1. (50 Points) Red-black tree (RBT) is a special version of BST that allows faster retrieval operations. Answer the followings about RBT based on the assigned reading:

a. Mention the properties of an RBT?

1. Every node is either red or black.

2. The root is black.

3. Every leaf (NIL) is black.

4. If a node is red, then both its children are black.

5. For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

---

b. What is the maximum height of an RBT with n nodes? If every path from the root to a null/NIL node contains b black nodes, the tree will have at least how many black nodes? Is it

possible to insert a black node into an RBT? Explain your answer.

2lg(n+1)

Because all simple paths from the node to descendant leaves contain the same number of black nodes , it is not possible to insert more black nodes as this would violate this property.

---

c. Demonstrate all the cases of insertion of a node in RBT? Provide the pseudocode of the insertion that covers all the cases.

Check if the tree is empty (ie. whether x is NIL). If yes, insert newNode as a root node and color it black.

Else, repeat steps following steps until leaf (NIL) is reached.

1.Compare newKey with rootKey.

2.If newKey is greater than rootKey, traverse through the right subtree.

3.Else traverse through the left subtree.

Assign the parent of the leaf as parent of newNode.

If leafKey is greater than newKey, make newNode as rightChild.

Else, make newNode as leftChild

Assign NULL to the left and rightChild of newNode.

Assign RED color to newNode.

Call InsertFix-algorithm to maintain the property of red-black tree if violated.

```
RB-INSERT(T,Z)

y = T.nil

x = T.root

while x  ≠  T.nil

      y = x

      if z.key < x.key

            x = x.left

      else x = x.right

z.p = y

if y == T.nil

      T.root = z

elseif z.key < y.key
```

```
        y.left = z
else y.right = z
z.left = T.nil
z.right = T.nil
z.color = RED
RB-INSERT-FIXUP(T, z)


RB-INSERT-FIXUP(T, z)
while z.p.color == RED
        if z.p == z.p.p. left
                y = z.p.p.right
                if y.color == RED
                        z.p.color = BLACK
                        y.color = BLACK
                        z.p.p.color = RED
                        z = z.p.p
                else if z == z.p.right
                        z = z.p
                        LEFT-ROTATE(T, z)
                        z.p.color = BLACK
                        z.p.p.color = RED
                        RIGHT-ROTATE(T, z.p.p)
```

else (same as then clause

with “right” an= “left” exchanged)

T.root.color = BLACK

---

d. Demonstrate all the cases of deletion of a node from RBT? Provide the pseudocode of the

deletion that covers all the cases.

1. If node to be delete is a red leafnode, Just remove it from the tree
2. IfDB node is root, Remove the DB and root node becomes black
3. (a) IfDB's sibling is black,and(b) DB's sibling's childrenare black, (a) Remove the DB (ifnull DB then delete thenode and for other nodes remove the DBsign)(b) Make DB's sibling red.(c) If DB's parent is black, make it DB.elsemake it black
4. IfDB's sibling is red，(a) Swap color DB's parent with DB's sibling(b) Perform rotation at parent node in thedirection ofDB nodeCheck which case can be applied to this new tree and perform that action
5. (a) DB's sibling is black(b) DB's sibling's childwhich is far from DB is black (c) DB's sibling's childwhich is near to DB is red; (a) Swap color of sibling with sibling's red child(b) Perform rotation at sibling node in directionopposite ofDB node(c) Apply case 6
6. (a) DB's sibling is black, and(b) DB's sibling's far child isred (remember this node) (a) Swap color of DB's parent with DB'ssibling's color(b) Perform rotation at DB's parent in directionofDB(c) Remove DB sign and make the node normalblack node(d) Change

colour ofDB's sibling's far red child to black.

```
RB-DELETE(T, z)

y = z

y-original-color = y.color

if z.left == T.nil

    x = z.right

    RB-TRANSPLANT(T, z.z.right)

elseif z.right == T.nil

    x = z.left

    RB-TRANSPLANT(T, z.z.left)

else y = TREE-MINIMUM.z.right/

    y-original-color = y.color

    x = y.right

    if y.p == z

        x.p = y

    else RB-TRANSPLANT(T, y.y.right)

        y.right = z.right

        y.right.p = y

    RB-TRANSPLANT(T,z,y)

    y.left = z.left

    y.left.p = y

    y.color = z.color
```

```
if y-original-color == BLACK
    RB-DELETE-FIXUP(T,x)

RB-DELETE-FIXUP(T, x)
while x == T.root anD x.color == BLACK
    if x == x.p. left
        w = x.p.right
        if w.color == RED
            w.color = BLACK
            x.p.color = RED
            LEFT-ROTATE(T, x.p)
            w = x.p.right
        if w.left.color == BLACK and w.right.color == BLACK
            w.color = RED
            x = x.p
        else if w.right.color == BLACK
            w.left.color = BLACK
            w.color = RED
            RIGHT-ROTATE(T,w)
            w = x.p.right
        w.color = x.p.color
        x.p.color = BLACK
```

w.right.color = BLACK

LEFT-ROTAT(T, x.p)

x = T.root

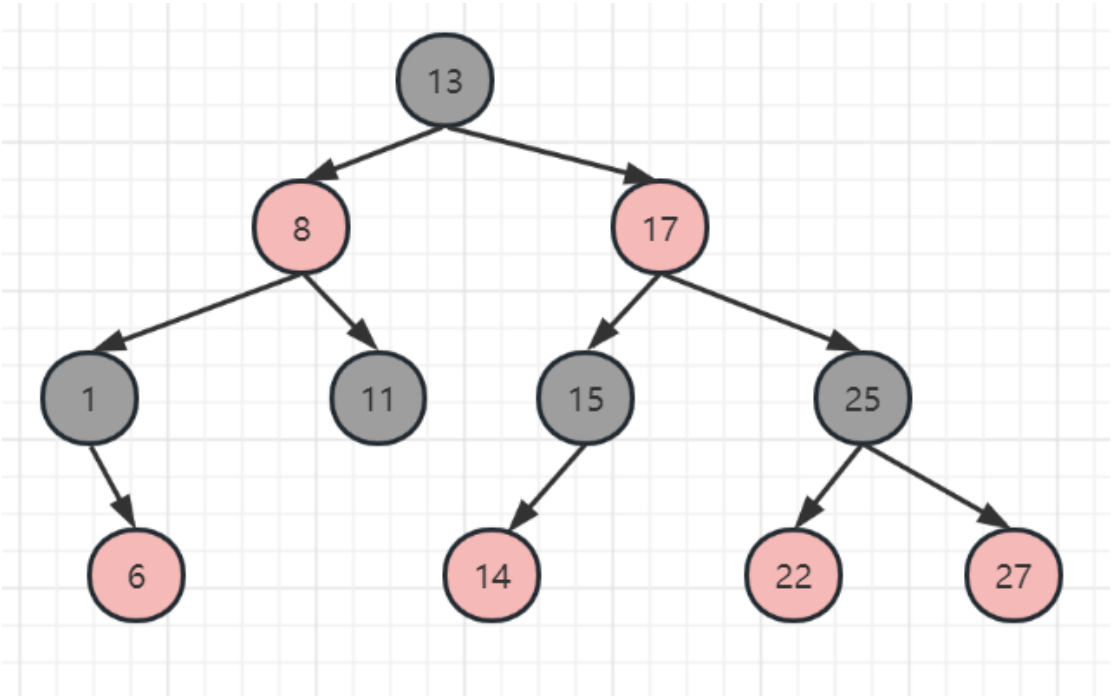
else (same as then clause with “right” and “left” exchanged)

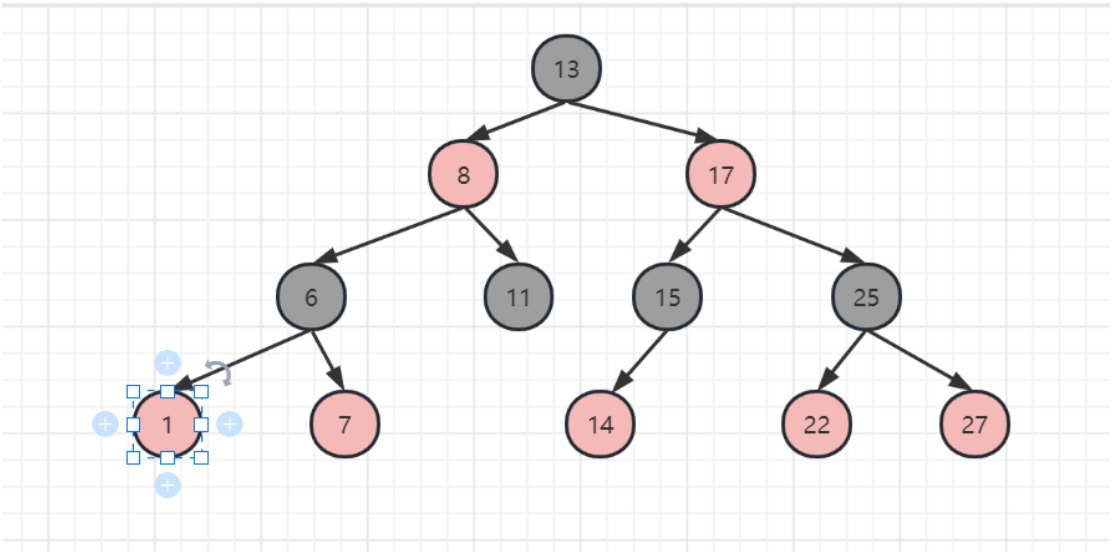
x.color = BLACK

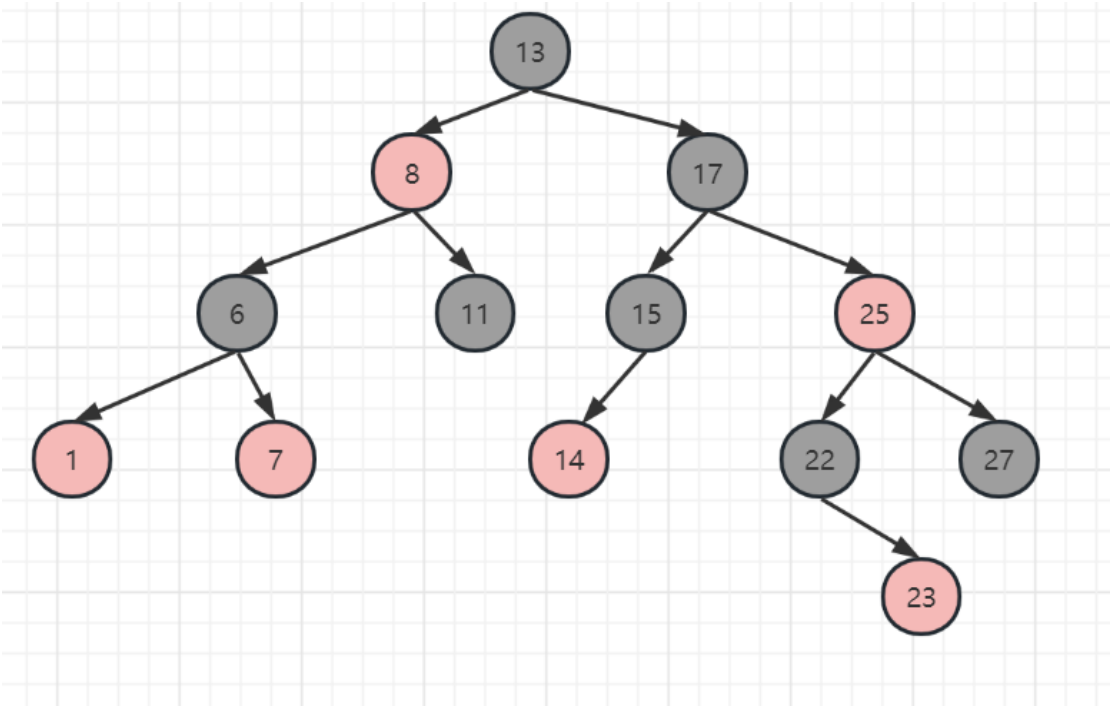
---

e. Insert the nodes with key/value 14, 7, 23 into the following RBT in the given order. Redraw

the tree for each of the steps. (Make sure to mention the color of the nodes)




13
8
17
6
11
15
25
1
7
14
22
27


13
8
17
6
11
15
25
1
7
14
22
27
23