Documentație - Testare in Python

Barbarasa Maria-Cristina-311

Oproiu Matei-311

Problema noastră : Se testează un program care verifică cel mai mare divizor comun între două numere a și b modulo c. Mai precis, pentru trei variabile care vor îndeplini condiția să fie numere naturale strict pozitive se va calcula folosind algoritmul lui Euclid cu scăderi repetate între a și b ,pentru un număr maxim de 1000 de iterații. Programul va produce un output care va indica rezultatul în urma calculului algoritmului modulo c.

1.Testarea funcţională clasei ModuloGCD

(a)Partiţionare de echivalenta

1.Domeniul de intrări: Pentru clasa compusa avem 3 intrări:

- a- valoare naturală strict pozitivă deci se disting două clase de echivalența:
  A_1={ a | a număr natural strict pozitiv}
  A_2={ a | a nu este număr natural strict pozitiv}
- b- valoare naturală strict pozitivă deci se disting două clase de echivalența:

  B_1={ b | b număr natural strict pozitiv}

  B_2={ b | b nu este număr natural strict pozitiv}
- c-valoare naturală strict pozitivă deci se disting două clase de echivalența:

  C_1={ c | c număr natural strict pozitiv}

  C_2={ c| c nu este număr natural strict pozitiv}

2.Domeniul de ieşire: Constă într-un număr natural result ∈ [0,c) și este rezultatul procesării numerelor a, b, c. Acesta este folosit pentru a împarți domeniul de intrare în mai multe clase, in funcție de valorile a, b,c:

X_1={ result | GCD(a, b)>c }

X_2={ result | GCD(a, b)==c }

X_3={ result | GCD(a, b) < c }

Clasele de echivalențe globale pe care le avem:

G_111={ (a,b,c) | a ∈ A_1 & b ∈ B_1 & c ∈ C_1 }

G_211={ (a,b,c) | a ∈ A_2 & b ∈ B_1 & c ∈ C_1 }

G_121={ (a,b,c) | a ∈ A_1 & b ∈ B_2 & c ∈ C_1 }

G_112={ (a,b,c) | a ∈ A_1 & b ∈ B_1 & c ∈ C_2 }

G_122={(a,b,c) | a ∈ A_1 & b ∈ B_2 & c ∈ C_2 }

G_212={(a,b,c) | a ∈ A_2 & b ∈ B_1 & c ∈ C_2 }

G_221={(a,b,c) | a ∈ A_2 & b ∈ B_2 & c ∈ C_1 }

G_222={(a,b,c) | a ∈ A_2 & b ∈ B_2 & c ∈ C_2 }

Setul de date se alcătuiește prin alegerea unei valori de intrare pentru fiecare clasa. Exemplu:

g_111: (10,5,2)

g_211: (-5,5,2)

g_121: (50,”5”,4)

g_112: (43,5,-2)

g_122: (5,-5,-2)

g_212: (100.2,31,none)

g_221: (-5,32.2,6)

g_222: (-5,15.2,”12”)

a	b	c	Rezultat afișat (expected)
10	5	2	1
15	10	4	1

8	12	5	4
-5	5	2	Ridică ValueError: “Valorile trebuie sa fie numere naturale pozitive”
50	“5”	4	Ridică ValueError: “Valorile trebuie sa fie numere naturale pozitive”
10	5	0	Ridică ValueError: “Valorile trebuie sa fie numere naturale pozitive”
5	-5	-2	Ridică ValueError: “Valorile trebuie sa fie numere naturale pozitive”
100.2	31	None	Ridică ValueError: “Valorile trebuie sa fie numere naturale pozitive”
-5	32.2	6	Ridică ValueError: “Valorile trebuie sa fie numere naturale pozitive”
-5	15.2	“12”	Ridică ValueError: “Valorile trebuie sa fie numere naturale pozitive”

```
"""Partitionare de echivalenta"""

def test_equivalence_classes(self):
    self.assertEqual(ModuloGCD(10,5,3).cmmdc(),2)
    self.assertEqual(ModuloGCD(15, 10,4).cmmdc(),1)
    self.assertEqual(ModuloGCD(8,12,5).cmmdc(),4)

    # G211 - Invalid a
    with self.assertRaises(ValueError):
        ModuloGCD(-5,5,2).cmmdc()
    # G121 - Invalid b
    with self.assertRaises(ValueError):
        ModuloGCD(50,'5',4).cmmdc()
    # G112 - Invalid c
    with self.assertRaises(ValueError):
        ModuloGCD(10,5,0).cmmdc()
    # G122 - Invalid b and c
    with self.assertRaises(ValueError):
        ModuloGCD(5,-5,-2).cmmdc()
    # G212 - Invalid a and c
    with self.assertRaises(ValueError):
        ModuloGCD(100.2,31,None).cmmdc()
    # G221 - Invalid a and b
    with self.assertRaises(ValueError):
        ModuloGCD(-5,32.2,6).cmmdc()
    # G222 - Invalid a, b, and c
    with self.assertRaises(ValueError):
        ModuloGCD(-5,15.2,'12').cmmdc()
```

"""1) Partitionare de echivalenta"""

def test_equivalence_classes(self):
self.assertEqual(ModuloGCD(10,5,3).cmmdc(),2)
self.assertEqual(ModuloGCD(15, 10,4).cmmdc(),1)
self.assertEqual(ModuloGCD(8,12,5).cmmdc(),4)

```
 # G211 - Invalid a
  with self.assertRaises(ValueError):
     ModuloGCD(-5,5,2).cmmdc()
  # G121 - Invalid b
  with self.assertRaises(ValueError):
     ModuloGCD(50,'5',4).cmmdc()
  # G112 - Invalid c
  with self.assertRaises(ValueError):
     ModuloGCD(10,5,0).cmmdc()
  # G122 - Invalid b and c
  with self.assertRaises(ValueError):
```

```
    ModuloGCD(5,-5,-2).cmmdc()
  # G212 - Invalid a and c
  with self.assertRaises(ValueError):
    ModuloGCD(100.2,31,None).cmmdc()
  # G221 - Invalid a and b
  with self.assertRaises(ValueError):
    ModuloGCD(-5,32.2,6).cmmdc()
  # G222 - Invalid a, b, and c
  with self.assertRaises(ValueError):
    ModuloGCD(-5,15.2,'12').cmmdc()
```

(b) Analiza valorilor de frontineră (boundary value analysis)

Analiza valorilor de frontieră este adesea utilizată împreună cu partiționarea în clase de echivalență. Această tehnică pune accentul pe testarea valorilor aflate la limita fiecărei clase, deoarece acestea reprezintă frecvent o sursă majoră de erori. În cazul exemplului nostru, odată ce clasele de echivalență au fost stabilite, identificarea valorilor de frontiera devine un proces intuitiv :

- valorile de frontieră sunt 0,1,2 pentru a, b, c

Se vor testa următoarele valori:

- N_1 : 0
- N_2 : 1
- N_3 : 2

Vom avea 9 date de test :

a	b	c	Rezultat afișat (expected)
0	5	2	Ridică ValueError: “Valorile trebuie sa fie numere naturale pozitive”

7	0	3	Ridică ValueError: “Valorile trebuie sa fie numere naturale pozitive”
7	21	0	Ridică ValueError: “Valorile trebuie sa fie numere naturale pozitive”
1	10	3	1
2	5	2	1
21	1	5	1
14	2	7	2
33	11	1	0
15	3	2	1

```
"""Analiza valorilor de frontiera"""

def test_boundary_values(self):
    # a=0
    with self.assertRaises(ValueError):
        ModuloGCD(0,5,2).cmmdc()
     # b=0
    with self.assertRaises(ValueError):
        ModuloGCD(7,0,3).cmmdc()
    # c=0
    with self.assertRaises(ValueError):
        ModuloGCD(7,21,0).cmmdc()

    # a=1
    self.assertEqual(ModuloGCD(1,10,3).cmmdc(),1)
    # a=2
    self.assertEqual(ModuloGCD(2,5,2).cmmdc(),1)
    # b=1
    self.assertEqual(ModuloGCD(21,1,5).cmmdc(),1)
    # b=2
    self.assertEqual(ModuloGCD(14,2,7).cmmdc(),2)
    # c=1
    self.assertEqual(ModuloGCD(33,11,1).cmmdc(),0)
    # c=2
    self.assertEqual(ModuloGCD(15,3,2).cmmdc(),1)
```

"""2) Analiza valorilor de frontiera"""

def test_boundary_values(self):

a=0

with self.assertRaises(ValueError):

ModuloGCD(0,5,2).cmmdc()

b=0

with self.assertRaises(ValueError):

ModuloGCD(7,0,3).cmmdc()

```
 # c=0

with self.assertRaises(ValueError):

        ModuloGCD(7,21,0).cmmdc()

  # a=1
   self.assertEqual(ModuloGCD(1,10,3).cmmdc(),1)
   # a=2
   self.assertEqual(ModuloGCD(2,5,2).cmmdc(),1)
   # b=1
   self.assertEqual(ModuloGCD(21,1,5).cmmdc(),1)
   # b=2
   self.assertEqual(ModuloGCD(14,2,7).cmmdc(),2)
   # c=1
   self.assertEqual(ModuloGCD(33,11,1).cmmdc(),0)
   # c=2
   self.assertEqual(ModuloGCD(15,3,2).cmmdc(),1)
```

2.Testarea structurală

1.Graful de flux de control

Am ales datele de test astfel încât să parcurgă toate elementele noastre introduse în cod.

```
12      def cmmdc(self):
13          a=self.validate_integer(self.a)
14          b=self.validate_integer(self.b)
15          c=self.validate_integer(self.c)
16
17          max_iterations=1000
18          iterations=0
19
20          if c<=0:
21              raise ValueError('Modulo trebuie sa fie pozitiv')
22
23          if a==b:
24              return a%c
25
26          while a!=b:
27              if iterations>max_iterations:
28                  raise RuntimeError('Prea multe iteratii (posibil loop infinit)')
29              iterations+=1
30
31              if a>b:
32                  a=a-b
33              else:
34                  b=b-a
35
36          return a%c
```

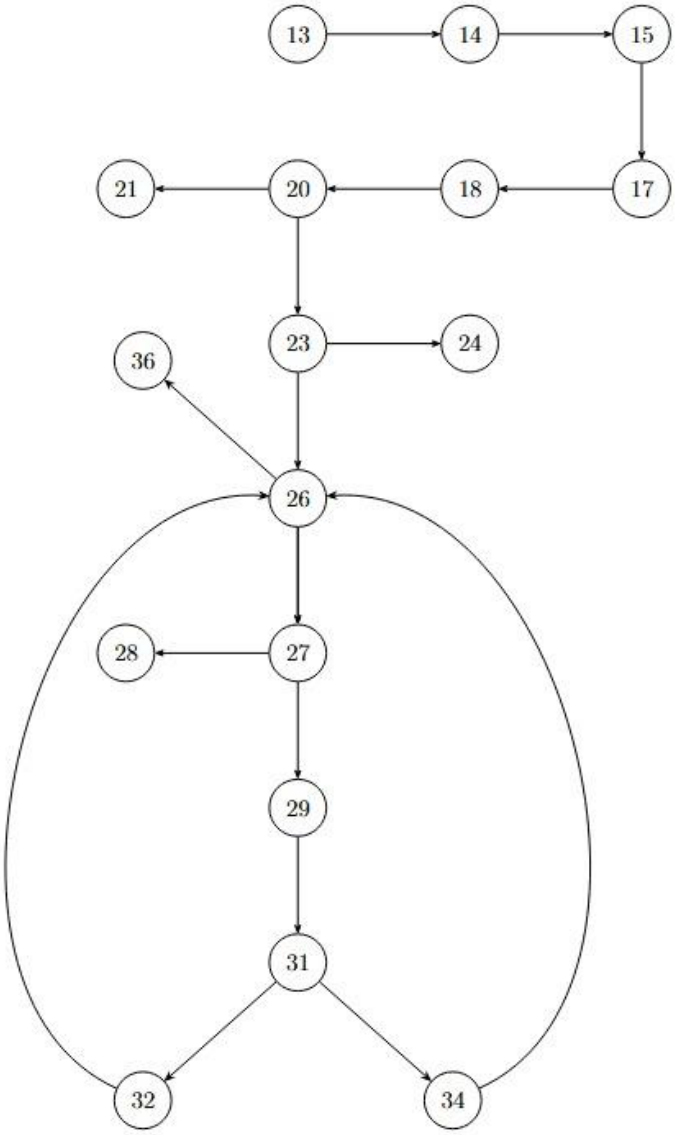
```
  def cmmdc(self):
```

```
    a=self.validate_integer(self.a)
    b=self.validate_integer(self.b)
    c=self.validate_integer(self.c)
    max_iterations=1000
    iterations=0
    if c<=0:
      raise ValueError('Modulo trebuie sa fie pozitiv')
    if a==b
return a%c
    while a!=b:
      if iterations>max_iterations:
        raise RuntimeError('Prea multe iteratii (posibil loop infinit)')
      iterations+=1
      if a>b:
        a=a-b
      else:
        b=b-a
    return a%c
```

Nodul 13 verifică a, nodul 14 verifică b si nodul 15 verifică c. Nodul 17 ne inițializează numărul maxim de iterații, nodul 18 ne inițializează numărul de iterații, nodul 20 verifică dacă c este valid în caz contrar programul se încheie, nodul 23 verifică daca a=b atunci iese din program. Nodul 26 intră în loop-ul while cât timp a!=b, nodul 27 verifică dacă am depășit nr maxim de iterații, nodul 29 creste indexarea iterațiilor, nodul 31 verifică dacă este mai mare decât în cazul acesta nodul 32 schimba valoarea lui a conform a=a-b

altfel nodul 34 schimbă valoarea lui b conform b=b-a. Iar nodul 36 este nodul nostru de ieșire.

De asemenea, pentru nodurile 13,14,15 este necesara trecerea de către condiția if not isinstance(value,int) or value < 0 pentru a se putea forma.



(a) Acoperirea la nivel de instrucțiune :

Ca să avem ca rezultat o acoperire la nivel de instrucțiune testul este luat astfel încât să parcurgă fiecare nod al grafului.

a	b	c	Rezultat afișat	Instrucțiuni parcurse
43	17	2	1	13,14,15,17,18,20,23,26,27,29,31,32,26,27,29,31,32,26,27,29,31,34,26,27,29,31,32,26,27,29,31,34,26,27,29,31,34,26,27,29,31,34,26,27,29,31,34,26,27,29,31,34,26,27,29,31,34,26,36
7	19	3	1	13,14,15,17,18,20,23,26,27,29,31,34,26,27,29,31,34,26,27,29,31,32,26,27,29,31,34,26,27,29,31,34,26,27,29,31,32,26,36
14	7	4	3	13,14,15,16,17,18,20,23,26,27,29,31,32,26,36

```
"""Acoperire la nivel de instructiune"""

def test_instruction_coverage(self):
    # Statement coverage 1
    self.assertEqual(ModuloGCD(43,17,2).cmmdc(),1)
    # Statement covarage 2
    self.assertEqual(ModuloGCD(7,19,3).cmmdc(),1)
    # Statement coverage 3
    self.assertEqual(ModuloGCD(14,7,4).cmmdc(),3)
```

"""3) Acoperire la nivel de instructiune"""

def test_instruction_coverage(self): # Statement coverage 1
self.assertEqual(ModuloGCD(43,17,2).cmmdc(),1) # Statement covarage 2
self.assertEqual(ModuloGCD(7,19,3).cmmdc(),1) # Statement coverage 3
self.assertEqual(ModuloGCD(14,7,4).cmmdc(),3)

(b) Acoperire la nivel de decizie (decision coverage)

	Decizii
(1)	if not isinstance(value,int) or value <0:
(2)	while a!=b:
(3)	if a>b:

a	b	c	Rezultat afișat	Decizii acoperite
10	10	12	10	(1)-A,(2)-F
14	10	5	2	(1)-A, (2)-A,(3)-A
6	9	7	3	(1)-A, (2)-A,(3)-F

```
"""Acoperire la nivel de decizie"""

def test_decision_coverage(self):
    # Loop skipped
    self.assertEqual(ModuloGCD(10,10,12).cmmdc(),10)
    # Loop true branch
    self.assertEqual(ModuloGCD(14,10,5).cmmdc(),2)
    # Else branch
    self.assertEqual(ModuloGCD(6,9,7).cmmdc(),3)

    # Invalid input
    with self.assertRaises(ValueError):
        ModuloGCD('10',5,2).cmmdc()
```

"""4) Acoperire la nivel de decizie"""

def test_decision_coverage(self): # Loop skipped
self.assertEqual(ModuloGCD(10,10,12).cmmdc(),10) # Loop true branch
self.assertEqual(ModuloGCD(14,10,5).cmmdc(),2) # Else branch
self.assertEqual(ModuloGCD(6,9,7).cmmdc(),3)

```
# Invalid input
 with self.assertRaises(ValueError):
    ModuloGCD('10',5,2).cmmdc()
```

(c) Acoperire la nivel de condiţie (condition coverage)

Decizii	Condiții individuale
if not isinstance(value,int) or value <=0 :	isintsance(value,int), value <=0
while a!= b:	a!=b
if a>b:	a>b

a	b	c	Rezultat afișat	Condiții/Decizii individuale acoperite
“9”	4	3	Ridică ValueError: “Valorile trebuie sa fie numere naturale pozitive”	isinstance(value,int)-F
10	-2	3	Ridică ValueError: “Valorile trebuie sa fie numere naturale pozitive”	value <=0-F
4	6	6	2	isinstance(value,int)-A, value <=0-A, a!=b-A, a>b-A
3	3	6	3	isinstance(value,int)-A, value <=0-A, a!=b-F
4	1	5	1	isinstance(value,int)-A, value <=0-A, a!=b-A, a>b-A
1	3	10	1	isinstance(value,int)-A, value <=0-A, a!=b-A, a>b-F

```
"""Acoperire la nivel de conditie"""

def test_condition_coverage(self):
    # Condition - isinstance fail
    with self.assertRaises(ValueError):
        ModuloGCD('9',4,3).cmmdc()
    # Condition value check fail
    with self.assertRaises(ValueError):
        ModuloGCD(10,-2,3).cmmdc()

    # While condition true
    self.assertEqual(ModuloGCD(4,6,6).cmmdc(),2)
    # While condition false
    self.assertEqual(ModuloGCD(3,3,6).cmmdc(),3)
    #If condition true
    self.assertEqual(ModuloGCD(4,1,5).cmmdc(),1)
    #If condition false
    self.assertEqual(ModuloGCD(1,3,10).cmmdc(),1)
```

"""Acoperire la nivel de conditie"""

def test_condition_coverage(self): # Condition - isinstance fail

with self.assertRaises(ValueError):

ModuloGCD('9',4,3).cmmdc()

Condition value check fail

with self.assertRaises(ValueError):

ModuloGCD(10,-2,3).cmmdc()

# While condition true
self.assertEqual(ModuloGCD(4,6,6).cmmdc(),2)
# While condition false
self.assertEqual(ModuloGCD(3,3,6).cmmdc(),3)
#If condition true
self.assertEqual(ModuloGCD(4,1,5).cmmdc(),1)
#If condition false

self.assertEqual(ModuloGCD(1,3,10).cmmdc(),1)

(d) Testarea circuitelor independente:

Numărul minim de circuite independente pentru a obține o acoperire completă a ramurilor este data de formula lui McCabe. V(G)=e-n+2p, unde:

e = numărul de muchii ale graficului = 18

n = numărul de noduri ale graficului = 17

p = numărul de componente conectate = 1

Deci, în cazul nostru V(G)= 18-17+2*1=3

- Circuitele independente:

a) 13,14,15,17,18,20,23,26 = fals -> 36

b) 13,14,15,17,18,20,23,26 = adevărat , repetăm 27,29,31,32 -> 36

c) 13,14,15,17,18,20,23,26 = adevărat repetam 27,29,31,34 ->36

```
"""Testarea circuitelor independente"""

def test_independent_paths(self):
    # Exist of isinstance fail
    with self.assertRaises(ValueError):
        ModuloGCD('10',3,6).cmmdc()
    # Exist of value check fail
    with self.assertRaises(ValueError):
        ModuloGCD(10,3.3,6).cmmdc()
    # Exist on both checks failing
    with self.assertRaises(ValueError):
        ModuloGCD('10',3.3,6).cmmdc()

    # Exist when a==b
    self.assertEqual(ModuloGCD(10,10,12).cmmdc(),10)
    # Loop executed
    self.assertEqual(ModuloGCD(55,10,12).cmmdc(),5)
```

"""6) Testarea circuitelor independente"""

def test_independent_paths(self):

Exist of isinstance fail

with self.assertRaises(ValueError):

ModuloGCD('10',3,6).cmmdc()

Exist of value check fail

with self.assertRaises(ValueError):

ModuloGCD(10,3.3,6).cmmdc()

```
# Exist on both checks failing
with self.assertRaises(ValueError):
        ModuloGCD('10',3.3,6).cmmdc()
# Exist when a==b
self.assertEqual(ModuloGCD(10,10,12).cmmdc(),10)
# Loop executed
self.assertEqual(ModuloGCD(55,10,12).cmmdc(),5)
```

3.Mutanti

Testarea mutanților este considerată o testare a calității testelor prin adăugarea unor modificări și a unor mutanți în codul de bază.

Mutation= modificare foarte mică

Evaluarea testează dacă se detectează mutanții.

Ideea este de a evalua cât de bune sunt testele făcute.

Pentru a evalua toti mutantii, i-am generat automat folosind mutpy.

Programul nostru :

```python
class ModuloGCD:
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

    def validate_integer(self, value):
        if not isinstance(value, int) or value <= 0:
            raise ValueError('Valorile trebuie sa fie numere naturale pozitive')
        return value

    def cmmdc(self):
        a = self.validate_integer(self.a)
        b = self.validate_integer(self.b)
        c = self.validate_integer(self.c)

        max_iterations = 1000
        iterations = 0

        if c <= 0:
            raise ValueError('Modulo trebuie sa fie pozitiv')

        if a == b:
            return a % c

        while a != b:
            if iterations > max_iterations:
                raise RuntimeError('Prea multe iteratii (posibil loop infinit)')
            iterations += 1

            if a > b:
                a = a - b
            else:
                b = b - a

        return a % c
```

Mutanti obtinuti:

```
[*] Start mutation process:
   - targets: ModuloGCD
   - tests: test_nou
[*] 7 tests passed:
   - test_nou [0.00100 s]
[*] Start mutants generation and execution:
   - [#   1] AOR ModuloGCD:
----------------------------------------------------------------------------------
  20:         if c <= 0:
  21:             raise ValueError('Modulo trebuie sa fie pozitiv')
  22:
  23:         if a == b:
- 24:             return a % c
+ 24:             return a * c
  25:
  26:         while a != b:
  27:             if iterations > max_iterations:
  28:                 raise RuntimeError('Prea multe iteratii (posibil loop infinit)')
----------------------------------------------------------------------------------
[0.04130 s] killed by test_condition_coverage (test_nou.ModuloGCDTest.test_condition_coverage)
```

```
    - [#   2] AOR ModuloGCD:
 ----------------------------------------------------------------------------------
   28:                 raise RuntimeError('Prea multe iteratii (posibil loop infinit)')
   29:             iterations += 1
   30:
   31:             if a > b:
 - 32:                 a = a - b
 + 32:                 a = a + b
   33:             else:
   34:                 b = b - a
   35:
   36:         return a % c
 ----------------------------------------------------------------------------------
 [0.00404 s] killed by test_boundary_values (test_nou.ModuloGCDTest.test_boundary_values)
```

```
    - [#   3] AOR ModuloGCD:
--------------------------------------------------------------------------------
  30:
  31:             if a > b:
  32:                 a = a - b
  33:             else:
- 34:                 b = b - a
+ 34:                 b = b + a
  35:
  36:         return a % c
--------------------------------------------------------------------------------
[0.00100 s] killed by test_boundary_values (test_nou.ModuloGCDTest.test_boundary_values)
```

```
    - [#   4] AOR ModuloGCD:
--------------------------------------------------------------------------------
  32:                 a = a - b
  33:             else:
  34:                 b = b - a
  35:
- 36:         return a % c
+ 36:         return a * c
--------------------------------------------------------------------------------
[0.00099 s] killed by test_boundary_values (test_nou.ModuloGCDTest.test_boundary_values)
```

AOR - Arithmetic Operator Replacement - schimbă operatorii aritmetici

```
    - [#   5] ASR ModuloGCD:
--------------------------------------------------------------------------------
  25:
  26:         while a != b:
  27:             if iterations > max_iterations:
  28:                 raise RuntimeError('Prea multe iteratii (posibil loop infinit)')
- 29:             iterations += 1
+ 29:             iterations -= 1
  30:
  31:             if a > b:
  32:                 a = a - b
  33:             else:
--------------------------------------------------------------------------------
[0.00102 s] survived
```

ROR - Relational Operator Replacement - schimbă operatorii de relație

```
    - [#   6] COD ModuloGCD:
--------------------------------------------------------------------------------
   4:         self.b = b
   5:         self.c = c
   6:
   7:     def validate_integer(self, value):
-  8:         if (not (isinstance(value, int)) or value <= 0):
+  8:         if (isinstance(value, int) or value <= 0):
   9:             raise ValueError('Valorile trebuie sa fie numere naturale pozitive')
  10:         return value
  11:
  12:     def cmmdc(self):
--------------------------------------------------------------------------------
[0.00500 s] killed by test_boundary_values (test_nou.ModuloGCDTest.test_boundary_values)
```

COD – Conditional Operator Deletion - ștergerea condiționată

```
   - [#   7] COI ModuloGCD:
--------------------------------------------------------------------------------
   4:         self.b = b
   5:         self.c = c
   6:
   7:     def validate_integer(self, value):
-  8:         if (not (isinstance(value, int)) or value <= 0):
+  8:         if not ((not (isinstance(value, int)) or value <= 0)):
   9:             raise ValueError('Valorile trebuie sa fie numere naturale pozitive')
  10:         return value
  11:
  12:     def cmmdc(self):
--------------------------------------------------------------------------------
[0.00429 s] killed by test_boundary_values (test_nou.ModuloGCDTest.test_boundary_values)
```

```
   - [#   8] COI ModuloGCD:
--------------------------------------------------------------------------------
  16:
  17:         max_iterations = 1000
  18:         iterations = 0
  19:
- 20:         if c <= 0:
+ 20:         if not (c <= 0):
  21:             raise ValueError('Modulo trebuie sa fie pozitiv')
  22:
  23:         if a == b:
  24:             return a % c
--------------------------------------------------------------------------------
[0.00101 s] killed by test_boundary_values (test_nou.ModuloGCDTest.test_boundary_values)
```

```
   - [#   9] COI ModuloGCD:
--------------------------------------------------------------------------------
  19:
  20:         if c <= 0:
  21:             raise ValueError('Modulo trebuie sa fie pozitiv')
  22:
- 23:         if a == b:
+ 23:         if not (a == b):
  24:             return a % c
  25:
  26:         while a != b:
  27:             if iterations > max_iterations:
--------------------------------------------------------------------------------
[0.00201 s] killed by test_boundary_values (test_nou.ModuloGCDTest.test_boundary_values)
```

```
   - [#  10] COI ModuloGCD:
--------------------------------------------------------------------------------
  22:
  23:         if a == b:
  24:             return a % c
  25:
- 26:         while a != b:
+ 26:         while not (a != b):
  27:             if iterations > max_iterations:
  28:                 raise RuntimeError('Prea multe iteratii (posibil loop infinit)')
  29:             iterations += 1
  30:
--------------------------------------------------------------------------------
[0.00100 s] killed by test_boundary_values (test_nou.ModuloGCDTest.test_boundary_values)
```

```
   - [#  11] COI ModuloGCD:
--------------------------------------------------------------------------------
  23:         if a == b:
  24:             return a % c
  25:
  26:         while a != b:
- 27:             if iterations > max_iterations:
+ 27:             if not (iterations > max_iterations):
  28:                 raise RuntimeError('Prea multe iteratii (posibil loop infinit)')
  29:             iterations += 1
  30:
  31:             if a > b:
--------------------------------------------------------------------------------
[0.00099 s] killed by test_boundary_values (test_nou.ModuloGCDTest.test_boundary_values)
```

```
   - [#  12] COI ModuloGCD:
--------------------------------------------------------------------------------
  27:             if iterations > max_iterations:
  28:                 raise RuntimeError('Prea multe iteratii (posibil loop infinit)')
  29:             iterations += 1
  30:
- 31:             if a > b:
+ 31:             if not (a > b):
  32:                 a = a - b
  33:             else:
  34:                 b = b - a
  35:
--------------------------------------------------------------------------------
[0.00200 s] killed by test_boundary_values (test_nou.ModuloGCDTest.test_boundary_values)
```

COI – Conditional Operator Inseration - inversează condițiile

```
   - [#  13] LCR ModuloGCD:
--------------------------------------------------------------------------------
   4:         self.b = b
   5:         self.c = c
   6:
   7:     def validate_integer(self, value):
-  8:         if (not (isinstance(value, int)) or value <= 0):
+  8:         if (not (isinstance(value, int)) and value <= 0):
   9:             raise ValueError('Valorile trebuie sa fie numere naturale pozitive')
  10:         return value
  11:
  12:     def cmmdc(self):
--------------------------------------------------------------------------------
[0.00201 s] killed by test_boundary_values (test_nou.ModuloGCDTest.test_boundary_values)
```

LCR – Logical connector replacement- înlocuirea conectoriilor logici

```
   - [#  14] ROR ModuloGCD:
--------------------------------------------------------------------------------
   4:         self.b = b
   5:         self.c = c
   6:
   7:     def validate_integer(self, value):
-  8:         if (not (isinstance(value, int)) or value <= 0):
+  8:         if (not (isinstance(value, int)) or value >= 0):
   9:             raise ValueError('Valorile trebuie sa fie numere naturale pozitive')
  10:         return value
  11:
  12:     def cmmdc(self):
--------------------------------------------------------------------------------
[0.00300 s] killed by test_boundary_values (test_nou.ModuloGCDTest.test_boundary_values)
```

```
   - [#  15] ROR ModuloGCD:
--------------------------------------------------------------------------------
   4:         self.b = b
   5:         self.c = c
   6:
   7:     def validate_integer(self, value):
-  8:         if (not (isinstance(value, int)) or value <= 0):
+  8:         if (not (isinstance(value, int)) or value < 0):
   9:             raise ValueError('Valorile trebuie sa fie numere naturale pozitive')
  10:         return value
  11:
  12:     def cmmdc(self):
--------------------------------------------------------------------------------
[0.00201 s] killed by test_boundary_values (test_nou.ModuloGCDTest.test_boundary_values)
```

```
   - [#  16] ROR ModuloGCD:
--------------------------------------------------------------------------------
  16:
  17:         max_iterations = 1000
  18:         iterations = 0
  19:
- 20:         if c <= 0:
+ 20:         if c >= 0:
  21:             raise ValueError('Modulo trebuie sa fie pozitiv')
  22:
  23:         if a == b:
  24:             return a % c
--------------------------------------------------------------------------------
[0.00131 s] killed by test_boundary_values (test_nou.ModuloGCDTest.test_boundary_values)
```

```
   - [#  17] ROR ModuloGCD:
--------------------------------------------------------------------------------
  16:
  17:         max_iterations = 1000
  18:         iterations = 0
  19:
- 20:         if c <= 0:
+ 20:         if c < 0:
  21:             raise ValueError('Modulo trebuie sa fie pozitiv')
  22:
  23:         if a == b:
  24:             return a % c
--------------------------------------------------------------------------------
[0.00151 s] survived
```

```
    - [#  18] ROR ModuloGCD:
--------------------------------------------------------------------------------
  19:
  20:         if c <= 0:
  21:             raise ValueError('Modulo trebuie sa fie pozitiv')
  22:
- 23:         if a == b:
+ 23:         if a != b:
  24:             return a % c
  25:
  26:         while a != b:
  27:             if iterations > max_iterations:
--------------------------------------------------------------------------------
[0.00000 s] killed by test_boundary_values (test_nou.ModuloGCDTest.test_boundary_values)
```

```
    - [#  19] ROR ModuloGCD:
--------------------------------------------------------------------------------
  22:
  23:         if a == b:
  24:             return a % c
  25:
- 26:         while a != b:
+ 26:         while a == b:
  27:             if iterations > max_iterations:
  28:                 raise RuntimeError('Prea multe iteratii (posibil loop infinit)')
  29:             iterations += 1
  30:
--------------------------------------------------------------------------------
[0.00100 s] killed by test_boundary_values (test_nou.ModuloGCDTest.test_boundary_values)
```

```
    - [#  20] ROR ModuloGCD:
--------------------------------------------------------------------------------
  23:         if a == b:
  24:             return a % c
  25:
  26:         while a != b:
- 27:             if iterations > max_iterations:
+ 27:             if iterations < max_iterations:
  28:                 raise RuntimeError('Prea multe iteratii (posibil loop infinit)')
  29:             iterations += 1
  30:
  31:             if a > b:
--------------------------------------------------------------------------------
[0.00200 s] killed by test_boundary_values (test_nou.ModuloGCDTest.test_boundary_values)
```

```
    - [#  21] ROR ModuloGCD:
--------------------------------------------------------------------------------
  23:         if a == b:
  24:             return a % c
  25:
  26:         while a != b:
- 27:             if iterations > max_iterations:
+ 27:             if iterations >= max_iterations:
  28:                 raise RuntimeError('Prea multe iteratii (posibil loop infinit)')
  29:             iterations += 1
  30:
  31:             if a > b:
--------------------------------------------------------------------------------
[0.00100 s] survived
```

```
   - [#  22] ROR ModuloGCD:
--------------------------------------------------------------------------------
  27:             if iterations > max_iterations:
  28:                 raise RuntimeError('Prea multe iteratii (posibil loop infinit)')
  29:             iterations += 1
  30:
- 31:             if a > b:
+ 31:             if a < b:
  32:                 a = a - b
  33:             else:
  34:                 b = b - a
  35:
--------------------------------------------------------------------------------
[0.00206 s] killed by test_boundary_values (test_nou.ModuloGCDTest.test_boundary_values)
```

```
   - [#  23] ROR ModuloGCD:
--------------------------------------------------------------------------------
  27:             if iterations > max_iterations:
  28:                 raise RuntimeError('Prea multe iteratii (posibil loop infinit)')
  29:             iterations += 1
  30:
- 31:             if a > b:
+ 31:             if a >= b:
  32:                 a = a - b
  33:             else:
  34:                 b = b - a
  35:
--------------------------------------------------------------------------------
[0.00000 s] survived
[*] Mutation score [0.47773 s]: 82.6%
   - all: 23
   - killed: 19 (82.6%)
   - survived: 4 (17.4%)
   - incompetent: 0 (0.0%)
   - timeout: 0 (0.0%)
```

ROR – Relational Operator Replacement - schimba operatorii de relatie