שאלות

שאלה 1

כתוב תוכנית הבודקת האם עץ בינארי הינו BST. מהו זמן הריצה? הסבירו היטב.

חתימת הפונקציה: $public\ boolean\ isBST()$

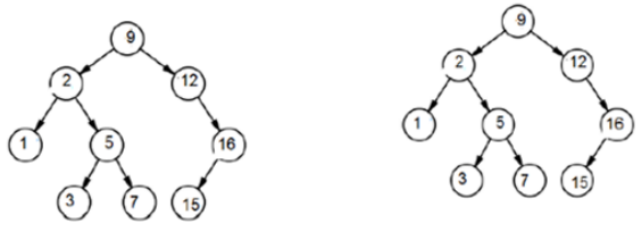
רמז: זכרו שניתן לכתוב פונקצית עזר ריקרוסיבית שמקבלת פרמטרים.

שאלה 2

כתבו פונקציה המקבלת שני שורשים ($Node\ a, Node\ b$)ומחזירה $True$ אם השורשים הם שורשים לשני עצי חיפוש בינארי שווים , ו - $False$ אחרת.

- שני עצי חיפוש בינארי הם שווים רק אם ערכי המפתח שלהם שווים ויש להם אותה הצורה בדיוק.

דוגמא 1: שני עצים חיפוש בינאריים הבאים שווים :

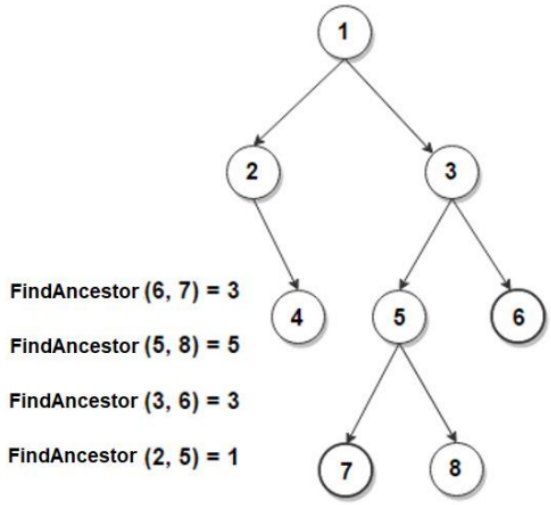


דוגמא 2 : שני עצים חיפוש בינאריים הבאים שונים :

שאלה 3

שאלת מבחן: כתוב פונקציה המקבלת שורש של עץ חיפוש בינארי ושני ערכים ומחזירה את האב הקדמון המשותף הקרוב ביותר של שני הערכים בעץ.

חתימת הפונקציה: $public\ Node\ findLCA(Node\ node,\ int\ n1,\ int\ n2)$



שאלה 4

כתוב תוכנית אשר מקבלת שורש ומספר k ומוצאת את האיבר ה k הקטן ביותר בעץ חיפוש בינארי.

חתימת הפונקציה: $public\ int\ kthSmallest(Node\ root,\ int\ k)$

שאלה 5

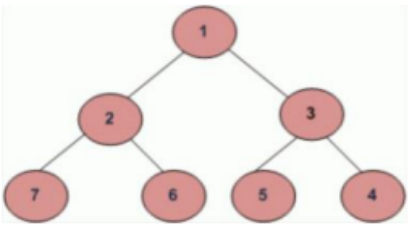
שאלת מבחן: כתוב פונקציה המקבלת שורש של עץ בינארי ומדפיסה את כל הקודקדים שיש להם בן שהוא עלה. יש לסרוק את העץ בסדר *inorder*.

שאלה 6

כתבו פונקציה המקבלת עץ ובודקת האם העץ הוא עץ מושלם.

שאלה 7

כתבו פונקציה המקבלת צומת שהוא השורש של עץ בינארי, ומדפיסה את איבריו בסדר order-level, כלומר רמה אחר רמה. **לדוגמא,** עבור העץ:



הפלט של הפונקציה יהיה 1 2 3 7 6 5 4.

סיבוכיות נדרשת: $O(n)$

שאלה 8

כתבו פונקציה המקבלת צומת שהוא השורש של עץ בינארי, ומדפיסה את איבריו בסדר order-level-reverse כלומר הדפסת רמות מלמטה למעלה.

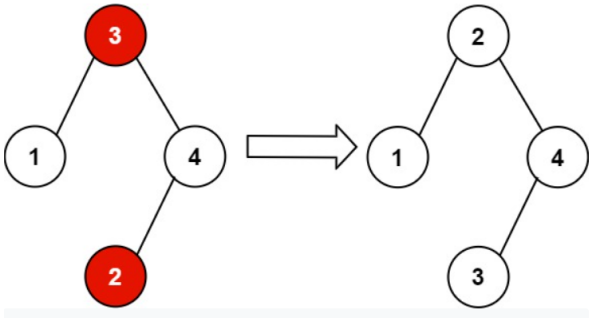
דוגמא: עבור העץ משאלה 7 הפלט של הפונקציה יהיה $7\ 6\ 5\ 4\ 2\ 3\ 1$.

סיבוכיות נדרשת: $O(n)$

שאלה 9 - שאלת בונוס

כתבו פונקציה המקבלת צומת שהוא השורש של עץ חיפוש בינארי,אשר בדיוק 2 מאיבריו הוחלפו ,ומתקנת את העץ חיפוש בינארי ליהיות תקין.

דוגמא: משמאל העץ שקיבלנו בקלט, מימין העץ המתוקן.



סיבוכיות נדרשת: $O(nlogn)$

למיטבי לכת - $O(n)$ ללא שימוש בזכרון נוסף.

פתרונות

פתרון שאלה 1

```
public boolean isBST() {
  return isBST(root,Integer.MIN_VALUE, Integer.MAX_VALUE);
}
private boolean isBST(Node curr, int min, int max) {
  /* an empty tree is BST */
  if (curr == null)
    return true;

  /* false if this node violates the min/max constraints */
  if (curr.data < min || curr.data > max)
    return false;

  /* otherwise check the subtrees recursively
      tightening the min/max constraints */
  // Allow only distinct values
  return (isBST(curr.left, min, curr.data-1) &&
      isBST(curr.right, curr.data+1, max));
}

// Find minimum value function in Binary Tree:
public static int minValue(Node current) {
  if(current == null) return Integer.MAX_VALUE;
  int data = current.data;
  int rdata = minValue(current.right);
  int ldata = minValue(current.left);
  if(rdata < data) data = rdata;
  if(ldata < data) data = ldata;
  return data;
}
```

פתרון שאלה 2

```
1   public static boolean isIdentical(Node x, Node y) {
2     //base case both are empty
3     if (x == null && y == null)
4       return true;
5
6     // both non-empty -> compare them
7     if (x != null && y != null)
8       return (x.data == y.data
9       && isIdentical(x.left, y.left)
10      && isIdentical(x.right, y.right));
11
12    //one reached null the other didnt -> not identical
13    return false;
14  }
```

פתרון שאלה 3

```
1   // This function assumes that n1 and n2 are present in Binary
    Tree.
2   // Please implement this function by first finding the nodes which
    hold n1  and n2
3   public Node findLCA(Node node, int n1, int n2)
4   {
5     if (node == null)
6       return null;
7
8     // If both n1 and n2 are smaller than root, then LCA lies in
    left
9     if (node.data > n1 && node.data > n2)
10      return findLCA(node.left, n1, n2);
11
12    // If both n1 and n2 are greater than root, then LCA lies in
    right
13    if (node.data < n1 && node.data < n2)
14      return findLCA(node.right, n1, n2);
15    return node;
16  }
```

פתרון שאלה 4

```
1    public int kthSmallest(Node root, int k) {
2      if( k > size() ) return Integer.MAX_VALUE;
3      List<Integer> list = new ArrayList<>();
4      InOrder(root, list);
5      return list.get(k - 1);
6    }
7
8
9    private void InOrder(Node root, List<Integer> list) {
10     if (root == null) return;
11     InOrder(root.left, list);
12     list.add(root.data);
13     InOrder(root.right, list);
14   }
```

פתרון שאלה 5

```
1  public boolean is_a_leaf(Node n){ //helper function to determine if
                                     //a given node is a leaf
2    if(n==null)
3      return false;
4    if(n.right==null && n.left==null)
5      return true;
6  }
7
8  public has_leaf_child(Node root){ //main function
9    if(root==null || is_a_leaf(root))//base case
10     return;
11   has_leaf_child(root.left)//starting inorder
12   if(is_a_leaf(root.left) || is_a_leaf(root.right))
13      system.out.println(root.key)
14 }
```

פתרון שאלה 6

```
1    public boolean isFullTree(Node node) {
2      // if empty tree
3      if(node == null)
4        return true;
5
6      // if leaf node
7      if(node.left == null && node.right == null )
8        return true;
9
10     // if both left and right subtrees are not null
11     // Return Recur on Left subtree and right Subtree
12     if((node.left != null) && (node.right != null))
13       return (isFullTree(node.left) && isFullTree(node.right));
14
15     // if none work
16     return false;
17   }
```

פתרון שאלה 7

```
public void PrintLevel() {
    Queue<Node> q = new LinkedList<>();
    q.add(root);
    while (!q.isEmpty()) {
        Node temp = q.poll();
        System.out.println(temp);
        if (temp.left != null)
            q.add(temp.left);
        if (temp.right != null)
            q.add(temp.right);
    }
}
```

פתרון שאלה 8

```
 public void PrintReverseLevel() {
      Stack<Node> a = new Stack<>();
      Queue<Node> q = new LinkedList<>();
      q.add(root);
      while (!q.isEmpty()) {
          Node temp = q.poll();
          if (temp.right != null)
              q.add(temp.right);
          if (temp.left != null)
              q.add(temp.left);
          a.add(temp);
      }
      while(!a.isEmpty()){
          System.out.println(a.pop());
      }
  }
```

פתרון שאלה 9

```
 //this solution assumes TreeNode has the following class members:
//TreeNode left, TreeNode right, int val
public void recoverTree(TreeNode root) {//main function
      ArrayList<INTEGER> nodes;
      inorderadd(nodes,root);
      Collections.sort(nodes);
      int i =0; //pointer for the sorted Arraylist
      inorderplace(nodes,root,i);
  }
  public void inorderadd(ArrayList<INTEGER> nodes , TreeNode curr){
      if(curr == null){
          return;
      }
      inorderadd(nodes,curr.left);
      nodes.add(curr.val);
      inorderadd(nodes,curr.right);
  }
  public void inorderplace(ArrayList<INTEGER> nodes , TreeNode
curr,int i){
      if(curr == nullptr){
          return;
      }
      inorderplace(nodes,curr.left,i);
      curr.val = nodes.at(i++);
      inorderplace(nodes,curr.right,i);
  }
```