**Write a Java program to find the nodes which are at the maximum distance in a Binary Tree**

```java
 import java.util.ArrayList;

public class MaxDistance {

    //Represent a node of binary tree
    public static class Node{
        int data;
        Node left;
        Node right;

        public Node(int data){
            //Assign data to the new node, set left and right children to null
            this.data = data;
            this.left = null;
            this.right = null;
            }
          }

    //Represent the root of binary tree
    public Node root;

    int[] treeArray;
    int index = 0;

    public MaxDistance(){
        root = null;
    }

    //calculateSize() will calculate size of tree
    public int calculateSize(Node node)
    {
        int size = 0;
        if (node == null)
         return 0;
        else {
            size = calculateSize (node.left) + calculateSize (node.right) + 1;
            return size;
        }
    }

    //convertBTtoArray() will convert binary tree to its array representation
    public void convertBTtoArray(Node node) {
        //Check whether tree is empty
        if(root == null){
            System.out.println("Tree is empty");
```

```
            return;
        }
        else {
            if(node.left != null)
                convertBTtoArray(node.left);
            //Adds nodes of binary tree to treeArray
            treeArray[index] = node.data;
            index++;
            if(node.right != null)
                convertBTtoArray(node.right);
            }
    }

    //getDistance() will find distance between root and a specific node
    public int getDistance(Node temp, int n1) {
        if (temp != null) {
            int x = 0;
            if ((temp.data == n1) || (x = getDistance(temp.left, n1)) > 0
                    || (x = getDistance(temp.right, n1)) > 0) {
                //x will store the count of number of edges between temp and node n1
                return x + 1;
            }
            return 0;
        }
        return 0;
    }

    //lowestCommonAncestor() will find out the lowest common ancestor for nodes node1
 and node2
    public Node lowestCommonAncestor(Node temp, int node1, int node2) {
        if (temp != null) {
            //If root is equal to either of node node1 or node2, return root
            if (temp.data == node1 || temp.data == node2) {
                return temp;
            }

            //Traverse through left and right subtree
            Node left = lowestCommonAncestor(temp.left, node1, node2);
            Node right = lowestCommonAncestor(temp.right, node1, node2);

            //If node temp has one node(node1 or node2) as left child and one node(node1
or node2) as right child
            //Then, return node temp  as lowest common ancestor
            if (left != null && right != null) {
                return temp;
            }

            //If nodes node1 and node2 are in left subtree
```

```
            if (left != null) {
                return left;
            }
            //If nodes node1 and node2 are in right subtree
            if (right != null) {
                return right;
            }
        }
        return null;
    }

    //findDistance() will find distance between two given nodes
    public int findDistance(int node1, int node2) {
        //Calculates distance of first node from root
        int d1 = getDistance(root, node1) - 1;
        //Calculates distance of second node from root
        int d2 = getDistance(root, node2) - 1;

        //Calculates lowest common ancestor of both the nodes
        Node ancestor = lowestCommonAncestor(root, node1, node2);

        //If lowest common ancestor is other than root then, subtract 2 * (distance of root
to ancestor)
        int d3 = getDistance(root, ancestor.data) - 1;
        return (d1 + d2) - 2 * d3;
    }

    //nodesAtMaxDistance() will display the nodes which are at maximum distance
    public void nodesAtMaxDistance(Node node) {
        int maxDistance = 0, distance = 0;
        ArrayList<Integer> arr = new ArrayList<>();

        //Initialize treeArray
        int treeSize = calculateSize(node);
        treeArray = new int[treeSize];

        //Convert binary tree to its array representation
        convertBTtoArray(node);

        //Calculates distance between all the nodes present in binary tree and stores max
mum distance in variable maxDistance
        for(int i = 0; i < treeArray.length; i++) {
            for(int j = i; j < treeArray.length; j++) {
                distance = findDistance(treeArray[i], treeArray[j]);
                //If distance is greater than maxDistance then, maxDistance will hold the valu
e of distance
                if(distance > maxDistance) {
                    maxDistance = distance;
```

```
                arr.clear();
                //Add nodes at position i and j to treeArray
                arr.add(treeArray[i]);
                arr.add(treeArray[j]);
            }
              //If more than one pair of nodes are at maxDistance then, add all pairs to tre
Array
              else if(distance == maxDistance) {
                arr.add(treeArray[i]);
                arr.add(treeArray[j]);
              }
           }
        }
        //Display all pair of nodes which are at maximum distance
        System.out.println("Nodes which are at maximum distance: ");
        for(int i = 0; i < arr.size(); i = i + 2) {
          System.out.println("( " + arr.get(i) + "," + arr.get(i+1) + " )");
        }
    }

    public static void main(String[] args) {

        MaxDistance bt = new MaxDistance();
        //Add nodes to the binary tree
        bt.root = new Node(1);
        bt.root.left = new Node(2);
        bt.root.right = new Node(3);
        bt.root.left.left = new Node(4);
        bt.root.left.right = new Node(5);
        bt.root.right.left = new Node(6);
        bt.root.right.right = new Node(7);
        bt.root.right.right.right = new Node(8);
        bt.root.right.right.right.left = new Node(9);

        //Finds out all the pair of nodes which are at maximum distance
        bt.nodesAtMaxDistance(bt.root);
     }
}
```

## Output:

```
Nodes which are at maximum distance:
( 4,9 )
( 5,9 )
```