COSC311Driver.java

- delete not working
- ~~find~~ deleter delete other records, sometimes multiple records

45/50

```
/*
Zane Wonsey
E01021309
Program Number 3
Version 2 (threaded)
*/
/**
 * This will be the main driver program for many of your programs.
Specifically,
 * you will need to define a data structure and related algorithms to use
with this program.
 * We will be using the data file I have provied for you: a file of 68
records. Each record is composed
 * of three fields: String lastName, String firstName, String ID
 * ID may be implemented as an integer, but it is easier to implement as a
string. Both lastName and firstName
 * may not be unique, but the ID **is** unique.
 *
 * @author Bill Sverdlik
 * @version Version 1.0
 */
/*
 * IMPORTANT!!!!!!
 * Your projects should not contain any code in this file that modifies the
class DataStructure directly.
 * You may find it convenient (but not required) that the file DataStructure
contain an inner class.
 */
import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;

public class COSC311Driver {

    private static DataStructure myStructure = new DataStructure();
    private static Scanner keyboard = new Scanner(System.in);

    public static void main(String[] args) {
        loadDataSet();
        /*The following declaration declares a data structure that will
change from one assignment to the next. For example, you will need to
implement
```

COSC311Driver.java

```
         * the following as a doubly linked list, as well as a tree.
         */
        int response;
        do {
            System.out.println(" 1 Add a new student");
            System.out.println(" 2 Delete a student");
            System.out.println(" 3 Find a student by ID");
            System.out.println(" 4 List students by ID increasing");
            System.out.println(" 5 List students by first name increasing");
            System.out.println(" 6 List students by last name increasing");
            System.out.println(" 7 List students by ID decreasing");
            System.out.println(" 8 List students by first name decreasing");
            System.out.println(" 9 List students by last name decreasing");
            System.out.println(" ");
            System.out.println(" 0 End");

            response = keyboard.nextInt();

            switch (response) {
            case 1: addIt();
            break;
            case 2: deleteIt();
            break;
            case 3: findIt();
            break;
            case 4: listItIncreasingID();       // or see below for
programming trick
            break;
            case 5: listItIncreasingfName();
            break;
            case 6: listItIncreasinglName();
            break;
            case 7: listItDecreasingID();
            break;
            case 8: listItDecreasingfName();
            break;
            case 9: listItDecreasinglName();
            default:
            }

        } while (response!=0);
    }
```

```
    // Case 1: Add a new student. We need a unique ID number
    public static void addIt() {

        String name1 = null, name2 = null, tempID;
        DataStructureRecord found;

        System.out.println("Enter a unique ID number to add");
        tempID = keyboard.next();


        //is it unique ?
        found = myStructure.find(tempID);
        if (found != null) {
            System.out.println("ID already in use\n");
        } else {
            System.out.println("Enter first and last name");
            name1=keyboard.next();
            name2=keyboard.next();

            // add to our data structure
            myStructure.insert(name1, name2, tempID);

        }

    }
    // Case 2: delete a student. A student ID must be prompted for. If the
ID number does not exist in the database,
    //    print out a message indicating a such, otherwise delete the entire
record
    public static void deleteIt() {
        DataStructureRecord dsr;
        String id;

        System.out.print("Enter an ID to delete: ");
        id = keyboard.next();

        dsr = myStructure.find(id);

        //If it is found it will return true. If true then it deletes it,
        //if false prints an error message

        if (dsr != null) {
```

COSC311Driver.java

```
            myStructure.delete(dsr);
        } else {
            System.err.print("ID not found");
        }
    }
    //Case 3: find a student. A student ID must be prompted for. If the ID
number is not found, print out a
    //    message indicating as such. Otherwise print out the entire record
    public static void findIt() {
        System.out.println("Enter the ID of the student to search for: ");
        String tempID = keyboard.next();
        //If returns true print out student record info
        if (myStructure.find(tempID) != null) {
            DataStructureRecord person = myStructure.find(tempID);
            System.out.println(person.toString() + "\n");
        } else {
            System.out.println("Could not find that ID\n");
        }
    }
    // Cases 4,5,6,7,8,9
    // A little programming trickery. All of the listing methods below can
call the SAME method in DataStructure.
    // We'll pass 2 parameters: the first indicates which field, the second
indicates which order
    // fieldNum=1 first namefieldNum=2 last namefieldNum=3 ID
    // orderNum=1 increasingorderNum=2 decreasing
    public static void listItIncreasingfName() {
        myStructure.listIt(1, 1);
    }
    public static void listItIncreasinglName() {
        myStructure.listIt(2, 1);
    }
    public static void listItIncreasingID() {
        myStructure.listIt(3, 1);
    }

    public static void listItDecreasingfName() {
        myStructure.listIt(1, 2);
    }
    public static void listItDecreasinglName() {
        myStructure.listIt(2, 2);
    }
```

```
    public static void listItDecreasingID() {
        myStructure.listIt(3, 2);
    }

    public static void loadDataSet() {
        Scanner fileIO = null;
        try {
            fileIO = new Scanner(new File("./dataset.txt"));
            while (fileIO.hasNextLine()) {
                //reads the three needed strings into variables to be stored
into the array
                String first = fileIO.next();
                String last = fileIO.next();
                String id = fileIO.next();
                //creates a record with the three strings and adds it to the
array
                if (myStructure.find(id) == null) {
                    myStructure.insert(first, last, id);
                }
                //advance to next line to read and increment the counter
                fileIO.nextLine();
            }
            fileIO.close();
        } catch (FileNotFoundException e) {
            System.err.println("File was not found");
        } catch (NoSuchElementException e) {
            fileIO.close();
        }
    }
}
```

DataStructure.java

```
public class DataStructure {
    private static DataStructureRecord[] data;
    private static int nElems;
    private Tree firstNameList = new Tree();
    private Tree lastNameList = new Tree();
    private Tree idList = new Tree();

    public DataStructure() {
        data = new DataStructureRecord[100];
    }
    public DataStructureRecord find(String tempID) {
        Node rover = idList.getRoot();
        while (rover != null) {
            if (rover.getKey().equalsIgnoreCase(tempID)) {
                return data[rover.getRecordNumber()];
            } else if (rover.getKey().compareToIgnoreCase(tempID) > 0) {
                System.out.println("1");
                if (!rover.getLeftThread()) {
                    System.out.println("2");
                    rover = rover.getLeft();
                } else {
                    return null;
                }
            } else {
                System.out.println("3");
                if (!rover.getRightThread()) {
                    System.out.println("4");
                    rover = rover.getRight();
                } else {
                    return null;
                }
            }
        }
        return null;
    }

    public void insert(String first, String last, String tempID) {
        data[nElems] = new DataStructureRecord(first, last, tempID);
        firstNameList.insert(first);
        lastNameList.insert(last);
        idList.insert(tempID);
        nElems++;
```

```
}

public void delete(DataStructureRecord dsr) {
    for (int i = 0; i < nElems; i++) {
        if (dsr.getID().equals(data[i].getID())) {
            idList.delete(dsr.getID());
            firstNameList.delete(dsr.getFirstName());
            lastNameList.delete(dsr.getLastName());
            nElems--;
        }
    }
}

public void listIt(int i, int j) {
    if (i == 1 && j == 1) {            //first name increasing
        Node rover = firstNameList.getRoot();
        while (!rover.getLeftThread() && rover.getLeft() != null) {
            rover = rover.getLeft();
        }
        for (int k = 0; k < nElems; k++) {
            System.out.println(data[rover.getRecordNumber()]);
            if (rover.getRightThread()) {
                rover = rover.getRight();
            } else {
                rover = rover.getRight();
                while (rover != null && !rover.getLeftThread()) {
                    rover = rover.getLeft();
                }
            }
        }
        System.out.println();
    } else if (i == 1 && j == 2) {     //first name decreasing
        Node rover = firstNameList.getRoot();
        while (!rover.getRightThread() && rover.getRight() != null) {
            rover = rover.getRight();
        }
        for (int k = 0; k < nElems; k++) {
            System.out.println(data[rover.getRecordNumber()]);
            if (rover.getLeftThread()) {
                rover = rover.getLeft();
            } else {
                rover = rover.getLeft();
```

unless delete from end, nElems no longer points to last element in array...

```
                while (rover != null && !rover.getRightThread()) {
                    rover = rover.getRight();
                }
            }
        }
        System.out.println();
    } else if (i == 2 && j == 1) {      //increasing last name
        Node rover = lastNameList.getRoot();
        while (!rover.getLeftThread() && rover.getLeft() != null) {
            rover = rover.getLeft();
        }
        for (int k = 0; k < nElems; k++) {
            System.out.println(data[rover.getRecordNumber()]);
            if (rover.getRightThread()) {
                rover = rover.getRight();
            } else {
                rover = rover.getRight();
                while (rover != null && !rover.getLeftThread()) {
                    rover = rover.getLeft();
                }
            }
        }
        System.out.println();
    } else if (i == 2 && j == 2) {      //decreasing last name
        Node rover = lastNameList.getRoot();
        while (!rover.getRightThread() && rover.getRight() != null) {
            rover = rover.getRight();
        }
        for (int k = 0; k < nElems; k++) {
            System.out.println(data[rover.getRecordNumber()]);
            if (rover.getLeftThread()) {
                rover = rover.getLeft();
            } else {
                rover = rover.getLeft();
                while (rover != null && !rover.getRightThread()) {
                    rover = rover.getRight();
                }
            }
        }
        System.out.println();
    } else if (i == 3 && j == 1) {      //increasing ID
        //System.out.println("increasing id");
```

```
        Node rover = idList.getRoot();
        while (!rover.getLeftThread() && rover.getLeft() != null) {
            System.out.println("here");
            rover = rover.getLeft();
        }
        for (int k = 0; k < nElems; k++) {
            //System.out.println("in for loop at all?");
            if (rover != null) {
                System.out.println(data[rover.getRecordNumber()]);
                if (rover.getRightThread()) {
                    rover = rover.getRight();
                } else {
                    //System.out.println("before second while");
                    rover = rover.getRight();
                    while (rover != null && !rover.getLeftThread()) {
                        //System.out.println("here2");
                        rover = rover.getLeft();
                    }
                }
            } else {

            }
        }
        System.out.println();
    } else if (i == 3 && j == 2) {      //decreasing ID
        Node rover = idList.getRoot();
        while (!rover.getRightThread() && rover.getRight() != null) {
            rover = rover.getRight();
        }
        for (int k = 0; k < nElems; k++) {
            System.out.println(data[rover.getRecordNumber()]);
            if (rover.getLeftThread()) {
                rover = rover.getLeft();
            } else {
                rover = rover.getLeft();
                while (rover != null && !rover.getRightThread()) {
                    rover = rover.getRight();
                }
            }
        }
        System.out.println();
    }
```

```
    }
}
```

DataStructureRecord.java

```
public class DataStructureRecord {

    private String firstName;
    private String lastName;
    private String ID;

    public DataStructureRecord(String firstName, String lastName, String ID)
{
        this.firstName = firstName;
        this.lastName = lastName;
        this.ID = ID;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public String getID() {
        return ID;
    }

    public void setId(String ID) {
        this.ID = ID;
    }

    public String toString() {
        return ID + ": "  + lastName + ", " + firstName;
    }

}
```

Node.java

```
public class Node {

    private String key;
    private int recordNumber;
    private Node right;
    private boolean rightThread;
    private Node left;
    private boolean leftThread;

    public Node(String key, int recordNumber) {
        this.key = key;
        this.recordNumber = recordNumber;
        leftThread = false;
        rightThread = false;
    }

    public Node(String key, int recordNumber, Node left, Node right) {
        this.key = key;
        this.recordNumber = recordNumber;
        this.left = left;
        this.right = right;
        if (left == null) {
            leftThread = false;
        } else {
            leftThread = true;
        }
        if (right == null) {
            rightThread = false;
        } else {
            rightThread = true;
        }
    }

    public String toString() {
        return key;
    }

    public String getKey() {
        return key;
    }
```

```
public int getRecordNumber() {
    return recordNumber;
}

public Node getRight() {
    return right;
}

public boolean getRightThread() {
    return rightThread;
}

public Node getLeft() {
    return left;
}

public boolean getLeftThread() {
    return leftThread;
}

public void setKey(String key) {
    this.key = key;
}

public void setRecordNumber(int recordNumber) {
    this.recordNumber = recordNumber;
}

public void setRight(Node right) {
    this.right = right;
}

public void setRightThread(boolean rightThread) {
    this.rightThread = rightThread;
}

public void setLeft(Node left) {
    this.left = left;
}

public void setLeftThread(boolean leftThread) {
    this.leftThread = leftThread;
```

```
    }

}
```

Tree.java

```
public class Tree {

    private Node root;
    private int nElems = 0;

    public boolean isEmpty() {
        return (root == null);
    }

    public Node getRoot() {
        return root;
    }

    public void insert(String key) {
        System.out.println(key + " " + nElems);
        if (isEmpty()) {
            root = new Node(key, nElems);
        } else {
            Node temp = root;
            while (true) {
                int compVal = temp.getKey().compareTo(key);
                if (compVal > 0) { // go left
                    Node left = temp.getLeft();
                    if (left == null) { //if null
                        left = new Node(key, nElems, null, temp);
                        temp.setLeft(left);
                        break;
                    } else if (temp.getLeftThread()) {  //if a thread
                        left = new Node(key, nElems, temp.getLeft(), temp);
                        temp.setLeft(left);
                        temp.setLeftThread(false);
                        break;
                    }
                    temp = temp.getLeft();  //move pointer for loop
                } else {        //go right
                    Node right = temp.getRight();
                    if (right == null) {//if null
                        right = new Node(key, nElems, temp, null);
                        temp.setRight(right);
                        break;
                    } else if (temp.getRightThread()) { //if a thread
```

```
                        right = new Node(key, nElems, temp,
temp.getRight());
                        temp.setRight(right);
                        temp.setRightThread(false);
                        break;
                    }
                    temp = temp.getRight(); //move pointer for loop
                }
            }
        }
        nElems++;
    }


    public void delete(String key) {
        Node current = root;
        Node parent = root;
        boolean isLeftChild = true;

        //find the node
        while (!current.getKey().equalsIgnoreCase(key)) {
            System.out.println("while in delete");
            parent = current;
            int comp = key.compareToIgnoreCase(current.getKey());
            if (comp < 0) {
                isLeftChild = true;
                if (!current.getLeftThread()) {
                    current = current.getLeft();
                }
            } else {
                isLeftChild = false;
                if (!current.getRightThread()) {
                    current = current.getRight();
                }
            }
            if (current.getLeftThread() && current.getRightThread()) {
                return;
            }
        }

        Node successor = getSuccessor(current);
```

Tree.java

```
if (!current.getLeftThread() && !current.getRightThread()) {
    System.out.println("1");
    if (current == root) {
        root = null;
    } else if (isLeftChild) {
        System.out.println("1lc");
        parent.setLeft(null);
    } else {//found big delete bug here, tried to fix
        System.out.println("1else");
        parent.setRight(successor); //
        successor.setLeft(current.getLeft());
    }
} else if (current.getRightThread() && !current.getLeftThread()) {
    System.out.println("2");
    if (current == root) {
        root = current.getLeft();
    } else if (isLeftChild) {
        parent.setLeft(current.getLeft());
    } else {
        parent.setRight(current.getLeft());
    }
} else if (current.getLeftThread() && !current.getRightThread()) {
    System.out.println("3");
    if (current == root) {
        root = current.getRight();
    } else if (isLeftChild) {
        parent.setLeft(current.getRight());
    } else {
        parent.setRight(current.getRight());
    }
} else {
    System.out.println("else in delete");
    if (current == root) {
        root = successor;
    } else if (isLeftChild) {
        parent.setLeft(successor);
    } else {
        parent.setRight(successor);
    }

    current.getRight().setLeft(current.getLeft());
    current.getRight().setLeftThread(true);
```

```
            current.getLeft().setRight(successor);
            current.getLeft().setRightThread(true);

        }

        nElems--;
    }

    private Node getSuccessor(Node delNode) {
        Node successorParent = delNode;
        Node successor = delNode;
        Node current = delNode.getRight();
        while (current != null) {
            successorParent = successor;
            successor = current;
            current = current.getLeft();
        }
        if (successor != delNode.getRight()) {
            successorParent.setLeft(successor.getRight());
            successor.setRight(delNode.getRight());
        }
        return successor;
    }

}
```