

Testare si Verificare – Proiect laborator – Raport

Coteanu Vlad – 506 Inginerie Software

1. Descriere problema

Se da un sir de lungime N ($2 \leq N \leq 100$) cu numere naturale. Verificati daca inversul ultimului numar apare printre celelalte elemente din sir.

De exemplu, pentru intrarea (7, [10, 321, 5, 23, 12, 17, 123]), rezultatul asteptat este true, iar pentru intrarea (4, [3, 5, 12, 89]) rezultatul asteptat este false.

Codul care rezolva problema aleasa este urmatorul:

```
public boolean solve() {  
    if(numberOfElements < 2 || numberOfElements > 100) {  
        throw new RuntimeException("The number of elements is outside the interval  
[2, 100].");  
    }  
  
    if(input == null) {  
        throw new RuntimeException("Input array is null.");  
    }  
  
    if(input.length != numberOfElements) {  
        throw new RuntimeException("Input array length is different than the N  
given.");  
    }  
  
    for(int i = 0; i < numberOfElements; i++) {  
        if(input[i] < 0) {  
            throw new RuntimeException("The element on position " + i + " is not a  
natural number.");  
        }  
    }  
  
    int lastElement = input[numberOfElements - 1];  
    int lastElementMirror = getMirror(lastElement);  
    for(int i = 0; i < numberOfElements - 1; i++) {  
        if(input[i] == lastElementMirror) {  
            return true;  
        }  
    }  
    return false;  
}
```

```
private int getMirror(int x) {
    int y = 0;

    while(x != 0) {
        y = y * 10 + x % 10;
        x /= 10;
    }

    return y;
}
```

2. Testarea functionala

2.1. Partitie in clase de echivalenta

Modul in care a fost facuta partitia datelor in clase de echivalenta este prezentat in liniile urmatoare:

```
/**
 * Input values:
 * n - between 2 and 100 -> therefore, there are 3 different partitions:
 *     n < 2
 *     2 <= n <= 100
 *     100 < n
 *
 * inputData - array that should have the same length with n and have positive values
 *             therefore I distinguished 5 partitions:
 *     inputData is null
 *     inputData is not null and does not have the length equal to n
 *     inputData is not null, has the length n, but does not have all the values
positive
 *     inputData is not null, has the length n and has only positive values, but the
reverse of the last number is
 *         not equal to any other element in the array, such that the result is
False
 *     inputData is not null, has the length n, has only positive numbers, and the
reverse of the last number is
 *         equal to at least one of the other elements in the array. such that
the result is True
 *
 * Considering the above possibilities, there are 5 possible cases ( n in [2, 100] x
inputData possibilities ) plus
 * 2 cases when n is outside the boundaries => 7 equivalence partitions.
 */
```

De asemenea, datele de test folosite sunt:

```
// 1. n < 2
Problem problem1 = new Problem(0, new int[] {1, 2, 3, 4}, "N < 2");
Result result1 = Result.makeErrorResult("The number of elements is outside the
interval [2, 100].");
testData.add(new Pair<Problem, Result>(problem1, result1));

// 2. n > 100
Problem problem2 = new Problem(123, new int[] {1, 2, 3, 4}, "N > 100");
Result result2 = Result.makeErrorResult("The number of elements is outside the
interval [2, 100].");
testData.add(new Pair<Problem, Result>(problem2, result2));

// 3. n inside [2, 100], inputData is null
Problem problem3 = new Problem(10, null, "N inside [2, 100] and null input data");
Result result3 = Result.makeErrorResult("Input array is null.");
testData.add(new Pair<Problem, Result>(problem3, result3));

// 4. n inside [2, 100], inputData length is different than n
Problem problem4 = new Problem(3, new int[] {1, 2, 3, 4, 5, 6}, "N inside [2, 100]
and input data of invalid length");
Result result4 = Result.makeErrorResult("Input array length is different than the N
given.");
testData.add(new Pair<Problem, Result>(problem4, result4));

// 5. n inside [2, 100], inputData has length n but has negative values
Problem problem5 = new Problem(6, new int[] {1, 2, -3, 4, -5, 6}, "N inside [2, 100]
and input data with negative values");
Result result5 = Result.makeErrorResult("The element on position 2 is not a natural
number.");
testData.add(new Pair<Problem, Result>(problem5, result5));

// 6. n inside [2, 100], inputData has length n, positive values, but the reverse of
the last number is not
// equal to any other element inside the array
Problem problem6 = new Problem(6, new int[] {10, 20, 30, 40, 50, 60}, "N inside [2,
100] and the result is false");
Result result6 = Result.makeNormalResult(false);
testData.add(new Pair<Problem, Result>(problem6, result6));

// 7. n inside [2, 100], inputData has length n, positive values, and the reverse of
the last number is equal
// to at least one other number inside the array
Problem problem7 = new Problem(6, new int[] {10, 11, 12, 13, 14, 31}, "N inside [2,
100] and the result is true");
Result result7 = Result.makeNormalResult(true);
testData.add(new Pair<Problem, Result>(problem7, result7));
```

2.2. Analiza valorilor limita

Modul in care au fost determinate valorile limita este prezentat in liniile urmatoare:

```
/**
 * For generating test data based on boundary analysis, we should consider the
 following possibilities:
 * n, which should belong to the interval [2, 100]:
 *   - values 0 and 1, for the case when n < 2
 *   - values 2, 50 and 100, for the case when n belong to the interval
 *   - values 101, 150, for the case when n is outside the interval
 * inputData - only valid, this time, with the reverse of the last number existing
 on:
 *   - first position in the array
 *   - before-the-last position in the array
 *
 * Therefore, there are 3 x 2 cases for a valid n, and 4 more cases when n is outside
 the boundaries
 * However, when n = 2, first position of the array is equal to before-the-last
 position, so, 9 testData items
 */
```

Datele de test alese si rezultatele asteptate sunt:

```
// 1. n is 0
Problem problem1 = new Problem(0, new int[] {}, "N is less than the lower bound");
Result result1 = Result.makeErrorResult("The number of elements is outside the
interval [2, 100].");
testData.add(new Pair<Problem, Result>(problem1, result1));

// 2. n is 1
Problem problem2 = new Problem(1, new int[] {1}, "N is less than the lower bound");
Result result2 = Result.makeErrorResult("The number of elements is outside the
interval [2, 100].");
testData.add(new Pair<Problem, Result>(problem2, result2));

// 3. n is 2, solution is on the first position in the array -> which is the same
with the solution being
// on the before-the-last position in the array
Problem problem3 = new Problem(2, new int[] {123, 321}, "N is equal to the lower
bound, solution is on the position 0");
Result result3 = Result.makeNormalResult(true);
testData.add(new Pair<Problem, Result>(problem3, result3));

// 4. n is 50, the reverse of the last number problem is on position 0
List<Integer> dummyData1 = new ArrayList<>();
for(int i = 0; i < 50; i++) {
    dummyData1.add(i + 1);
}
dummyData1.set(0, 321);
dummyData1.set(49, 123);
Problem problem4 = new Problem(50, dummyData1.stream().mapToInt(i -> i).toArray(),
    "N is inside the expected bounds, solution is on the position 0");
```

```

Result result4 = Result.makeNormalResult(true);
testData.add(new Pair<Problem, Result>(problem4, result4));

// 5. n is 50, the reverse of the last number problem is on position 48
List<Integer> dummyData2 = new ArrayList<>();
for(int i = 0; i < 50; i++) {
    dummyData2.add(i + 1);
}
dummyData2.set(48, 321);
dummyData2.set(49, 123);
Problem problem5 = new Problem(50, dummyData2.stream().mapToInt(i -> i).toArray(),
    "N is inside the expected bounds, solution is on the before-the-last position");
Result result5 = Result.makeNormalResult(true);
testData.add(new Pair<Problem, Result>(problem5, result5));

// 6. n is 100, the reverse of the last number problem is on position 0
List<Integer> dummyData3 = new ArrayList<>();
for(int i = 0; i < 100; i++) {
    dummyData3.add(i + 1);
}
dummyData3.set(0, 321);
dummyData3.set(99, 123);
Problem problem6 = new Problem(100, dummyData3.stream().mapToInt(i -> i).toArray(),
    "N is equal to the higher bound, solution is on the position 0");
Result result6 = Result.makeNormalResult(true);
testData.add(new Pair<Problem, Result>(problem6, result6));

// 7. n is 100, the reverse of the last number problem is on position 98
List<Integer> dummyData4 = new ArrayList<>();
for(int i = 0; i < 100; i++) {
    dummyData4.add(i + 1);
}
dummyData4.set(98, 321);
dummyData4.set(99, 123);
Problem problem7 = new Problem(100, dummyData4.stream().mapToInt(i -> i).toArray(),
    "N is equal to the higher bound, solution is on the position 99");
Result result7 = Result.makeNormalResult(true);
testData.add(new Pair<Problem, Result>(problem7, result7));

// 8. n is 101
Problem problem8 = new Problem(101, new int[] {123, 321}, "N is greater than the
higher bound.");
Result result8 = Result.makeErrorResult("The number of elements is outside the
interval [2, 100].");
testData.add(new Pair<Problem, Result>(problem8, result8));

// 9. n is 150
Problem problem9 = new Problem(150, new int[] {123, 321}, "N is greater than the
higher bound.");
Result result9 = Result.makeErrorResult("The number of elements is outside the
interval [2, 100].");
testData.add(new Pair<Problem, Result>(problem9, result9));

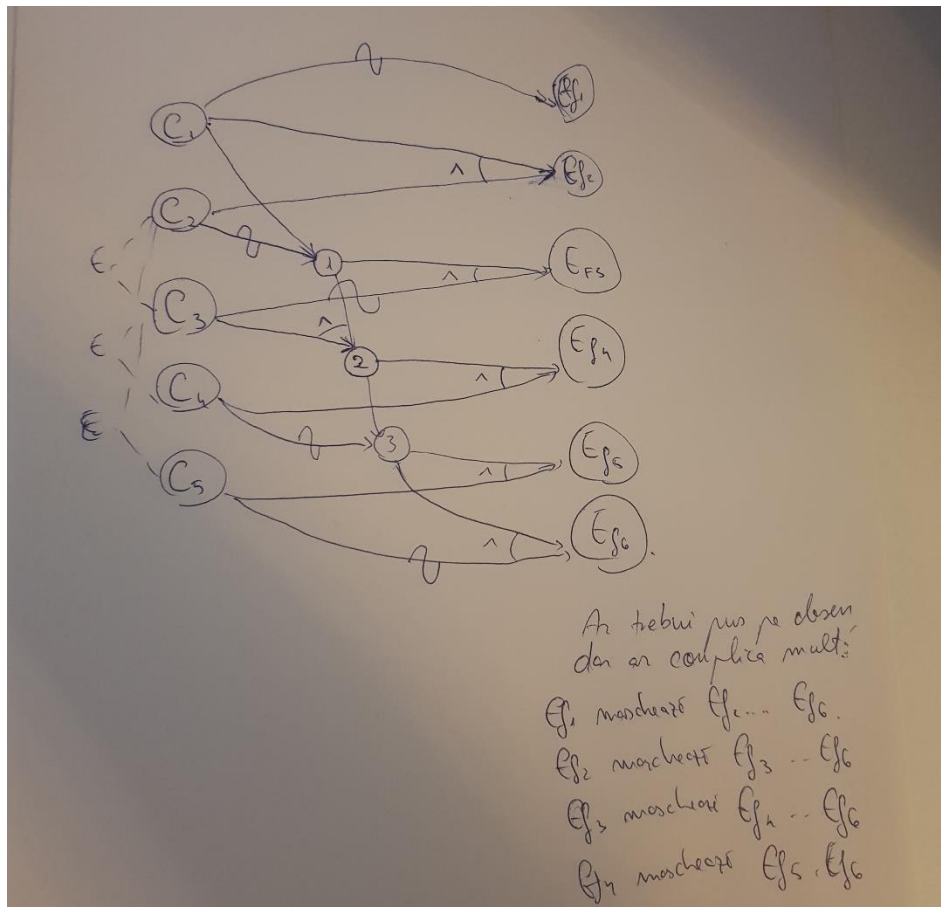
```

2.3. Graful cauza-efect

Cauzele si efectele identificate sunt:

```
/**
 * The causes that need to be considered are:
 * C1: N is inside [2, 100]
 * C2: inputArray is null
 * C3: inputArray has the length equal to N
 * C4: inputArray has negative values
 * C5: There is an element equal to the reverse of the Last number in the array
 *
 * Ef1: RuntimeException: The number of elements is outside the interval [2, 100].
 * Ef2: RuntimeException: Input array is null.
 * Ef3: RuntimeException: Input array length is different than the N given.
 * Ef4: RuntimeException: The element on position _ is not a natural number.
 * Ef5: true
 * Ef6: false
```

Graful este reprezentat in imaginea urmatoare:



Tabelul de decizie si procesul de creare al lui sunt prezentate in urmatoarea imagine:

C_1	0	1	1	1	1	1
C_2	0	1	0	0	0	0
C_3	0	0	0	1	1	1
C_4	0	0	0	1	0	0
C_5	0	0	0	0	1	0
G_1	1	0	0	0	0	0
G_2	0	1	0	0	0	0
G_3	0	0	1	0	0	0
E_1	0	0	0	1	0	0
E_5	0	0	0	0	1	0
E_6	0	0	0	0	0	1

Step 1: $\neg C_1 \Rightarrow 1 \Rightarrow \neg E_1 \wedge \neg E_2, E_1 \wedge \neg E_3, E_1 \wedge \neg E_5, E_1 \wedge \neg E_6$

\vee_1

C_1	C_2	C_3	C_4	C_5	E_1	E_2	E_3	E_4	E_5	E_6
1	0	0	0	0	1	0	0	0	0	0

Step 2: $E_2 = 1 \Rightarrow C_2 = 1$, Polomul marcate, si catelele inlaturate

$C_1 \Rightarrow \neg E_1$

\vee_2

C_1	C_2	C_3	C_4	C_5	E_1	E_2	E_3	E_4	E_5	E_6
1	1	0	0	0	0	1	0	0	0	0

Step 3: $E_3 = 1 \Rightarrow \begin{cases} C_3 = 0 \\ C_2 = 0 \\ C_1 = 1 \end{cases}$ — " — $\begin{cases} C_1 = \neg E_1 \\ C_2 = \neg E_2 \end{cases}$

\vee_3

C_1	C_2	C_3	C_4	C_5	E_1	E_2	E_3	E_4	E_5	E_6
1	0	0	0	0	0	0	1	0	0	0

Step 4: $G_4 = 1 \Rightarrow \begin{cases} C_4 = 1 \\ C_3 = 1 \\ C_2 = 0 \\ C_1 = 1 \end{cases}$ — " — $\begin{cases} C_1 = \neg E_1 \\ C_2 = \neg E_2 \\ C_3 = \neg E_3 \end{cases}$

\vee_4

C_1	C_2	C_3	C_4	C_5	E_1	E_2	E_3	E_4	E_5	E_6
1	0	1	1	0	0	0	0	1	0	0

Step 5: $E_5 = 1 \Rightarrow \begin{cases} C_1 = 1 \\ C_2 = 0 \\ C_3 = 1 \\ C_4 = 0 \\ C_5 = 1 \end{cases}$ — " — $\begin{cases} C_1 = \neg E_1 \\ C_2 = \neg E_2 \\ C_3 = \neg E_3 \\ C_4 = \neg E_4 \end{cases}$

Step 6: $E_6 = 1 \Rightarrow \begin{cases} C_1 = 1 \\ C_2 = 0 \\ C_3 = 0 \\ C_4 = 0 \\ C_5 = 0 \end{cases}$ — " —

Astfel, am generat urmatoarele date de test:

```
// 1. C1 false -> producing Ef1
Problem problem1 = new Problem(0, new int[] {1, 2, 3, 4}, "C1 false -> Ef1 true");
Result result1 = Result.makeErrorResult("The number of elements is outside the
interval [2, 100].");
testData.add(new Pair<Problem, Result>(problem1, result1));

// 2. C1 true, C2 true -> producing Ef2
Problem problem2 = new Problem(10, null, "C1 true, C2 true -> Ef2 true");
Result result2 = Result.makeErrorResult("Input array is null.");
testData.add(new Pair<Problem, Result>(problem2, result2));

// 3. C1 true, C2 false, C3 false -> producing Ef3
Problem problem3 = new Problem(3, new int[] {1, 2, 3, 4, 5, 6}, "C1 true, C2 false,
C3 false -> Ef3 true");
Result result3 = Result.makeErrorResult("Input array length is different than the N
given.");
testData.add(new Pair<Problem, Result>(problem3, result3));

// 4. C1 true, C2 false, C3 true, C4 true -> producing Ef4
Problem problem4 = new Problem(6, new int[] {1, 2, -3, 4, -5, 6}, "C1 true, C2 false,
C3 true, C4 true -> Ef4 true");
Result result4 = Result.makeErrorResult("The element on position 2 is not a natural
number.");
testData.add(new Pair<Problem, Result>(problem4, result4));

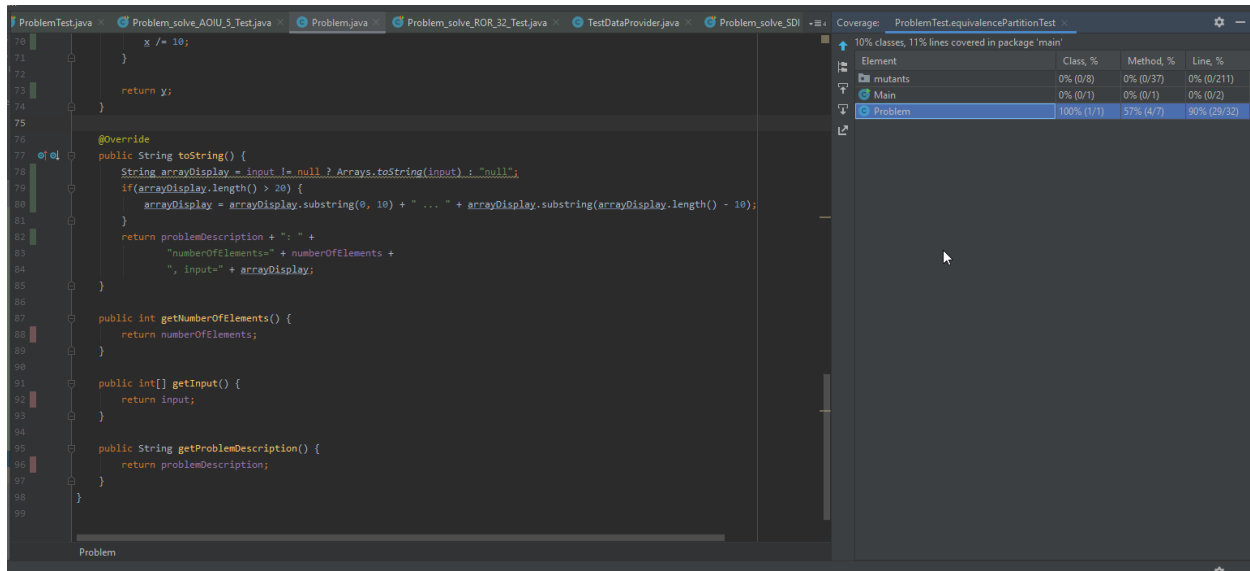
// 5. C1 true, C2 false, C3 true, C4 false, C5 true -> producing Ef5
Problem problem5 = new Problem(6, new int[] {16, 2, 3, 4, 5, 61}, "C1 true, C2 false,
C3 true, C4 false, C5 true -> Ef5 true");
Result result5 = Result.makeNormalResult(true);
testData.add(new Pair<Problem, Result>(problem5, result5));

// 6. C1 true, C2 false, C3 true, C4 false, C5 false -> producing Ef6
Problem problem6 = new Problem(6, new int[] {10, 11, 12, 13, 14, 38}, "C1 true, C2
false, C3 true, C4 false, C5 false -> Ef6 true");
Result result6 = Result.makeNormalResult(false);
testData.add(new Pair<Problem, Result>(problem6, result6));
```


2.4. Acoperirea datelor de test

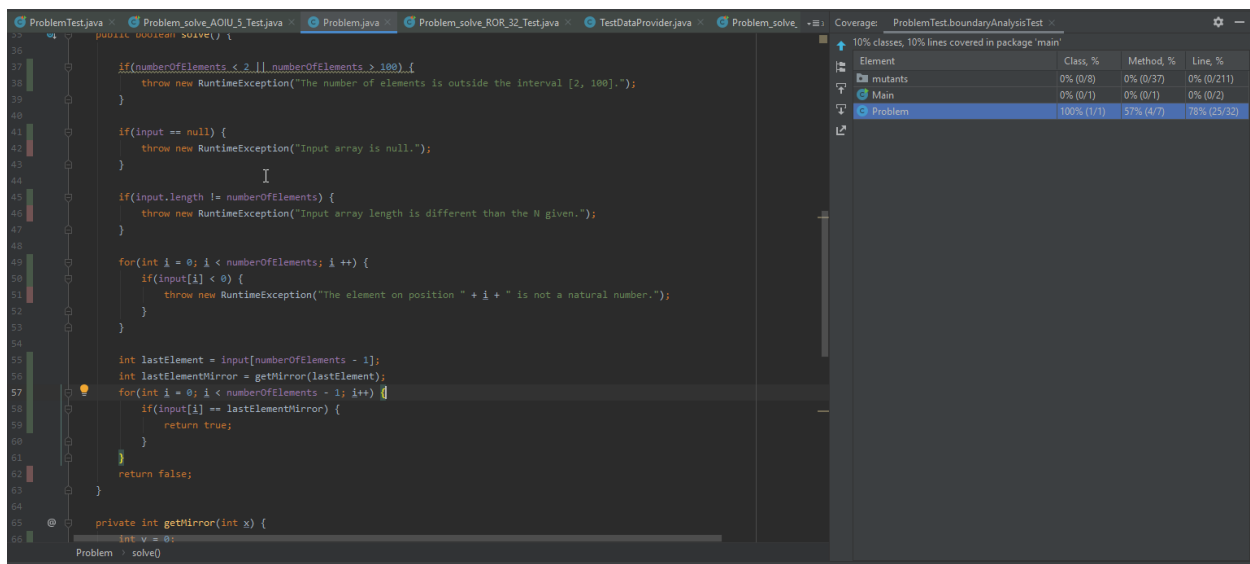
Nivelul de acoperire al datelor de test a fost calculate cu ajutorul tool-ului de code-coverage Jacoco, care este integrat in IntelliJ Idea.

Setul de teste pentru partiile de echivalenta au o acoperire de 57% a metodelor si de 90% a liniilor de cod. Totusi, coverage-ul a fost rulat la nivel de clasa. Daca ar fi sa analizam doar datele celor doua metode care ne intereseaza, avem un coverage de 100%



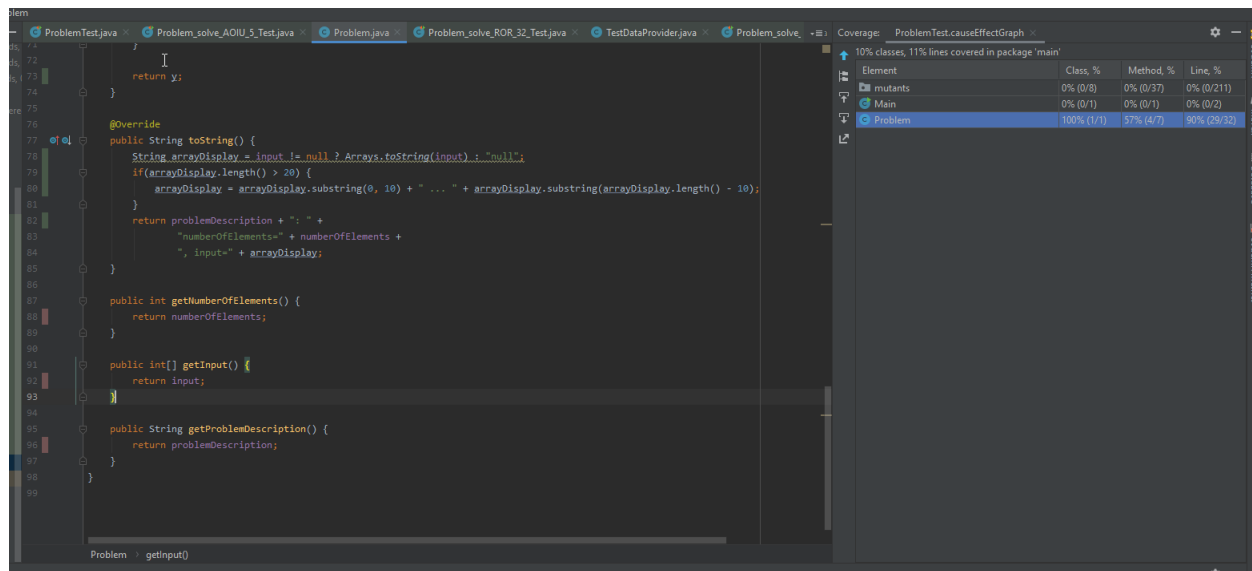
Element	Class, %	Method, %	Line, %
mutants	0% (0/8)	0% (0/37)	0% (0/211)
Main	0% (0/1)	0% (0/1)	0% (0/2)
Problem	100% (1/1)	57% (4/7)	90% (29/32)

In cazul testelor pentru analiza valorilor limita, avem o acoperire de 57% a metodelor si de 78% a liniilor de cod. Cauza este data de faptul ca nu avem teste care sa determine aruncarea exceptiilor "Input array is null" sau "The element on position _ is not a natural number". De asemenea, nu avem un test care sa aiba ca rezultat False.



Element	Class, %	Method, %	Line, %
mutants	0% (0/8)	0% (0/37)	0% (0/211)
Main	0% (0/1)	0% (0/1)	0% (0/2)
Problem	100% (1/1)	57% (4/7)	78% (25/32)

In cazul testelor pentru graful cauza-efect, avem exact aceleasi rezultate de acoperire ca si la metoda impartirii in clase de echivalenta.



3. Testarea structurala

Graful in care a fost transformat programul este atasat ca Anexa 1 la acest raport.

Testele, explicatia pentru alegerea lor si rezultatele lor sunt prezentate in liniile urmatoare:

```
/**
 * 1. Line 37: numberOfElements < 2 || numberOfElements > 100 as C1 || C2
 *    - we should consider C1 false C2 true, C1 true C2 false, C1 false C2 false
 *    - possible test data: (120, {1, 2, 3}), (1, {1, 2, 3}), (4, {1, 2, 3, 4})
 */
Problem problem1 = new Problem(120, new int[] {1, 2, 3}, "Line 37: 'numberOfElements < 2' is false, 'numberOfElements > 100' is true");
Result result1 = Result.makeErrorResult("The number of elements is outside the interval [2, 100].");
testData.add(new Pair<Problem, Result>(problem1, result1));

Problem problem2 = new Problem(1, new int[] {1, 2, 3}, "Line 37: 'numberOfElements < 2' is true, 'numberOfElements > 100' is false");
Result result2 = Result.makeErrorResult("The number of elements is outside the interval [2, 100].");
testData.add(new Pair<Problem, Result>(problem2, result2));

Problem problem3 = new Problem(4, new int[] {1, 2, 3, 4}, "Line 37: 'numberOfElements < 2' is false, 'numberOfElements > 100' is false");
Result result3 = Result.makeNormalResult(false);
testData.add(new Pair<Problem, Result>(problem3, result3));
```

```

/**
 * 2. Line 41: input == null as C1
 *     - we should consider C1 true and false
 *     - possible test data: (4, null), (4, {1, 2, 3, 4})
 *     - note: even if second test already exists, I am adding it again, with a
different description
 */
Problem problem4 = new Problem(4, null, "Line 41: 'input == null' is true");
Result result4 = Result.makeErrorResult("Input array is null.");
testData.add(new Pair<Problem, Result>(problem4, result4));

Problem problem5 = new Problem(4, new int[] {1, 2, 3, 4}, "Line 41: 'input == null'
is false");
Result result5 = Result.makeNormalResult(false);
testData.add(new Pair<Problem, Result>(problem5, result5));

/**
 * 3. Line 45: input.length != numberOfElements as C1
 *     - we should consider C1 true and false
 *     - possible test data: (4, {1, 2, 3, 4}), (4, {1, 2, 3})
 */
Problem problem6 = new Problem(4, new int[] {1, 2, 3}, "Line 45: 'input.length !=
numberOfElements' is true");
Result result6 = Result.makeErrorResult("Input array length is different than the N
given.");
testData.add(new Pair<Problem, Result>(problem6, result6));

Problem problem7 = new Problem(4, new int[] {1, 2, 3, 4}, "Line 45: 'input.length !=
numberOfElements' is false");
Result result7 = Result.makeNormalResult(false);
testData.add(new Pair<Problem, Result>(problem7, result7));

/**
 * 4. Line 49: i < numberOfElements
 *     - considering the problem statement that until this point, we should have at
least 2 elements in the array,
 *     this condition will always be true at least one time, for every test that has
true/false as result (problem7)
 *     - also, for every type of input that comes to a true/false result, the
condition will become false in the end
 *     - possible test data (4, {1, 2, 3, 4})
 */
Problem problem8 = new Problem(4, new int[] {1, 2, 3, 4}, "Line 49: 'i <
numberOfElements' is true and false");
Result result8 = Result.makeNormalResult(false);
testData.add(new Pair<Problem, Result>(problem8, result8));

/**
 * 5. Line 50: input[i] < 0
 *     - we should consider C1 true and false
 *     - possible test data: (4, {-1, 2, 3, 4}) (4, {1, 2, 3, 4}),
 */
Problem problem9 = new Problem(4, new int[] {-1, 2, 3, 4}, "Line 50: 'input[i] < 0'

```

```

is true");
Result result9 = Result.makeErrorResult("The element on position 0 is not a natural
number.");
testData.add(new Pair<Problem, Result>(problem9, result9));

Problem problem10 = new Problem(4, new int[] {1, 2, 3, 4}, "Line 50: 'input[i] < 0'
is false");
Result result10 = Result.makeNormalResult(false);
testData.add(new Pair<Problem, Result>(problem10, result10));

/**
 * 6. Line 68: x != 0
 *     - considering the purpose of this condition and the lines that follow it, if
we have any non-zero element
 *     on the last position, the condition will be true and false in the same
problem
 *     - possible test data (4, {1, 2, 3, 4})
 */

Problem problem11 = new Problem(4, new int[] {1, 2, 3, 4}, "Line 68: 'x != 0' is true
and false");
Result result11 = Result.makeNormalResult(false);
testData.add(new Pair<Problem, Result>(problem11, result11));

/**
 * 7. Line 57: i < numberOfElements - 1
 *     - considering that at this point we are assured that the for should iterate
trough at least one number,
 *     the condition will be true at least once in a problem that returns
true/false.
 *     - also, considering that we return early if the problem result is true, any
problem that will have a false
 *     result should have this condition false, at the end of the loop.
 *     - possible test data (4, {1, 2, 3, 4})
 */

Problem problem12 = new Problem(4, new int[] {1, 2, 3, 4}, "Line 57: 'i <
numberOfElements - 1' is true and false");
Result result12 = Result.makeNormalResult(false);
testData.add(new Pair<Problem, Result>(problem12, result12));

/**
 * 6. Line 58: input[i] == lastElementMirror
 *     - we should consider C1 true and false
 *     - possible test data (4, {12, 2, 3, 21}) (4, {1, 2, 3, 4})
 */

Problem problem13 = new Problem(4, new int[] {12, 2, 3, 21}, "Line 58: 'input[i] ==
lastElementMirror' is true");
Result result13= Result.makeNormalResult(true);
testData.add(new Pair<Problem, Result>(problem13, result13));

Problem problem14 = new Problem(4, new int[] {1, 2, 3, 4}, "Line 58: 'input[i] ==
lastElementMirror' is false");

```

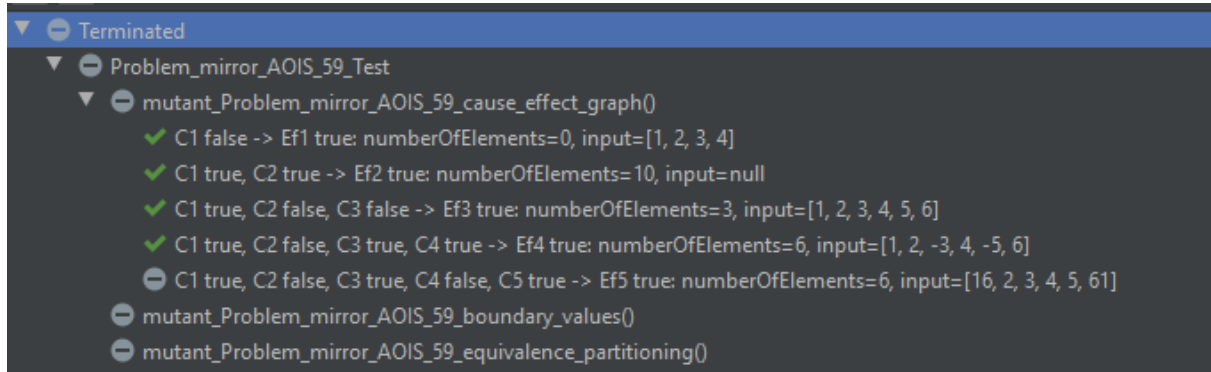
```
Result result14 = Result.makeNormalResult(false);  
testData.add(new Pair<Problem, Result>(problem14, result14));
```

Exista 6 noduri in grafin care, pe baza conditiei se poate inainte in doua directii diferite, deci sunt 6 conditii generatoare de teste.

4. Testarea mutationala

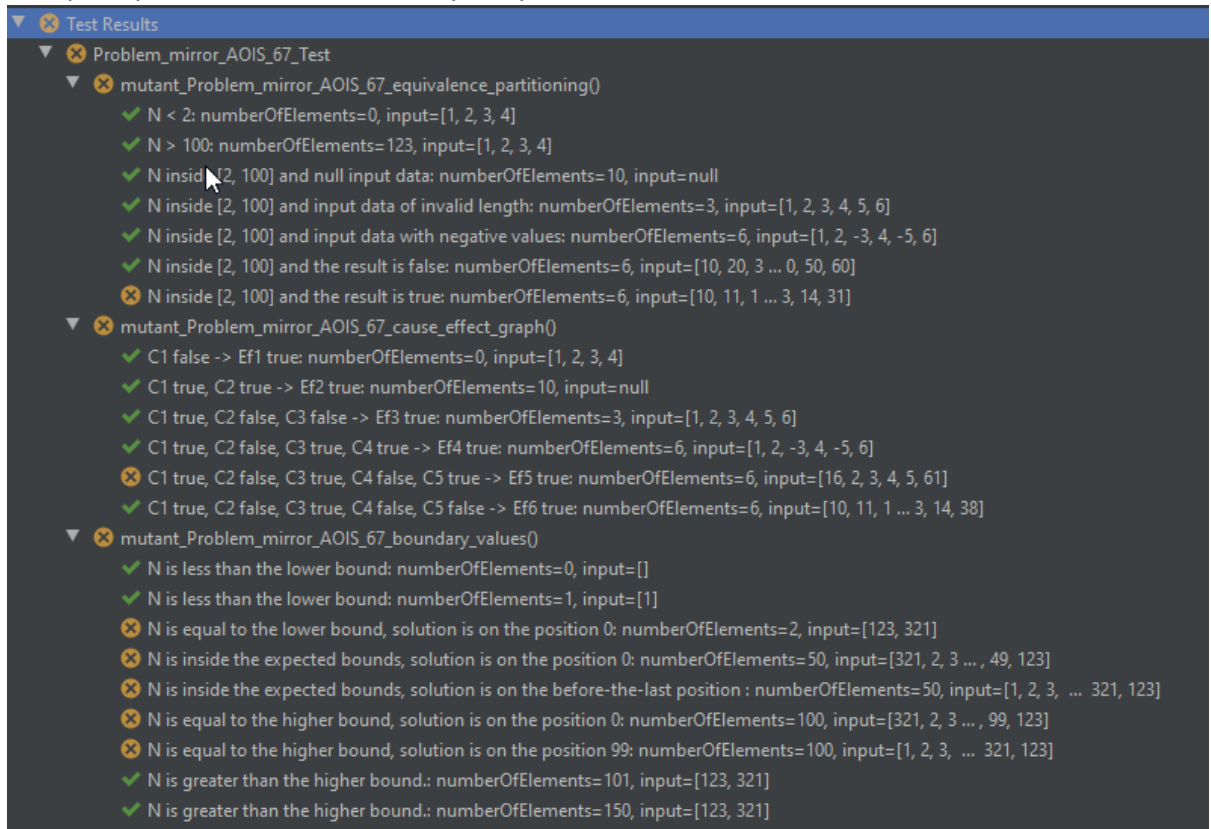
Pentru generarea mutantilor, am folosit MuJava. Din setul mare de mutant posibili generate, am selectat 8 mutanti pe care am rulat setul de teste de la punctul 1.

1. Mutantul **Problem_mirror_AOIS_59** a fost creat prin schimbarea liniei 68 din functia getMirror din `while(x != 0)` in `while(++x != 0)`. Rezultatul rularii testelor:



Se poate observa ca unele teste nu sunt rulate fiindca modificarea genereaza iteratii infinite.

2. Mutantul **Problem_mirror_AOIS_67** a fost creat prin schimbarea liniei 69 din functia getMirror din `y = y * 10 + x % 10` in `y = y * 10 + ++x % 10`. Rezultatul rularii testelor:



Este normal ca testele care ne asteptam sa intoarca true sa esueze, deoarece se schimba inversul elementului de pe ultima pozitie.

3. Mutantul **Problem_mirror_AORB_20** a fost creat prin schimbarea liniei 69 din functia getMirror din $y = y * 10 + x \% 10$ in $y = y * 10 - x \% 10$. Rezultatul rularii testelor:

Test Results	17 ms
Problem_mirror_AORB_20_Test	17 ms
mutant_Problem_mirror_AORB_20_equivalence_partitioning()	11 ms
N < 2: numberOfElements=0, input=[1, 2, 3, 4]	8 ms
N > 100: numberOfElements=123, input=[1, 2, 3, 4]	
N inside [2, 100] and null input data: numberOfElements=10, input=null	1 ms
N inside [2, 100] and input data of invalid length: numberOfElements=3, input=[1, 2, 3, 4, 5, 6]	
N inside [2, 100] and input data with negative values: numberOfElements=6, input=[1, 2, -3, 4, -5, 6]	
N inside [2, 100] and the result is false: numberOfElements=6, input=[10, 20, 3 ... 0, 50, 60]	1 ms
N inside [2, 100] and the result is true: numberOfElements=6, input=[10, 11, 1 ... 3, 14, 31]	1 ms
mutant_Problem_mirror_AORB_20_boundary_values()	4 ms
N is less than the lower bound: numberOfElements=0, input=[]	1 ms
N is less than the lower bound: numberOfElements=1, input=[1]	
N is equal to the lower bound, solution is on the position 0: numberOfElements=2, input=[123, 321]	
N is inside the expected bounds, solution is on the position 0: numberOfElements=50, input=[321, 2, 3 ..., 49, 123]	1 ms
N is inside the expected bounds, solution is on the before-the-last position : numberOfElements=50, input=[1, 2, 3, ..., 321, 123]	
N is equal to the higher bound, solution is on the position 0: numberOfElements=100, input=[321, 2, 3 ..., 99, 123]	1 ms
N is equal to the higher bound, solution is on the position 99: numberOfElements=100, input=[1, 2, 3, ..., 321, 123]	
N is greater than the higher bound.: numberOfElements=101, input=[123, 321]	
N is greater than the higher bound.: numberOfElements=150, input=[123, 321]	1 ms
mutant_Problem_mirror_AORB_20_cause_effect_graph()	2 ms
C1 false -> Ef1 true: numberOfElements=0, input=[1, 2, 3, 4]	
C1 true, C2 true -> Ef2 true: numberOfElements=10, input=null	
C1 true, C2 false, C3 false -> Ef3 true: numberOfElements=3, input=[1, 2, 3, 4, 5, 6]	1 ms
C1 true, C2 false, C3 true, C4 true -> Ef4 true: numberOfElements=6, input=[1, 2, -3, 4, -5, 6]	1 ms
C1 true, C2 false, C3 true, C4 false, C5 true -> Ef5 true: numberOfElements=6, input=[16, 2, 3, 4, 5, 61]	
C1 true, C2 false, C3 true, C4 false, C5 false -> Ef6 true: numberOfElements=6, input=[10, 11, 1 ... 3, 14, 38]	

Acelasi motiv ca mai devreme, vor pica testele unde se asteapta ca rezultatul sa fie true, deoarece se modifica inversul elementului de pe ultima pozitie

4. Mutantul **Problem_mirror_CDL_11** a fost creat prin schimbarea liniei 69 din functia getMirror din $y = y * 10 + x \% 10$ in $y = y * 10 + x$. Rezultatul rularii testelor:

Test Results	14 ms
Problem_mirror_CDL_11_Test	14 ms
mutant_Problem_mirror_CDL_11_cause_effect_graph()	10 ms
C1 false -> Ef1 true: numberOfElements=0, input=[1, 2, 3, 4]	8 ms
C1 true, C2 true -> Ef2 true: numberOfElements=10, input=null	1 ms
C1 true, C2 false, C3 false -> Ef3 true: numberOfElements=3, input=[1, 2, 3, 4, 5, 6]	
C1 true, C2 false, C3 true, C4 true -> Ef4 true: numberOfElements=6, input=[1, 2, -3, 4, -5, 6]	
C1 true, C2 false, C3 true, C4 false, C5 true -> Ef5 true: numberOfElements=6, input=[16, 2, 3, 4, 5, 61]	1 ms
C1 true, C2 false, C3 true, C4 false, C5 false -> Ef6 true: numberOfElements=6, input=[10, 11, 1 ... 3, 14, 38]	
mutant_Problem_mirror_CDL_11_equivalence_partitioning()	2 ms
N < 2: numberOfElements=0, input=[1, 2, 3, 4]	1 ms
N > 100: numberOfElements=123, input=[1, 2, 3, 4]	
N inside [2, 100] and null input data: numberOfElements=10, input=null	
N inside [2, 100] and input data of invalid length: numberOfElements=3, input=[1, 2, 3, 4, 5, 6]	
N inside [2, 100] and input data with negative values: numberOfElements=6, input=[1, 2, -3, 4, -5, 6]	1 ms
N inside [2, 100] and the result is false: numberOfElements=6, input=[10, 20, 3 ... 0, 50, 60]	
N inside [2, 100] and the result is true: numberOfElements=6, input=[10, 11, 1 ... 3, 14, 31]	
mutant_Problem_mirror_CDL_11_boundary_values()	2 ms
N is less than the lower bound: numberOfElements=0, input=[]	
N is less than the lower bound: numberOfElements=1, input=[1]	
N is equal to the lower bound, solution is on the position 0: numberOfElements=2, input=[123, 321]	1 ms
N is inside the expected bounds, solution is on the position 0: numberOfElements=50, input=[321, 2, 3 ..., 49, 123]	
N is inside the expected bounds, solution is on the before-the-last position : numberOfElements=50, input=[1, 2, 3, ..., 321, 123]	1 ms
N is equal to the higher bound, solution is on the position 0: numberOfElements=100, input=[321, 2, 3 ..., 99, 123]	
N is equal to the higher bound, solution is on the position 99: numberOfElements=100, input=[1, 2, 3, ..., 321, 123]	
N is greater than the higher bound.: numberOfElements=101, input=[123, 321]	
N is greater than the higher bound.: numberOfElements=150, input=[123, 321]	

Vor pica toate testele unde se asteapta ca rezultatul sa fie true, iar elemental de pe ultima pozitie are mai mult de o cifra.

5. Mutantul **Problem_solve_AOIU_5** a fost creat prin schimbarea liniei 35 din functia solve din “for (int i = 0; i < -numberOfElements; i++) {” in linia “for (int i = 0; i < numberOfElements; i++) {”.

Rezultatul rularii testelor:

Test Results	17 ms
Problem_solve_AOIU_5_Test	17 ms
mutant_Problem_solve_AOIU_5_equivalence_partitioning()	13 ms
✓ N < 2: numberOfElements=0, input=[1, 2, 3, 4]	9 ms
✓ N > 100: numberOfElements=123, input=[1, 2, 3, 4]	
✓ N inside [2, 100] and null input data: numberOfElements=10, input=null	1 ms
✓ N inside [2, 100] and input data of invalid length: numberOfElements=3, input=[1, 2, 3, 4, 5, 6]	1 ms
✗ N inside [2, 100] and input data with negative values: numberOfElements=6, input=[1, 2, -3, 4, -5, 6]	2 ms
✓ N inside [2, 100] and the result is false: numberOfElements=6, input=[10, 20, 3 ... 0, 50, 60]	
✓ N inside [2, 100] and the result is true: numberOfElements=6, input=[10, 11, 1 ... 3, 14, 31]	
▶ mutant_Problem_solve_AOIU_5_boundary_values()	3 ms
mutant_Problem_solve_AOIU_5_cause_effect_graph()	1 ms
✓ C1 false -> Ef1 true: numberOfElements=0, input=[1, 2, 3, 4]	
✓ C1 true, C2 true -> Ef2 true: numberOfElements=10, input=null	1 ms
✓ C1 true, C2 false, C3 false -> Ef3 true: numberOfElements=3, input=[1, 2, 3, 4, 5, 6]	
✗ C1 true, C2 false, C3 true, C4 true -> Ef4 true: numberOfElements=6, input=[1, 2, -3, 4, -5, 6]	
✓ C1 true, C2 false, C3 true, C4 false, C5 true -> Ef5 true: numberOfElements=6, input=[16, 2, 3, 4, 5, 61]	
✓ C1 true, C2 false, C3 true, C4 false, C5 false -> Ef6 true: numberOfElements=6, input=[10, 11, 1 ... 3, 14, 38]	

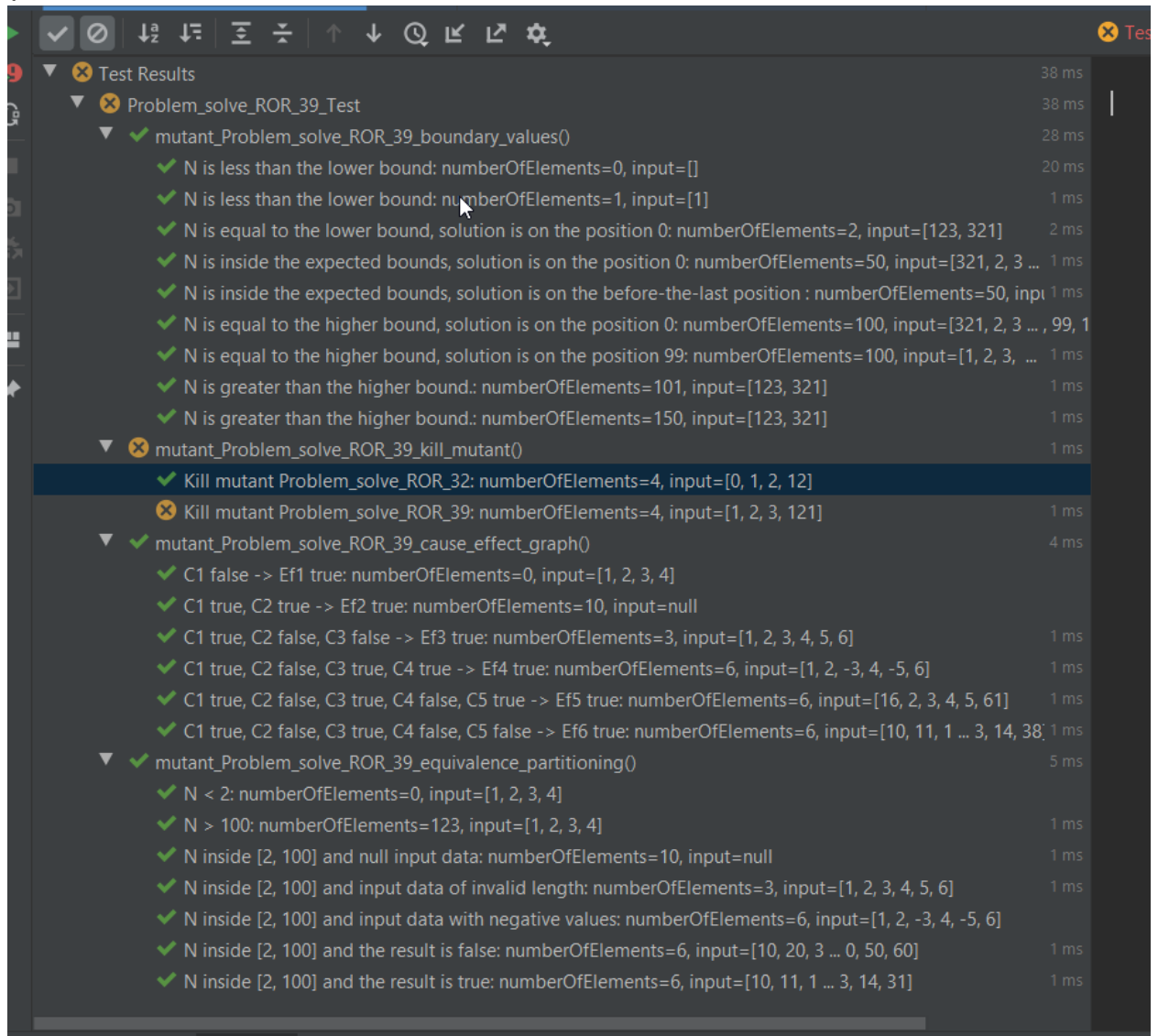
Vor pica toate testele care se asteapta ca rezultatul sa fie o exceptie cu mesajul “The element on position _ is not a natural number”.

6. Mutantul **Problem_solve_ROR_32** a fost creat prin schimbarea liniei 36 din functia solve din "if (input[i] < 0) {" in linia "if (input[i] <= 0) {}". Rezultatul rularii testelor:

Test Results	13 ms
Problem_solve_ROR_32_Test	13 ms
mutant_Problem_solve_ROR_32_cause_effect_graph()	9 ms
C1 false -> Ef1 true: numberOfElements=0, input=[1, 2, 3, 4]	8 ms
C1 true, C2 true -> Ef2 true: numberOfElements=10, input=null	
C1 true, C2 false, C3 false -> Ef3 true: numberOfElements=3, input=[1, 2, 3, 4, 5, 6]	
C1 true, C2 false, C3 true, C4 true -> Ef4 true: numberOfElements=6, input=[1, 2, -3, 4, -5, 6]	
C1 true, C2 false, C3 true, C4 false, C5 true -> Ef5 true: numberOfElements=6, input=[16, 2, 3, 4, 5, 61]	
C1 true, C2 false, C3 true, C4 false, C5 false -> Ef6 true: numberOfElements=6, input=[10, 11, 1 ... 3, 14, 1 ... 31]	1 ms
mutant_Problem_solve_ROR_32_equivalence_partitioning()	
N < 2: numberOfElements=0, input=[1, 2, 3, 4]	
N > 100: numberOfElements=123, input=[1, 2, 3, 4]	
N inside [2, 100] and null input data: numberOfElements=10, input=null	
N inside [2, 100] and input data of invalid length: numberOfElements=3, input=[1, 2, 3, 4, 5, 6]	
N inside [2, 100] and input data with negative values: numberOfElements=6, input=[1, 2, -3, 4, -5, 6]	
N inside [2, 100] and the result is false: numberOfElements=6, input=[10, 20, 3 ... 0, 50, 60]	
N inside [2, 100] and the result is true: numberOfElements=6, input=[10, 11, 1 ... 3, 14, 31]	
mutant_Problem_solve_ROR_32_kill_mutant()	
Kill mutant Problem_solve_ROR_32: numberOfElements=4, input=[0, 1, 2, 12]	
Kill mutant Problem_solve_ROR_39: numberOfElements=4, input=[1, 2, 3, 121]	
mutant_Problem_solve_ROR_32_boundary_values()	4 ms
N is less than the lower bound: numberOfElements=0, input=[]	
N is less than the lower bound: numberOfElements=1, input=[1]	
N is equal to the lower bound, solution is on the position 0: numberOfElements=2, input=[123, 321]	1 ms
N is inside the expected bounds, solution is on the position 0: numberOfElements=50, input=[321, 2, 3 ... 121]	1 ms
N is inside the expected bounds, solution is on the before-the-last position : numberOfElements=50, input=[123, 321, 2, 3 ... 121]	1 ms
N is equal to the higher bound, solution is on the position 0: numberOfElements=100, input=[321, 2, 3 ... 121]	1 ms
N is equal to the higher bound, solution is on the position 99: numberOfElements=100, input=[1, 2, 3, ... 32]	
N is greater than the higher bound.: numberOfElements=101, input=[123, 321]	
N is greater than the higher bound.: numberOfElements=150, input=[123, 321]	

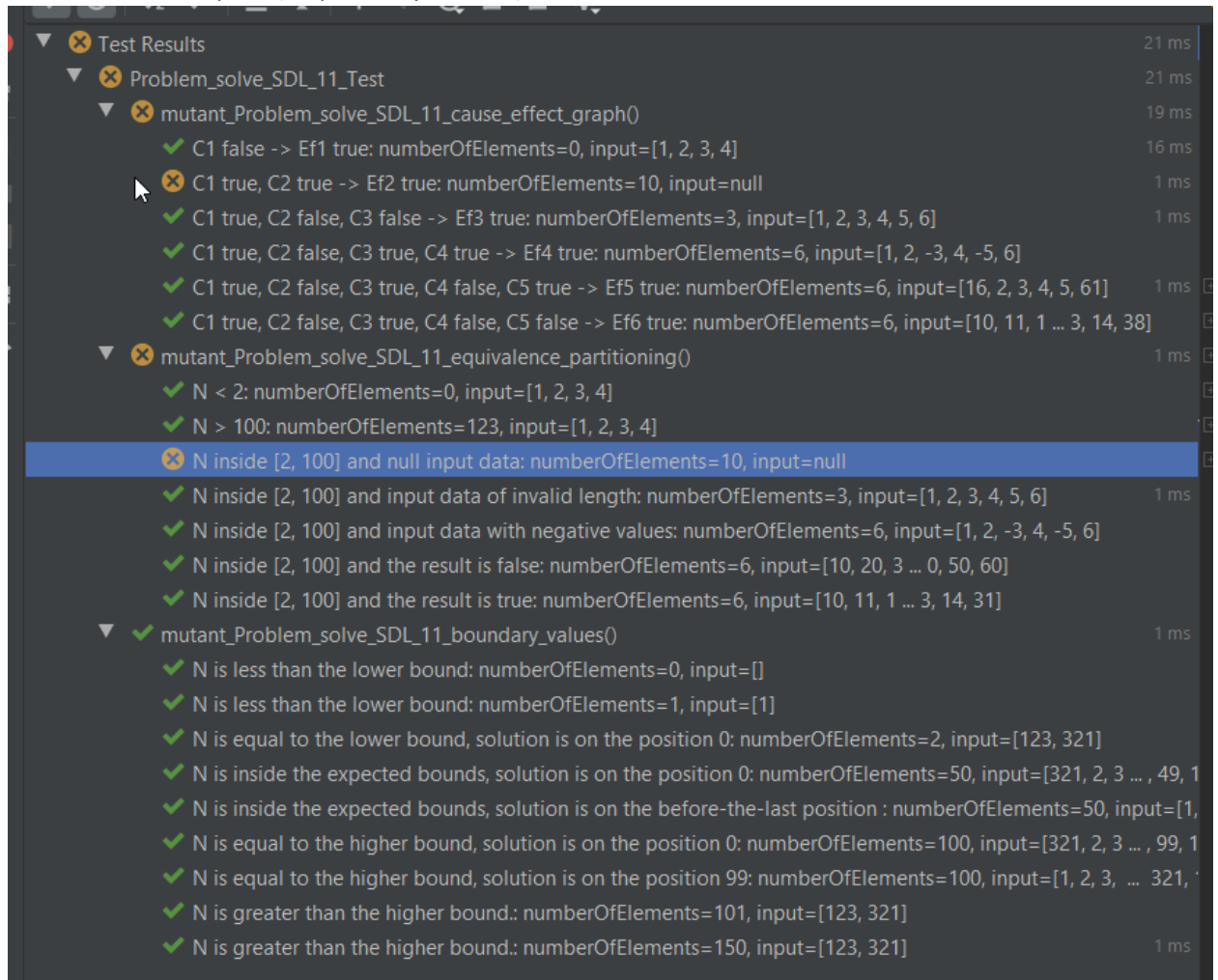
Concluzia este ca acest mutant este unul neechivalent cu problema initiala, care nu era omorat de niciun test deja existent. Acest lucru se datoreaza faptului ca nu aveam niciun test cu valori nule in sir. Am adaugat testul (4, [0, 1, 2, 12]) care ar trebui sa intoarca fals, dar arunca exceptie datorita modificarii mutantului.

7. Mutantul **Problem_solve_ROR_39** a fost creat prin schimbarea liniei 42 din functia solve() din "for (int i = 0; i < numberOfElements - 1; i++) {" in linia "for (int i = 0; i <= numberOfElements - 1; i++) {".
- Rezultatul rularii testelor:



Concluzia este ca iar am dat peste un mutant neechivalent care nu a fost omorat de testele deja existente. Modificarea afecteaza testele care ar trebui sa intoarca false, iar elementul de pe ultima pozitie este un palindrom. Am adaugat astfel testul (4, [1, 2, 3, 121]) care ar trebui sa intoarca fals, dar intoarce true.

8. Mutantul **Problem_solve_SDL_11** a fost creat prin eliminarea liniei 42 din functia solve: "throw new RuntimeException("Input array is null.");". Rezultatul rularii testelor:



Cum era de asteptat, pica toate testele care se asteapta ca rezulta tul sa fie exceptia "Input array is null".

ANEXA

