TESTARE SI VERIFICARE

PROIECT LABORATOR

Student: Neagu Andrei-Cosmin

Grupa: 505

I. Problema

Se dau doua numere intregi **min, max** unde (min < max , min > 0 , max > 0) si o lista de **size** elemente intregi unde (size > 0 si size < 100). Sa se obtina din cadrul listei numarul de elemente care fac parte din intervalul [min , max] si sunt perfecte.

Input:

min = 2

max = 34

size = 6

list = (3,6,9,21,28,35)

Output:

counter = 2 (rezultat)

Avem 2 numere perfecte (6 si 28) deoarece acestea doua au suma divizorilor egala cu valoarea numarului.

II. Solutie

```java
public static long getTotalPerfectNumbers(int min, int max, int size, List<Integer> list) {
    if (Objects.isNull(list) || list.size() != size) {
        return -1;
    }

    if (min < 0 || max < 0 || min >= max || size < 1 || size > 100) {
        return -1;
    }

    return list.stream()
            .filter(current -> current >= min && current <= max)
            .filter(current -> {
                int divSum = 1;
                for (int div = 2; div <= current / 2; div++) {
                    if (current % div == 0) {
                        divSum += div;
                    }
                }
                return current == divSum;
            })
            .count();
```

III. Testare functionala

a. Equivalence partitioning

Input:

- min, max (doua numere intregi)
- size (un numar intreg)
- list (lista de numere intreg)

Domeniul de intrari:

- MIN, MAX > 0 si MIN < MAX => 4 clase de echivalenta
    - $M\_1 = \{(min, max) \mid 0 \leq min < max\}$
    - $M\_2 = \{(min, max) \mid 0 \leq max < min\}$
    - $M\_3 = \{(min, max) \mid min < 0\}$
    - $M\_4 = \{(min, max) \mid max < 0\}$
- SIZE este in intervalul (1,100) => 3 clase de echivalenta
    - $S\_1 = 1 \ldots 100$
    - $S\_2 = \{size \mid size < 1\}$
    - $S\_3 = \{size \mid size > 100\}$

- LIST nu va avea niciun caz de echivalenta, deoarece lungimea acesteia este reprezentata de SIZE, astfel nu vor exista cazuri diferite pentru aceasta intrare.

Output:

- Totalul de elemente care reprezinta un numar perfect din lista de intrare.
- In cazul in care avem eroare / exceptie se va intoarce valoarea -1, astfel vom avea:
  - E_1 = { counter | counter ≥ 0, datele de intrare sunt corecte }
  - E_2 = { -1 | datele de intrare sunt incorecte }

Clasele de echivalenta

- C_11 = {(min, max, size, list) | size in S_1, |list| = size,(min, max) in M_1, E_1}
- C_12 = {(min, max, size, list) | size in S_1, |list| = size, (min, max) in M_2, E_2}
- C_13 = {(min, max, size, list) | size in S_1, |list| = size, (min, max) in M_3}
- C_14 = {(min, max, size, list) | size in S_1, |list| = size, (min, max) in M_4}
- C_2 = {(min, max, size, list) | size in S_2}
- C_3 = {(min, max, size, list) | size in S_3}

Date de test

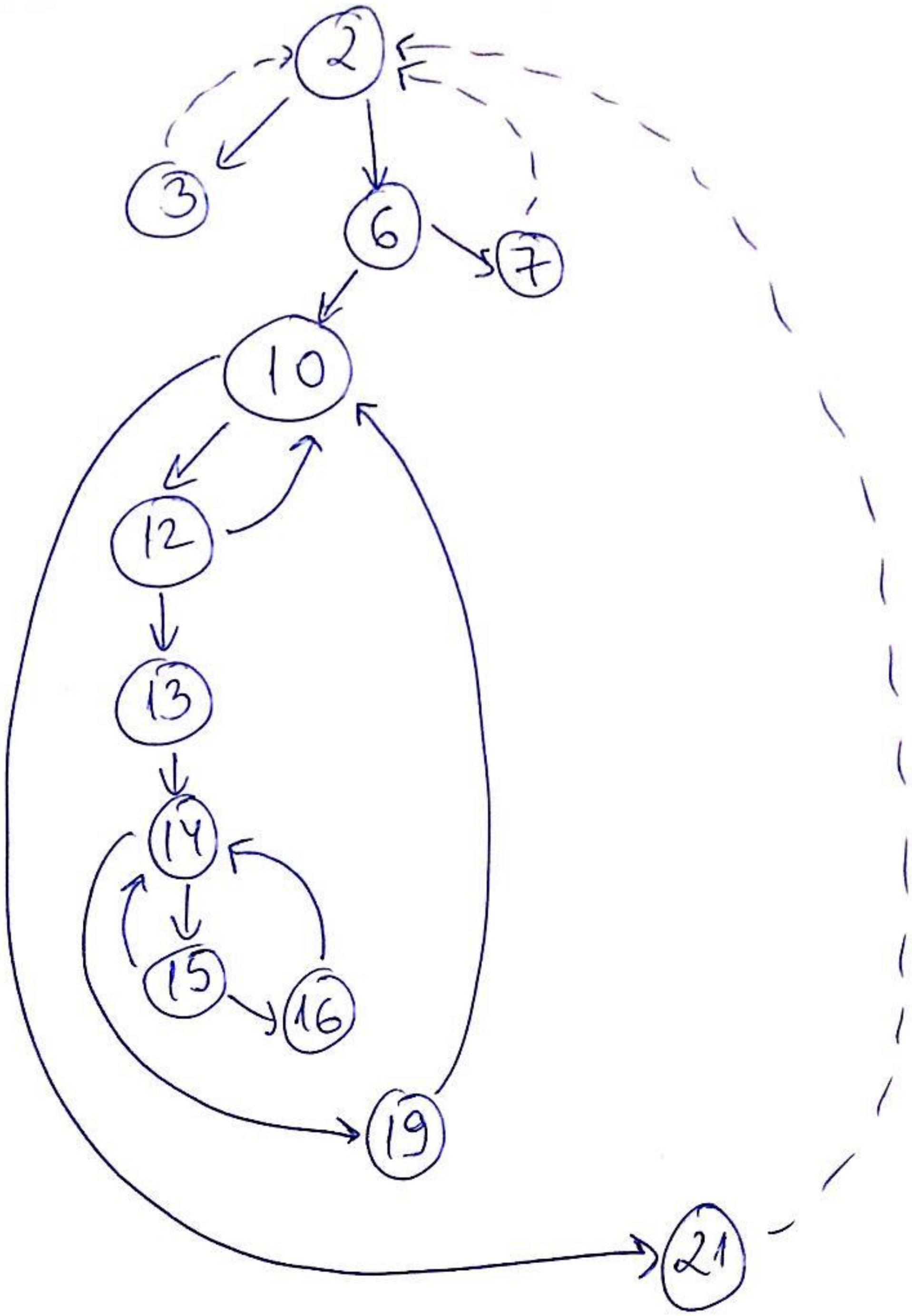
- C_11: (2, 34, 6, [3,6,9,21,28,35])
- C_12: (8, 26, 6, [3,6,9,21,28,35])
- C_13: (-1, _, _, _)
- C_14: (_, -1, _, _)
- C_2: (_, _, 0, _)
- C_3: (_, _, 101, _)

b. Boundary value analysis

- M_1 = {(min, max) | 0 <= min < max}
    - (0, 1) (valoare de margine) si o alta pereche din interior
- M_2 = {(min, max) | 0 <= max < min}
    - (1, 0) (valoare de margine) si o alta pereche din interior
- M_3 = {(min, max) | min < 0}
    - (-1, _) (valoare de margine) si orice alta valoare pentru min
- M_4 = {(min, max) | max < 0}
    - (_, -1) (valoare de margine) si orice alta valoare pentru max
- S_1 = 1 ...100
    - 1, 100 (valori de margine) și o valoare din interior
- S_2 = {size | size < 1}
    - 0 (valoare de margine) și o valoare din interior
- S_3 = {size | size > 100}
    - 101 (valoare de margine) și o valoare din interior

IV. Testare structurala

a. Graful asociat



b. Statement coverage (la nivel de instructiune)

- SC_1: (2, 34, 6, [3,6,9,21,28,35]) – prin acest set de date se asigura parcurgerea tuturor ramurilor din graf.
- SC_2: (2, 34, 6, []) – prin acest set de date de intrare se asigura prima conditie de iesire fortata din cauza inputului.
- SC_3: (4,2, 6, [1,2,3,4,5,6]) – prin acest set de date de intrare se asigura a doua conditie de iesire fortata din cauza inputului.

c. Decision coverage (la nivel de decizie)

Metoda contine 4 instructiuni de decizie, toate cazurile sunt acoperite de testele SC_1, SC_2, SC_3 cu exceptia cazului in care nu se regaseste niciun numar perfect. Pe care il vom trata in testul DC_1:

- DC_1: (2,34,6,[3,4,9,21,27,34]) – prin acest set de date de intrare se asigura negasirea unui numar perfect.

d. Condition coverage (la nivel de conditie)

<table>
<tr><th>Decizii</th><th>Conditii individuale</th></tr>
<tr><td>if (Objects.isNull(list) || list.size() != size)</td><td>Objects.isNull(list), list.size() != size</td></tr>
<tr><td>if (min < 0 || max < 0 || min >= max || size < 1 || size > 100)</td><td>min < 0, max < 0, min >= max, size < 1, size > 100</td></tr>
<tr><td>current >= min && current <= max</td><td>current >= min, current <= max</td></tr>
<tr><td>for (int div = 2; div <= current / 2; div++)</td><td>div <= current / 2</td></tr>
<tr><td>if (current % div == 0)</td><td>current % div == 0</td></tr>
</table>

Pentru acoperirea totala a cazurilor sunt necesare 2^2 teste pentru prima decizie, iar pentru cea de a doua decizie 2^5 teste.

Majoritatea conditiilor sunt verificate la punctul b) de mai sus, astfel nu a mai fost necesara scrierea altor teste intr-o clasa separata.

V. Complexitate

Vom aplica formula lui McGabe de determinare a complexitatii ciclomatice.

$$M = E - N + 2$$

Unde:

E = numarul de colturi ale grafului

N = numarul de noduri

M = complexitatea

Astfel:

$$M = 21 - 12 + 2 = 7$$

Obtinem 7 circuite independente:

C1: 2-3-2

C2: 2-6-7-2

C3: 2-6-10-21-2

C4: 10-12-10

C5: 10-12-13-14-19-10

C6: 14-15-14

C7: 14-15-16-14

Pentru testarea la nivel de circuit:

1. Se obține pentru o lista fara elemente (SC_2).
2. Se obtine cand nu se respecta valorile (min,max,size) sunt iesite din intervalul lor de baza (III.b)
3. Niciun element nu se afla in intervalul (min,max).
4. Pentru numerele prime care sunt numere perfecte
5. Pentru numerele prime care nu sunt numere perfecte
6. Numere prime
7. Suma divizorilor

Cazurile acestea sunt acoperite de sectiunile anterioare.

VI. Mutating

Executam cu ajutorul comenzii: mvn org.pitest:pitest-maven:mutationCoverage, iar ulterior rezultatele sunt regasite in target/pit-reports/..

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
1	93%	13/14	86%	25/29

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
test	1	93%	13/14	86%	25/29

```
package test;

import java.util.List;
import java.util.Objects;

public class Utility {

    private Utility() {}

    public static long getTotalPerfectNumbers(int min, int max, int size, List<Integer> list) {
        if (Objects.isNull(list) || list.size() != size) {
            return -1;
        }

        if (min < 0 || max < 0 || min >= max || size < 1 || size > 100) {
            return -1;
        }

        return list.stream()
                .filter(current -> current >= min && current <= max)
                .filter(current -> {
                    int divSum = 1;
                    for (int div = 2; div <= current / 2; div++) {
                        if (current % div == 0) {
                            divSum += div;
                        }
                    }
                    return current == divSum;
                })
                .count();
    }
}
```

Se observa urmatorii mutanti. Linia 15 si linia 20.

Pentru a se rezolva mutantii de pe linia 15 trebuie scris un numar destul de mare de teste.

Cel de pe linia 20 se poate rezolva

Mutations

11	1. negated conditional → KILLED 2. negated conditional → KILLED
12	1. replaced return of long value with value + 1 for test/Utility::getTotalPerfectNumbers → KILLED
15	1. changed conditional boundary → KILLED 2. changed conditional boundary → SURVIVED 3. changed conditional boundary → SURVIVED 4. changed conditional boundary → KILLED 5. changed conditional boundary → KILLED 6. negated conditional → KILLED 7. negated conditional → KILLED 8. negated conditional → KILLED 9. negated conditional → KILLED 10. negated conditional → KILLED
16	1. replaced return of long value with value + 1 for test/Utility::getTotalPerfectNumbers → KILLED
19	1. replaced return of long value with value + 1 for test/Utility::getTotalPerfectNumbers → KILLED
20	1. changed conditional boundary → SURVIVED 2. changed conditional boundary → SURVIVED 3. negated conditional → KILLED 4. negated conditional → KILLED 5. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED

Dupa efectuarea schimbarilor am ajuns sa ramanem cu mutanti doar pe linia 15.

Pit Test Coverage Report

Package Summary

test

Number of Classes	Line Coverage		Mutation Coverage	
1	93%	13/14	93%	27/29

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
Utility.java	93%	13/14	93%	27/29

```
6       public class Utility {
7
8           private Utility() {}
9
10          public static long getTotalPerfectNumbers(int min, int max, int size, List<Integer> list) {
11 2            if (Objects.isNull(list) || list.size() != size) {
12 1                return -1;
13              }
14
15 10           if (min < 0 || max < 0 || min >= max || size < 1 || size > 100) {
16 1                return -1;
17              }
18
19 1            return list.stream()
20 5                    .filter(current -> current >= min && current <= max)
21                      .filter(current -> {
22                          int divSum = 1;
23 4                        for (int div = 2; div <= current / 2; div++) {
24 2                            if (current % div == 0) {
25 1                                divSum += div;
26                              }
27                          }
28 2                        return current == divSum;
29                      })
30                      .count();
31          }
32      }
```

Cod sursa: [neaguandrei/mutation-testing: A detailed testing of a program that looks for the total of perfect numbers from a list of integer values. Includes some mutation testing with PIT Mutation Testing. (github.com)]