```
/ Author: Jose Enriquez// Author: Jose Enriquez
// Email: jose.enriquez@okstate.edu
// 4/30/2021
// simulates a social network to demonstrate a Graph AdjacencyList structure
public class BinarySearchTree {
    private BSTNode root;
    private int size;

    //Creates a BST.
    public BinarySearchTree(){
        this.size = 0;
        this.root = null;
    }

    //deletes a given user
    public void delete(User user){
        delete(this.root, user);
    }

    /**
     * Deletes a node from a binary tree
     * @param root the current node we are comparing.
     * @param user the data of the node to delete.
     * @return deleted node.
     */
    public BSTNode delete(BSTNode root, User user){
        //pointer to store the parent of the current node
        BSTNode parent = null;

        //start with root node;
        BSTNode curr = root;

        //search key in the BST and set its parent pointer
        while(curr != null && curr.user != user){
            //update the parent to current node
            parent = curr;

            if(user.userIDOnlyNumbers < curr.user.userIDOnlyNumbers){
                //traverse left subtree
                curr = curr.left;
            } else {
                //traverse right subtree
                curr = curr.right;
            }
        }

        if(curr == null){
            //return if the User is not found in the tree
            return null;
        }
```

```
        if(curr.left == null && curr.right == null){
            //case 1: Node to be deleted is a leaf

            if(curr != root){
                if(parent.left == curr) {
                    //Node to be deleted is not the root and its a leftchild
                    parent.left.user = null;
                    parent.left = null;
                }
                else{
                    //Node to be deleted is not the root and its a rightchild
                    parent.right.user = null;
                    parent.right = null;
                }
            }
            else{
                //tree only has a root node
                //System.out.println("reach");
                root.user = null;
                root = null;

            }
        }

        else if(curr.left != null && curr.right != null){
            //Case 2: Node to be deleted has two children.
            BSTNode successor = getMin(curr.right);

            User data = successor.user;

            //recursively delete successor. Successor should have at most one child
            delete(root, successor.user);

            //copy value of successor to the current node
            curr.user = data;
        }

        else{
            //Case 3: node to be deleted has only one child
            BSTNode child = (curr.left != null)? curr.left: curr.right;

            if(curr != root){
                //if the node to be deleted is not a root node, set its parent to
its child
                if(curr == parent.left) {
                    parent.left.user = child.user;
                    parent.left = child;
                }
                else {
```

```
                parent.right.user = child.user;
                parent.right = child;
            }
        } else {
            System.out.println("case 3 reach");

            //node to be deleted is root node
            root.user = child.user;
            root = child;
        }

    }

    return root;


}

//gets the min value in a subtree
public static BSTNode getMin(BSTNode curr){
    while(curr.left != null){
        curr = curr.left;
    }
    return curr;
}

/**
 * helps to check if tree contains user.
 * @param user The specific user we are looking for
 * @return boolean true if User is found.
 */
public boolean search(User user){
    boolean check = search(root, user);
    return check;
}

/**
 * Checks if root contains a specific User.
 * @param root The current node we are comparing to.
 * @param user The specific user we are looking for
 * @return boolean true if User is found.
 */
public boolean search(BSTNode root, User user){
    if(root == null|| root.user == null){
        //User is not in the tree
        return false;
    }
    if(root.user.userIDOnlyNumbers == user.userIDOnlyNumbers){
        //User is in the tree.
        return true;
```

```
    }

    if(root.user.userIDOnlyNumbers < user.userIDOnlyNumbers){
        //traverse right subtree
        return search(root.right, user);
    }

    if(root.user.userIDOnlyNumbers > user.userIDOnlyNumbers){
        //traverse left subtree
        return search(root.left, user);
    }

    return false;
}

/**
 * helps the Insert method.
 * @param user The data we are inserting.
 */
public void insert(User user){
    root = insert(this.root, user, null);
}

/**
 * Inserts a node into the binary search tree.
 * @param root The current node we are comparing to.
 * @param user The data we are inserting
 * @param parent The parent of root.
 * @return BSTnode The node containing the inserted data.
 */
private BSTNode insert(BSTNode root, User user, BSTNode parent){
    if(root == null || root.user == null){

        //If we have reached the end of the tree.
        root = new BSTNode(user);
        root.parent = parent;
        if(size==0){
            this.root = root;
        }
        size++;
        return root;
    }

   if(user.userIDOnlyNumbers < root.user.userIDOnlyNumbers){
       //traverse left of the tree
       root.left = insert(root.left, user, root);

    } else if(user.userIDOnlyNumbers > root.user.userIDOnlyNumbers){
       //traverse right of the tree
       root.right = insert(root.right, user, root);
```

```
        }
       return root;
    }


    //main method Used to test.
    public static void main(String[] args) {
        BinarySearchTree bst = new BinarySearchTree();
        User a1 = new User("joe", "A1234");
        User a2 = new User("joe", "A1222");
        User a3 = new User("joe", "A1555");
        User a4 = new User("joe", "A1444");
        User b1 = new User("joe", "B5454");
        User b2 = new User("joe", "B7474");
        User b3 = new User("joe", "B8998");

        bst.insert(a1);
        bst.insert(a2);
        bst.insert(a3);
        bst.insert(a4);
        bst.insert(b1);
        bst.insert(b2);
        bst.insert(b3);

        User b3Dublicate = new User("joe", "B8998");
        User nonNuplicate = new User("joe", "B8910");

        System.out.println(bst.search(b3Dublicate));
        System.out.println(bst.search(nonNuplicate));
    }

}
```