EE360T: Software Testing
Problem Set 7

Out: Nov 7, 2024; **Due: Nov 21, 2024 11:59pm**
Submission: *.zip via Canvas
Maximum points: 40

Recall the graph implementation from the previous graded homework. Observe that each graph represents a binary relation over numbers, e.g., the graph with `numNodes` = 2 and `edges` = {{true, false}, {false, true}} represents the relation {(0, 0), (1, 1)}, which is reflexive.

1 Graphs and properties of binary relations [20 points]

Consider the following partial implementation of graph data structure (which is a slight modification of the previous homework) to represent binary relations:

```
package ee360t.pset7;

import java.util.Arrays;
import java.util.Set;

public class Graph {
    private int numNodes; // number of nodes in the graph
    private boolean[][] edges;
    // edges[i][j] is true if and only if there is an edge from node i to node j

    // class invariant: edges != null; edges is a square matrix;
    //                  numNodes >= 0; numNodes is number of rows in edges

    public Graph(int size) {
        numNodes = size;
        edges = new boolean[size][size];
    }

    @Override
    public String toString() {
        return "numNodes: " + numNodes + "; " + "edges: " + Arrays.deepToString(edges);
    }

    @Override
    public boolean equals(Object o) {
        if (o.getClass() != ee360t.pset7.Graph.class) return false;
        return toString().equals(o.toString());
    }

    @Override
    public int hashCode() {
        // your code goes here
        // ...

    }

    public void addEdge(int from, int to) {
        // postcondition: adds a directed edge "from" -> "to" to this graph
        edges[from][to] = true;
    }

    public int numEdges() {
        // post: returns the number of edges in this

        // your code goes here
        // ...

    }

    public boolean hasExactlyOneEdge() {
        // post: returns true if and only if there is exactly one edge in this

        // your code goes here
        // ...

    }

    public boolean isReflexive() {
        // post: returns true if this represents a reflexive relation

        // your code goes here
        // ...

    }

    public boolean isSymmetric() {
        // post: returns true if and only if this represents a symmetric relation

        // your code goes here
        // ...

    }

    public boolean isTransitive() {
        // post: returns true if and only if this represents a transitive relation

        // your code goes here
        // ...

    }
}
```

1.1 Implementing `hashCode` [2 points]

Implement the method `hashCode`. Make sure your implementation satisfies the contract Java language enforces regarding `hashCode` and `equals` methods.

1.2 Implementing `numEdges` [4 points]

Implement the method `numEdges` as specified.

1.3 Implementing `hasExactlyOneEdge` [2 points]

Implement the method `hasExactlyOneEdge` as specified.

1.4 Implementing `isReflexive` [3 points]

Implement the method `isReflexive` as specified.

1.5 Implementing `isSymmetric` [4 points]

Implement the method `isSymmetric` as specified.

1.6 Implementing `isTransitive` [5 points]

Implement the method `isTransitive` as specified.

2 Generating binary relations [20 points]

The following code gives a partial implementation of a graph generator that creates all graphs that have the given size and represent binary relations with the given properties:

```
package ee360t.pset7;

import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.HashSet;
import java.util.Set;

public class GraphGenerator {
    public static Set<Graph> generateAllGraphs(int num) {
        // pre: num >= 0
        // post: returns a set of all graphs that have num nodes

        // your code goes here
        // ...

    }

    public static Set<Graph> generateAllGraphs(int num, String property) {
        // pre: num >= 0 and
        //      property is the name of a valid boolean method in class Graph
        // post: returns a set of all graphs (with num nodes) that represent binary
        //       relations with the given property

        return generateAllGraphs(num, new String[]{ property });
    }

    public static Set<Graph> generateAllGraphs(int num, String[] properties) {
        // pre: num >= 0 and
        //      each element of properties is the name of a valid boolean method in class Graph
        // post: returns a set of all graphs (with num nodes) that represent binary
        //       relations with all the given properties

        // your code goes here
        // ...

    }
}
```

2.1 Generating all graphs with given size

Implement the method `generateAllGraphs` with one parameter (`num`) as specified. You may find it useful to observe that each k-node graph that represents a binary relation can be encoded using a binary string of length $k \times k$, e.g., the graph with `numNodes` = 3 and `edges` = {{false, true, true}, {true, false, true}, {false, true, false}} can be encoded as "011101010". Therefore, your graph generator could simply first generate all binary strings of length equal to `num`^{2}, and then for each string, initialize boolean matrix `edges` to create the corresponding graph.

2.2 Generating all graphs with given size and properties

Implement the method `generateAllGraphs` with two parameters (`num` and `properties`) as specified. As a general strategy, consider first creating all graphs with `num` nodes (using your implementation from Question 2.1), and then checking each graph with respect to the given `properties` (using the boolean methods you implemented in Question 1) so graphs that violate any of the `properties` are omitted from the result. Use Java reflection API to invoke these boolean methods.

3 Tests

To help with testing (and debugging) your code, the following code contains a (small) test suite:

```
package ee360t.pset7;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class GraphGenTester {
    @Test public void theoe0() {
        assertEquals(1, GraphGenerator.generateAllGraphs(1, "hasExactlyOneEdge").size());
    }

    @Test public void theoe1() {
        assertEquals(4, GraphGenerator.generateAllGraphs(2, "hasExactlyOneEdge").size());
    }

    @Test public void theoe2() {
        assertEquals(9, GraphGenerator.generateAllGraphs(3, "hasExactlyOneEdge").size());
    }

    @Test public void tir0() {
        assertEquals(1, GraphGenerator.generateAllGraphs(1, "isReflexive").size());
    }

    @Test public void tir1() {
        assertEquals(4, GraphGenerator.generateAllGraphs(2, "isReflexive").size());
    }

    @Test public void tir2() {
        assertEquals(64, GraphGenerator.generateAllGraphs(3, "isReflexive").size());
    }

    @Test public void tis0() {
        assertEquals(2, GraphGenerator.generateAllGraphs(1, "isSymmetric").size());
    }

    @Test public void tis1() {
        assertEquals(8, GraphGenerator.generateAllGraphs(2, "isSymmetric").size());
    }

    @Test public void tis2() {
        assertEquals(64, GraphGenerator.generateAllGraphs(3, "isSymmetric").size());
    }

    @Test public void titr0() {
        assertEquals(2, GraphGenerator.generateAllGraphs(1, "isTransitive").size());
    }

    @Test public void titr1() {
        assertEquals(13, GraphGenerator.generateAllGraphs(2, "isTransitive").size());
    }
    @Test public void titr2() {
        assertEquals(171, GraphGenerator.generateAllGraphs(3, "isTransitive").size());
    }

    @Test public void theoeir() {
        assertEquals(0, GraphGenerator.generateAllGraphs(3,
                new String[]{"hasExactlyOneEdge", "isReflexive"}).size());
    }

    @Test public void theoeis() {
        assertEquals(3, GraphGenerator.generateAllGraphs(3,
                new String[]{"hasExactlyOneEdge", "isSymmetric"}).size());
    }

    @Test public void theoeitr() {assertEquals(9, GraphGenerator.generateAllGraphs(3,
                new String[]{"hasExactlyOneEdge", "isTransitive"}).size());
    }

    @Test public void teq0() {
        assertEquals(1, GraphGenerator.generateAllGraphs(1,
                new String[]{"isReflexive", "isSymmetric", "isTransitive"}).size());
    }

    @Test public void teq1() {
        assertEquals(2, GraphGenerator.generateAllGraphs(2,
                new String[]{"isReflexive", "isSymmetric", "isTransitive"}).size());
    }

    @Test public void teq2() {
        assertEquals(5, GraphGenerator.generateAllGraphs(3,
                new String[]{"isReflexive", "isSymmetric", "isTransitive"}).size());
    }
}
```