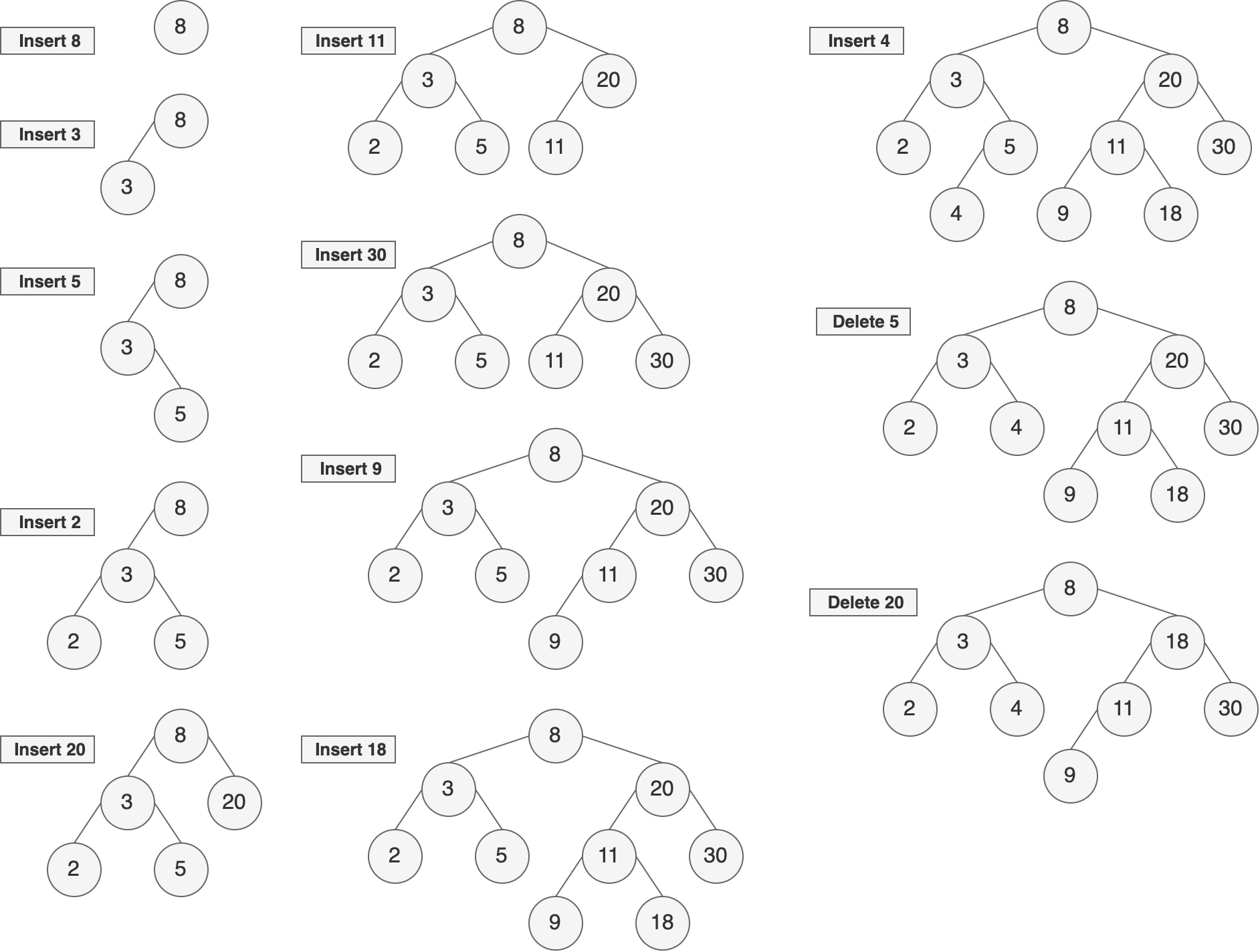
**PROBLEM 11:**

Binary search tree is b and c

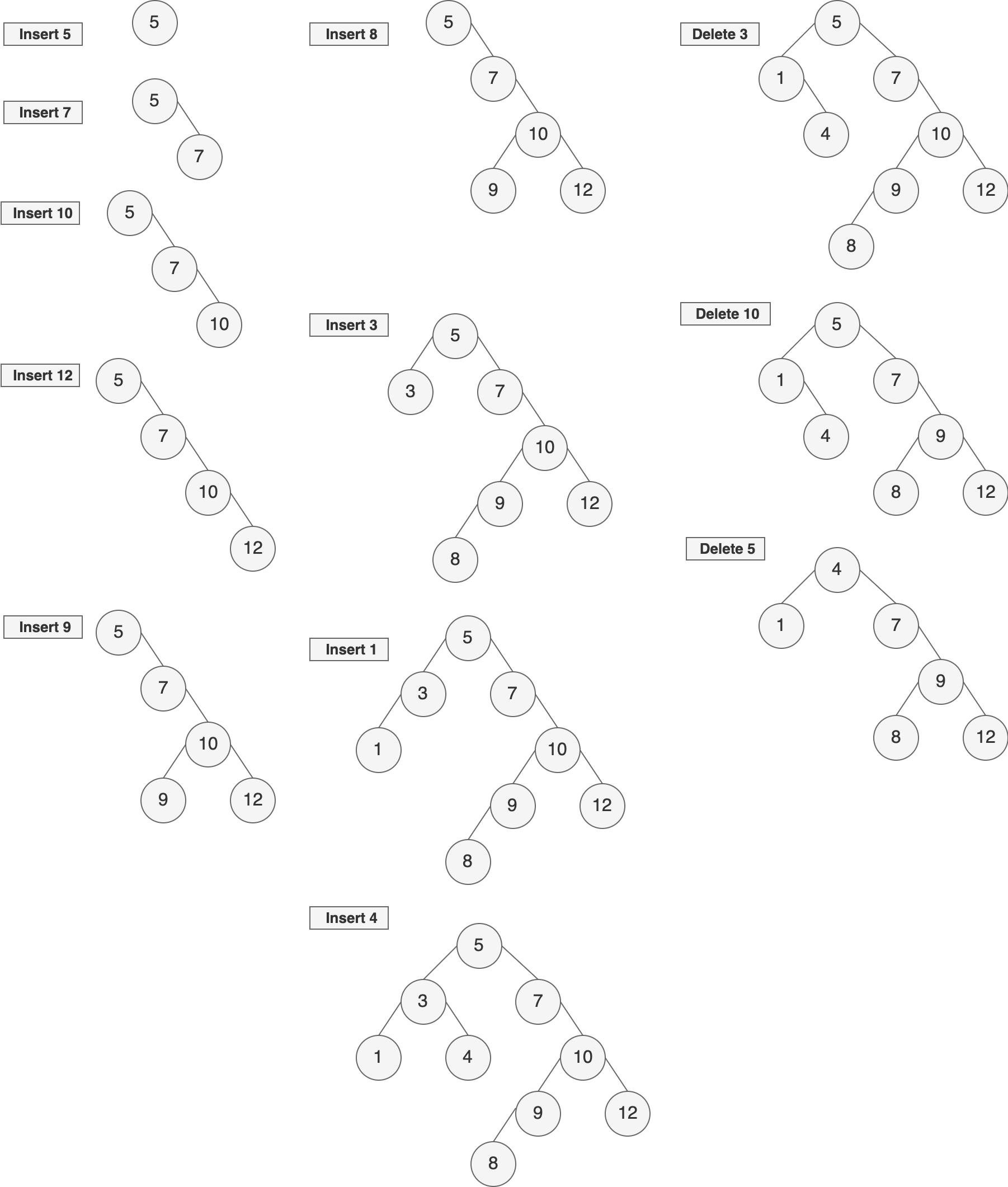
****

**PROBLEM 12:**

**a) Insert** 8, 3, 5, 2, 20, 11, 30, 9, 18, 4**. Delete the nodes:** 5, 20.



**b) Insert** 5, 7, 10, 12, 9, 8, 3, 1, 4. **Delete the nodes** 3, 10, 5.



**PROBLEM 13:**

Yes. Because black node can have have red as well as black children. So, there can be a Red-Black tree where all nodes are black.

**PROBLEM 14:**

The resulting tree is a red-black tree. Because

**Property 1** is trivially satisfied since only one node is changed and it is not changed to some mysterious third color.

**Property 3** is trivially satisfied since no new leaves are introduced.

**Property 4** is satisfied since there was no red node introduced, and root is in every path from the root to the leaves, but no others.

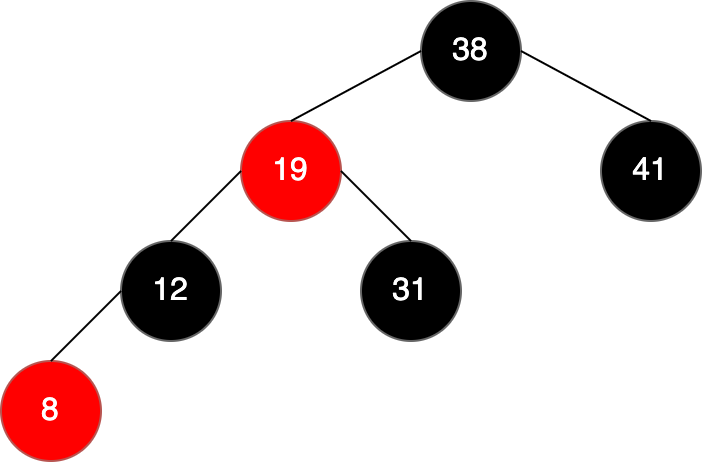
**Property 5** is satisfied since the only paths we will be changing the number of black nodes in are those coming from the root. All of these will increase by 1, and so will all be equal.

**PROBLEM 15:**

From the red-black properties, we have that every simple path from node x to a descendant leaf has the same number of black nodes and that red nodes do not occur immediately next to each other on such paths. Then the shortest possible simple path from node x to a descendant leaf will have all black nodes, and the longest possible simple path will have alternating black and red nodes. Since the leaf must be black, there are at most the same number of red nodes as black nodes on the path.

**PROBLEM 16:**

The red-black trees that result after successively inserting the keys 41, 38, 31, 12, 19, 8 into an initially empty red-black tree.



**PROBLEM 17:**

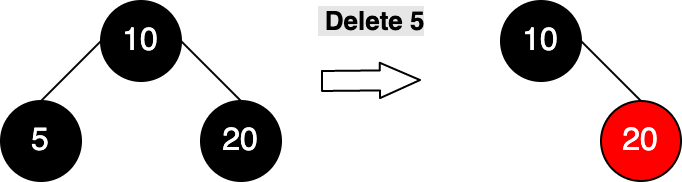
* **Case 1:** z and z.p.p are RED, if the loop terminates, then *z* could not be the root, thus *z* is RED after the fix up.
* **Case 2:** *z* and z.pare RED, and after the rotation z.p could not be the root, thus z.p is RED after the fix up.
* **Case 3:** *z* is RED and z could not be the root, thus *z* is RED after the fix up.

Therefore, there is always at least one red node.

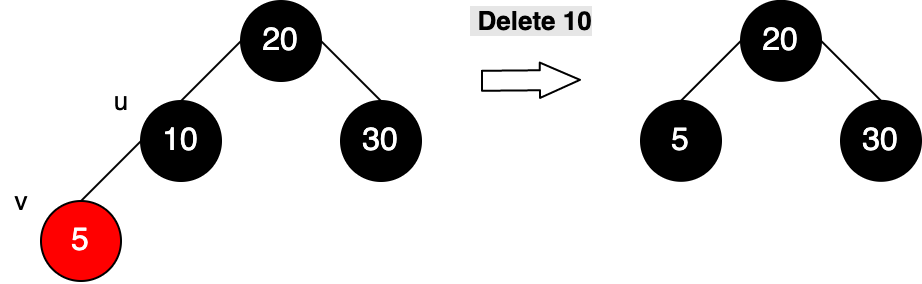
**PROBLEM 18:**

There are 5 cases when we delete a node from a red-black tree

**1. Simple case**

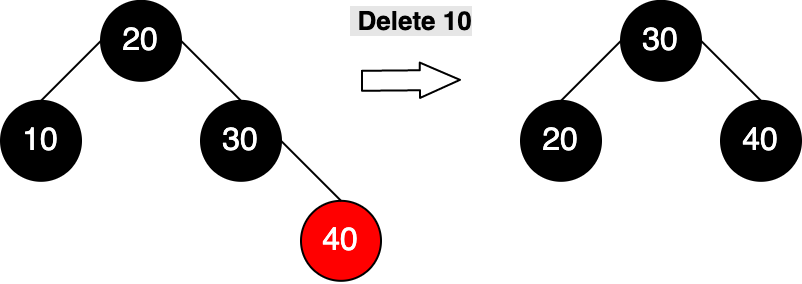
****

**2. If either u or v is red**

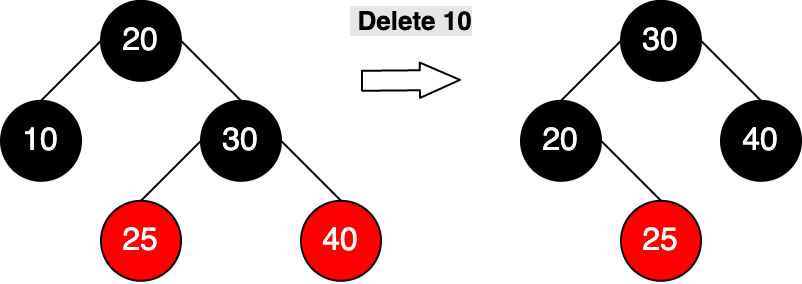
****

**3. If sibling is black and at least one of sibling’s children is red**

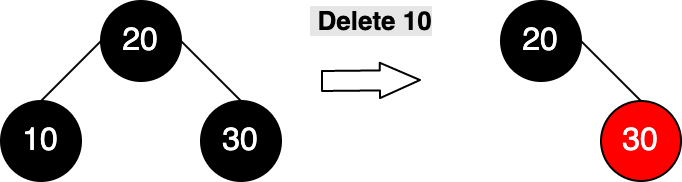
**Ex1:**

****

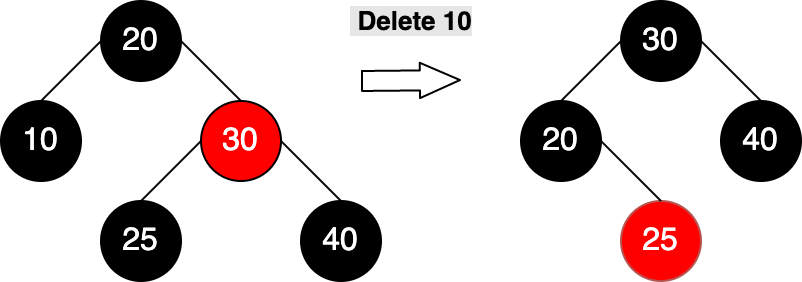
**Ex2:**

****

**4. If sibling is black and its both children are black**

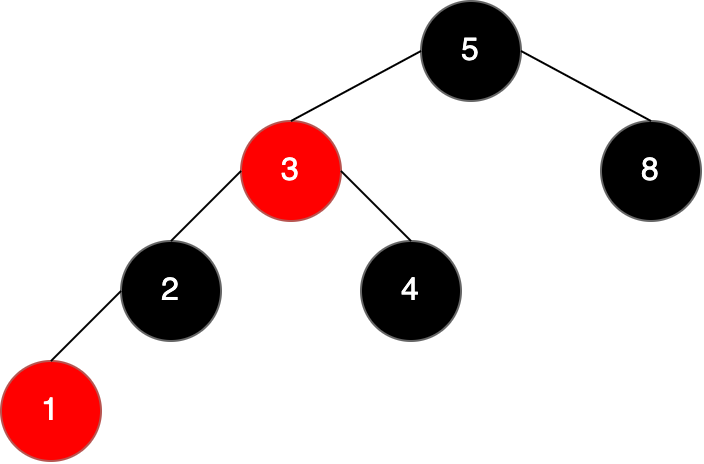
****

**5. If sibling is red**

****

**PROBLEM 19:**

This Red-Black Tree is not an AVL tree structure

****