Introduction to Software Engineering

Assignment 9

Dimitri Tabatadze, 2024-12-11

1 Functional Test

Consider the following interface

```java
public interface Account {
  private BigDecimal balance;
  //...

  /**
   * Withdraws the amount specified, provided balance+creditLine >= amount
   *
   * @param amount
   * The amount to be withdrawn (>= 0)
   * @param creditLine
   * The credit line of the account (how much one can borrow) (>= 0)
   * @return true if withdrawal is possible, false otherwise
   */
  public boolean withdraw(BigDecimal amount, BigDecimal creditLine);
  //...
}
```

a) Determine the equivalence classes of method `withdraw()` and provide representatives. Describe each equivalence class in a short sentence.

b) Determine the boundaries of your equivalence classes. Which additional values should be tested when taking the boundaries into account?

Solution

a) We must use the assertions

- `amount >= 0`,
- `creditLine >= 0`.

This gives the equivalence classes:

Equiv Class	Description	Possible values
$\mathrm{EC}_{\texttt{amount},I}$	Invalid `amount`	`amount` < 0
$\mathrm{EC}_{\texttt{amount},V}$	Valid `amount`	`amount` ≥ 0
$\mathrm{EC}_{\texttt{creditLine},I}$	Invalid `creditLine`	`creditLine` < 0
$\mathrm{EC}_{\texttt{creditLine},V}$	Valid `creditLine`	`creditLine` ≥ 0

b) The boundaries are clear:

- $\mathrm{EC}_{\texttt{amount},I}$ – only relevant boundary is exclusive 0, should test for $-1 \in \mathrm{EC}_{\texttt{amount},I}$ and $0 \notin \mathrm{EC}_{\texttt{amount},I}$.
- $\mathrm{EC}_{\texttt{amount},V}$ – only relevant boundary is inclusive 0, should test for $-1 \notin \mathrm{EC}_{\texttt{amount},V}$ and $0 \in \mathrm{EC}_{\texttt{amount},V}$.
- $\mathrm{EC}_{\texttt{creditLine},I}$ – only relevant boundary is exclusive 0, should test for $-1 \in \mathrm{EC}_{\texttt{creditLine},I}$ and $0 \notin \mathrm{EC}_{\texttt{creditLine},I}$.
- $\mathrm{EC}_{\texttt{creditLine},V}$ – only relevant boundary is inclusive 0, should test for $-1 \notin \mathrm{EC}_{\texttt{creditLine},V}$ and $0 \in \mathrm{EC}_{\texttt{creditLine},V}$.

> **Note:** Bless the guy who invented copy-pasting.

2 Control flow-oriented test

Consider the method `isIdentity()`.

```java
public boolean isIdentity(double[][] mtx) {
  if (mtx == null || mtx.length != mtx[0].length) {
    return false;
  }
  int n = mtx.length;

  for (int r = 0; r < n; r++) {
    for (int c = 0; c < n; c++) {
      if (r == c && Math.abs(mtx[r][c] - 1) >= 1E-8) {
        return false;
      } else if (r != c & Math.abs(mtx[r][c]) >= 1E-8) {
        return false;
      }
    }
  }
  return true;
}
```

a) Translate the code of `isIdentity(double[][] mtx)` into the intermediate language presented in class.

b) Draw the control flow graph for `isIdentity(double[][] mtx)`. Enter the source code statements into the boxes and number the boxes. (Only entering the line numbers is not sufficient.)

c) Provide a minimal test case set and the paths traversed by them, such that statement coverage is achieved. Do the same for branch coverage. If it is not possible to achieve full coverage, explain why.

> **Hint:** “Minimal” means the test case set must contain enough test cases to achieve coverage, and not more.

d) Which test cases do you need for boundary-interior test? Explain how you would handle the nested loops in this case.

e) Explain why path coverage for `isIdentity(double[][] mtx)` is not practicable.

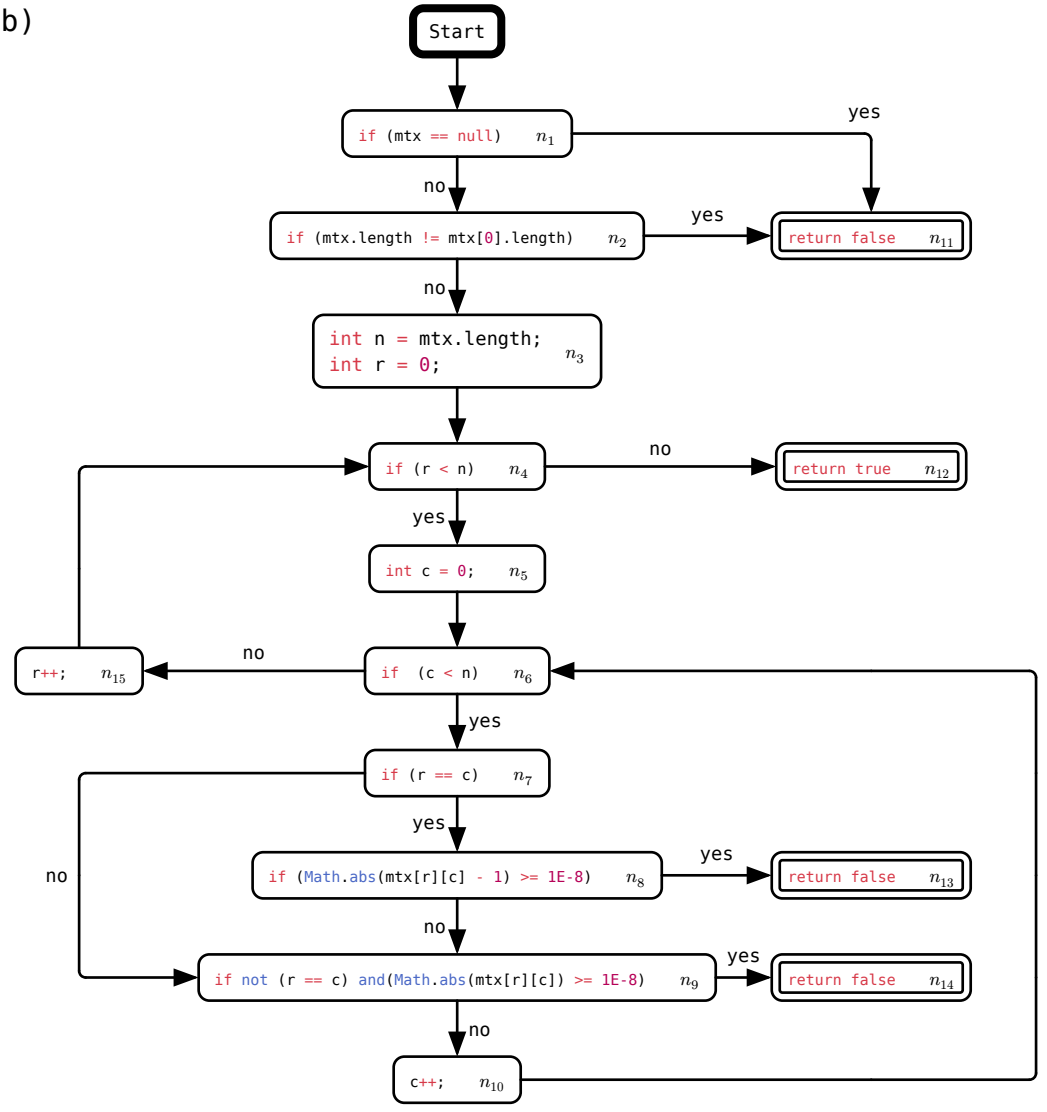
Solution

a) The translated code:

```java
if (mtx == null) goto 04;
if (mtx.length != mtx[0].length) goto 04;
```

```
 3  goto 05;
 4  return false;
 5  int n = mtx.length;
 6  int r = 0;
 7  if not (r < n) goto 20;
 8  int c = 0;
 9  if not (c < n) goto 18;
10  if (r == c) goto 13;
11  if (Math.abs(mtx[r][c] - 1) >= 1E-8) goto 13;
12  return false;
13  if not (r == c) and (Math.abs(mtx[r][c]) >= 1E-8) goto 15;
14  goto 16;
15  return false;
16  c++;
17  goto 09;
18  r++;
19  goto 07;
20  return true;
```

b)



c)

test	covered satatements
null	n_1, n_{11}
$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$	$n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_{13}$
$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	$n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9, n_{10}, n_{14}$
$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9, n_{10}, n_{15}, n_{12}$

Table 1: Statement coverage

test	covered satatements
null	n_1, n_{11}
$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$	n_1, n_2, n_{11}
$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$	$n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_{13}$
$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	$n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9, n_{10}, n_6, n_7, n_9, n_{14}$
$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9, n_{10}, n_6, n_7, n_9, n_{10}, n_6, n_{15},$ $n_4, n_5, n_6, n_7, n_8, n_9, n_{10}, n_6, n_7, n_9, n_{10}, n_6, n_{15}, n_4, n_{12}$

Table 2: Branch coverage

d) What is boundary-interior test?

e) For every element of a matrix there's a three way condition (if, else-if, else).

For an $n \times n$ matrix, there are n^2 elements in the matrix and to cover all paths, we would need the total of 3^{n^2} tests. Since the amount of tests is dependent on n and we don't have n fixed, that means that we would have to write tests for every possible n.

Assuming that $n \in \mathbb{N}, n < N$, we can calculate the total number of tests by:

$$\#\text{Tests} = \sum_{i=1}^{N} 3^{i^2} \tag{1}$$

This number grows way too fast compared to N. Already for $N = 5$ we would have $\#\text{Tests} > 8 \cdot 10^{11}$.

3 Code Inspections

Inspect the class `Matrix` below for issues. An issue is a violation of one of the rules in the checklist following the code. Record each issue in a table of the following form

Rule	Lines	Short description

If you discover an issue, enter the violated rule, the line number in the source text, and a short description of the issue.

```java
package org.iMage.HDrize.matrix;

import org.iMage.HDrize.base.matrix.IMatrix;
import org.ojalgo.matrix.store.PrimitiveDenseStore;

/**
 * A matrix.
 *
 */
public final class Matrix implements IMatrix {

  private final int rows;
  private final int cols;

  final double[][] data;
  /**
   * Create a new matrix.
   *
   * @param m the original matrix
   */
  public Matrix(IMatrix m) {
    this(m.copy());
  }

  /**
   * Create a new matrix.
   *
   */
  private Matrix(double[][] m) {
    this(m.length, m.length == 0 ? 0 : m[0].length);
    for (int r = 0; r < this.rows; r++) {
      if (m[r].length != this.cols) {
        throw new IllegalArgumentException("Rows have not equal lengths.");
      }
      for (int c = 0; c < this.cols; c++)
        this.set(r, c, m[r][c]);
    }

  }

  /**
   * Create a matrix (only zeros).
   *
   * @param rows the amount of rows
   */
  public Matrix(int rows, int cols) {
    if (rows < 1 || cols < 1) {
      throw new IllegalArgumentException("Rows and cols have to be >= 1");
    }
    this.rows = rows;
    this.cols = cols;
    this.data = new double[rows][cols];
```

```
53    }
54
55    @Override
56    public double[][] copy() {
57      double[][] copy = new double[this.rows][this.cols];
58      for (int r = 0; r < this.rows; r++) {
59        for (int c = 0; c < this.cols; c++) {
60          copy[r][c] = this.data[r][c];
61        }
62      }
63      return copy;
64    }
65
66    @Override
67    public int rows() {
68       return this.rows;
69    }
70
71    @Override
72    public int cols() {
73        return this.cols;
74    }
75    @Override
76    public void set(int r, int c, double v) {
77      this.data[r][c] = v;
78
79    }
80
81    @Override
82    public double get(int r, int c) {
83      return this.data[r][c];
84    }
85
86    @Override
87    public String toString() {
88      StringBuilder builder = new StringBuilder();
89      String closeParen = "}";
90      builder.append("{\n");
91      for (int r = 0; r < this.rows(); r++) {
92          builder.append("{");
93          for (int c = 0; c < this.cols(); c++) {
94           builder.append(this.get(r, c));
95           if (c != this.cols() - 1) {
96               builder.append(", ");
97           }
98         }
99         builder.append("}\n");
100       }
101       builder.append("}");
102       return builder.toString();
103     }
104   }
105 }
```

Rules

1. Variable and Constant Declarations

10. Are all identifiers intelligible and in accordance with the Java naming conventions?
11. Are the types of all variables and constants correct?
12. Are all variables and constants correctly initialized?
13. Are there literals that should be declared as constants?
14. Are there local variables, instance or class variables that should be constant?
15. Are there class or instance variables that should be local?
16. Have class and instance variables the appropriate visibility? (default, private, protected, public)?
17. Are there instance variables that should be static or vice versa?
18. Are there libraries that are not used anywhere?

2. Method and Constructor Declarations

20. Are all identifiers intelligible and in accordance with the Java naming conventions?
21. Is every input parameter checked for correct value?
22. Does every return-statement deliver the correct value?
23. Have all methods and constructors the appropriate visibility? (default, private, protected, public)?
24. Are there instance methods or constructors that should be static or vice versa?

3. Comments

30. Have all classes and all non-private constructors, methods, constants, class and instance variables a complete JavaDoc comment?
31. Do comments describe the associated code sections correctly?

4. Layout

40. Is indentation correct and consistent?
41. Do all blocks have curly braces?

5. Performance

50. Is it possible to cache values rather than computing them repeatedly?
51. Is every computed value used?
52. Can computations be moved out of loops?

Solution

Team members: Giorgi Bakradze, Giorgi Otinashivli

Rule	Lines	Short description
10	21-23, 29-36, 76-77	Variables `m`, `r`, `v`, and `c` could be made more intelligible
10	89	Variable name `closeParen` is not appropriate for a brace
16	15	The variable `data` should be private
18	89	The variable `closeParen` is not used anywhere
20	67,72,76	Getters and setter should use `getRows()`, `getCols()` and `setElement(rows, cols, val)`
21	76,82	The parameters `r` and `c` may be out of bounds
30	10	The javadoc doesn’t document the class `Matrix`
30	46	The javadoc doesn’t document the parameter `cols`
30	56	There is no javadoc that documents the method `copy()`
31	25-39	The comment doesn’t describe the method *completely* but it is *correct*
40	68,73, 92-104	Indentation is not consistent
41	35	The for loop doesn’t have curly braces
50	87	The `toString()` method can be cached