

Proiect Laborator TV

Ariton Cosmin 506

Problema aleasa

Link pbinfo: <https://www.pbinfo.ro/probleme/4402/ciocolata1>

Irina și Mihaela sunt surori. Într-o zi, mama lor le aduce N tablete de ciocolată, numerotate de la 1 la N , pe care le așează, în această ordine, pe o poliță a unui raft. Pentru fiecare tabletă se cunoaște gramajul (numărul de grame pe care le cântărește). Cantitatea totală de ciocolată consumată de o fată este egală cu suma gramajelor tuturor tabletelor consumate de ea. Pentru a consuma ciocolată, fetele trebuie să respecte următoarele reguli:

- cantitatea totală de ciocolată consumată de Irina trebuie să fie mai mare sau egală cu cantitatea totală de ciocolată consumată de sora sa;
- diferența dintre cantitatea totală de ciocolată consumată de Irina și cantitatea totală de ciocolată consumată de Mihaela trebuie să fie cât mai mică;
- fiecare fată trebuie să consume cel puțin o tabletă de ciocolată;
- fiecare fată consumă tablete de ciocolată de pe raft: Irina începe de la cea numerotată cu 1 și continuă, în ordine, de la stânga la dreapta, iar Mihaela începe cu cea numerotată cu N și continuă, în ordine, de la dreapta la stânga;
- fiecare fată poate întrerupe oricând consumul tabletelor de ciocolată, iar cele rămase fie sunt abandonate pe raft, fie sunt consumate de fata cealaltă, dacă ajunge la ele;
- fiecare tabletă de ciocolată fie este consumată complet de una dintre fete, fie rămâne pe raft, dar fetele NU pot sări peste nicio tabletă de ciocolată.

Cerința

Determinați și afișați:

1. cel mai des întâlnit gramaj în șirul de tablete așezate inițial pe poliță, iar dacă sunt mai multe gramaje care apar de un număr maxim de ori, se alege cel mai mic dintre acestea;
2. diferența minimă dintre cantitatea totală de ciocolată consumată de Irina și cantitatea totală de ciocolată consumată de Mihaela.

Restricții și precizări

- $C \in \{1, 2\}$
- $1 \leq N \leq 100.000$
- gramajul fiecărei tablete este un număr natural nenul mai mic sau egal cu 10.000
- se garantează că există întotdeauna soluție.
- Pentru 30 de puncte, $C = 1$
- Pentru 5 de puncte, $C = 2$ și $N = 2$
- Pentru 10 de puncte, $C = 2$ și $1 \leq N \leq 100$
- Pentru 25 de puncte, $C = 2$ și $1 \leq N \leq 1000$
- Pentru 30 de puncte, $C = 2$, fără restricții suplimentare

Cu precizarea ca "se garanteaza ca exista intotdeauna solutie" este cu referire la sir pentru $c = 2$, daca toate celelalte conditii de existenta sunt asigurate(inclusiv $N \leq 2$ pentru $c = 2$).

In rest, se vor intoarce intregi cu valori negative ce semnifica o eroare astfel:

- 1 pentru erori cu privire la c pentru apartenenta la multimea indicata
- 2 pentru erori cu privire la n pentru apartenenta la intervalul indicat
- 3 pentru erori cu privire la elementele sirului
- 4 pentru erori cu privire la $n < 2$ pentru $c = 2$, caz ce apare indirect din

"fiecare fată trebuie să consume cel puțin o tabletă de ciocolată" cand este considerata problema 2

Mai mult, verificarile vor fi executate in ordinea descrisa anterior, ex: verificarea pentru eroarea -4 nefiind executata daca $c = 1$

Rezolvare:

```
public static Integer ciocolata(Integer c, Integer n, List<Integer> sir) {  
    if (c != 1 && c != 2)  
        return -1;  
  
    if (n < 1 || n > 100_000)  
        return -2;  
  
    for (Integer elem : sir)  
        if (elem <= 0 || elem > 10_000)  
            return -3;  
  
    if (c == 1) {  
        ArrayList<Integer> vectorAparitii = new ArrayList<>(10_000);  
        for (Integer i = 0; i <= 10_000; i++) {  
            vectorAparitii.add(0);  
        }  
  
        for (Integer elem : sir)  
            vectorAparitii.set(elem, vectorAparitii.get(elem) + 1);  
  
        Integer max = Integer.MIN_VALUE;  
        Integer maxIdx = -1;  
        for (Integer i = 1; i <= 10_000; i++)  
            if (vectorAparitii.get(i) > max) {  
                max = vectorAparitii.get(i);  
                maxIdx = i;  
            }  
  
        return maxIdx;  
    }  
  
    else {  
        if (n < 2)  
            return -4;  
  
        Deque<Integer> d_sir = new ArrayDeque<>(sir);  
  
        Integer irina = 0;  
        Integer mihaela = 0;  
        Integer min = Integer.MAX_VALUE;  
  
        irina += d_sir.pollFirst();  
        mihaela += d_sir.pollLast();  
  
        if (irina - mihaela >= 0 && min > irina - mihaela)  
            min = irina - mihaela;  
  
        while (d_sir.peek() != null) {
```

```
        while (d_sir.peek() != null && irina - mihaela < 0)
            irina += d_sir.pollFirst();

        if (irina - mihaela >= 0 && min > irina - mihaela) {
            min = irina - mihaela;
        }

        while (d_sir.peek() != null && mihaela - irina < 0) {
            mihaela += d_sir.pollLast();

            if (irina - mihaela >= 0 && min > irina - mihaela) {
                min = irina - mihaela;
            }
        }

        if (min == 0)
            return min;
    }
    return min;
}
```

Equivalence partitioning

Intrare

c un intreg pozitiv

n un intreg pozitiv

un sir de numere naturale nenule mai mici sau egale cu 10_000. Notare: "sir"

Domeniu pt c

$$C1 = \{1\}$$

$$C2 = \{2\}$$

$$C3 = \{c \mid c \text{ nu in } \{1, 2\}\}$$

Domeniu pt n

$$N1 = \{n \mid n < 1\}$$

$$N2 = \{n \mid n > 1 \ \&\& \ n \leq 100_000\}$$

$$N3 = \{n \mid n > 100_000\}$$

$$N4 = \{1\}$$

Domeniu pt sir

$$SIR1 = \{x \mid x > 0, x < 10_000, k \geq 2 \text{ in numere_naturale}, x = x_k, \text{sum}(x_1..x_i) - \text{sum}(x_j..x_n) > 0 \text{ cu } i < j, \text{numere_naturale in } [1, k]\}; \text{ sir ce garanteaza solutie}$$

$SIR2 = \{x \mid x > 0, x < 10_000, k \geq 2 \text{ in } \text{numere_naturale}, x = x_k, \sum(x_1..x_i) - \sum(x_j..x_n) = 0 \text{ cu } i < j, \text{numere_naturale in } [1, k]\}$; sir ce garanteaza solutie

$SIR3 = \{x \mid x > 0, x < 10_000\}$; sir cu un singur numar ce respecta intervalul specificat; sirul nu garanteaza existenta unei solutii, dar este disjunct cu $c = 2$

$SIR4 = \{x \mid \text{exista } x \text{ a.i } x \leq 0, k \text{ in } \text{numere_naturale}, x = x_k\}$

$SIR5 = \{x \mid \text{exista } x \text{ a.i } x > 10_000, k \text{ in } \text{numere_naturale}, x = x_k\}$

$SIR6 = \{x \mid x > 0, x < 10_000, k \geq 2 \text{ in } \text{numere_naturale}, x = x_k, \sum(x_1..x_i) - \sum(x_j..x_n) < 0 \text{ cu } i < j, \text{numere_naturale in } [1, k]\}$; sirul nu reprezinta solutie pentru c_2 , deci compatibil doar cu c_1

Cum problema garanteaza solutie si nu se precizeaza nimic despre tratarea diferita a sirurilor ce nu au solutie

atunci nu determina clase de echivalenta suplimentare

Intregul n determina lungimea sirului de numere "sir" si nu se precizeaza nimic despre tratarea diferita a sirurilor

de lungime diferita deci nu determina clase de echivalenta suplimentare

Iesiri

$I1 = \{-1\}$

$I2 = \{-2\}$

$I3 = \{-3\}$

$I4 = \{-4\}$

$I5 = \{x \mid x > 0, x \text{ respecta cerinta}\}$

$$I6 = \{0\}$$

Clasele

$$CL3 = \{(c, n, sir) \mid c \text{ in } C3 \text{ si iesirea } I1\} \text{ -----} > c = 4, n = 3, sir = (1, 2, 3)$$

$$CL11 = \{(c, n, sir) \mid c \text{ in } C1, n \text{ in } N1 \text{ si iesirea } I2\} \text{ -----} > c = 1, n = -5, sir = (1, 2, 3)$$

$$CL21 = \{(c, n, sir) \mid c \text{ in } C2, n \text{ in } N1 \text{ si iesirea } I2\} \text{ -----} > c = 2, n = -5, sir = (1, 2, 3)$$

$$CL13 = \{(c, n, sir) \mid c \text{ in } C1, n \text{ in } N3 \text{ si iesirea } I2\} \text{ -----} > c = 1, n = 200_000, sir = (1, 2, 3)$$

$$CL23 = \{(c, n, sir) \mid c \text{ in } C2, n \text{ in } N3 \text{ si iesirea } I2\} \text{ -----} > c = 2, n = 200_000, sir = (1, 2, 3)$$

$$CL114 = \{(c, n, sir) \mid c \text{ in } C1, n \text{ in } N2, sir \text{ in } SIR4, \text{ si iesirea } I3\} \text{ -----} > c = 1, n = 6, sir = (-1, 2, 3, 4, 5, 6)$$

$$CL115 = \{(c, n, sir) \mid c \text{ in } C1, n \text{ in } N2, sir \text{ in } SIR5, \text{ si iesirea } I3\} \text{ -----} > c = 1, n = 6, sir = (20_000, 2, 3, 4, 5, 6)$$

$$CL214 = \{(c, n, sir) \mid c \text{ in } C2, n \text{ in } N2, sir \text{ in } SIR4, \text{ si iesirea } I3\} \text{ -----} > c = 2, n = 6, sir = (-1, 2, 3, 4, 5, 6)$$

$$CL215 = \{(c, n, sir) \mid c \text{ in } C2, n \text{ in } N2, sir \text{ in } SIR5, \text{ si iesirea } I3\} \text{ -----} > c = 2, n = 6, sir = (20_000, 2, 3, 4, 5, 6)$$

$$\text{CL144} = \{(c, n, \text{sir}) \mid c \text{ in C1, } n \text{ in N4, sir in SIR4, si iesirea I3}\} \text{ -----} \rightarrow c = 1, n = 1, \text{sir} \\ = (-1,)$$

$$\text{CL145} = \{(c, n, \text{sir}) \mid c \text{ in C1, } n \text{ in N4, sir in SIR5, si iesirea I3}\} \text{ -----} \rightarrow c = 1, n = 1, \text{sir} \\ = (20_000,)$$

$$\text{CL244} = \{(c, n, \text{sir}) \mid c \text{ in C2, } n \text{ in N4, sir in SIR4, si iesirea I3}\} \text{ -----} \rightarrow c = 2, n = 1, \text{sir} \\ = (-1,)$$

$$\text{CL245} = \{(c, n, \text{sir}) \mid c \text{ in C2, } n \text{ in N4, sir in SIR5, si iesirea I3}\} \text{ -----} \rightarrow c = 2, n = 1, \text{sir} \\ = (20_000,)$$

$$\text{CL243} = \{(c, n, \text{sir}) \mid c \text{ in C2, } n \text{ in N4, sir in SIR3 si iesirea I4}\} \text{ -----} \rightarrow c = 2, n = 1, \text{sir} \\ = (1,)$$

$$\text{CL121} = \{(c, n, \text{sir}) \mid c \text{ in C1, } n \text{ in N2, sir in SIR1, si iesirea I5}\} \text{ -----} \rightarrow c = 1, n = 6, \text{sir} \\ = (1, 2, 3, 4, 5, 7)$$

$$\text{CL122} = \{(c, n, \text{sir}) \mid c \text{ in C1, } n \text{ in N2, sir in SIR2, si iesirea I5}\} \text{ -----} \rightarrow c = 1, n = 6, \text{sir} \\ = (1, 2, 3, 4, 5, 6)$$

$$\text{CL126} = \{(c, n, \text{sir}) \mid c \text{ in C1, } n \text{ in N2, sir in SIR6, si iesirea I5}\} \text{ -----} \rightarrow c = 1, n = 6, \text{sir} \\ = (1, 2, 3, 4, 5, 300)$$

$$\text{CL143} = \{(c, n, \text{sir}) \mid c \text{ in C1, } n \text{ in N4, sir in SIR3, si iesirea I5}\} \text{ -----} \rightarrow c = 1, n = 1, \text{sir} \\ = (2,)$$

$$\text{CL221} = \{(c, n, \text{sir}) \mid c \text{ in C1, } n \text{ in N2, sir in SIR1, si iesirea I5}\} \text{ -----} \rightarrow c = 2, n = 6, \text{sir} \\ = (1, 2, 3, 4, 5, 7)$$

CL222 = {(c, n, sir) | c in C1, n in N2, sir in SIR2, si iesirea I5} -----> c = 2, n = 6, sir
= (1, 2, 3, 4, 5, 6)

Cod rezultat:

```
@Test
void equivalencePartitioning() {
    Assertions.assertEquals(-1, Ciocolata.ciocolata(4, 3, Arrays.asList(1, 2, 3)));

    Assertions.assertEquals(-2, Ciocolata.ciocolata(1, -5, Arrays.asList(1, 2, 3)));
    Assertions.assertEquals(-2, Ciocolata.ciocolata(2, -5, Arrays.asList(1, 2, 3)));
    Assertions.assertEquals(-2, Ciocolata.ciocolata(1, 200_000, Arrays.asList(1, 2, 3)));
    Assertions.assertEquals(-2, Ciocolata.ciocolata(2, 200_000, Arrays.asList(1, 2, 3)));

    Assertions.assertEquals(-3, Ciocolata.ciocolata(1, 6, Arrays.asList(-1, 2, 3, 4, 5, 6)));
    Assertions.assertEquals(-3, Ciocolata.ciocolata(1, 6, Arrays.asList(20_000, 2, 3, 4, 5, 6)));
    Assertions.assertEquals(-3, Ciocolata.ciocolata(2, 6, Arrays.asList(-1, 2, 3, 4, 5, 6)));
    Assertions.assertEquals(-3, Ciocolata.ciocolata(2, 6, Arrays.asList(20_000, 2, 3, 4, 5, 6)));
    Assertions.assertEquals(-3, Ciocolata.ciocolata(1, 1, Arrays.asList(-1)));
    Assertions.assertEquals(-3, Ciocolata.ciocolata(1, 1, Arrays.asList(20_000)));
    Assertions.assertEquals(-3, Ciocolata.ciocolata(2, 1, Arrays.asList(-1)));
    Assertions.assertEquals(-3, Ciocolata.ciocolata(2, 1, Arrays.asList(20_000)));

    Assertions.assertEquals(-4, Ciocolata.ciocolata(2, 1, Arrays.asList(1)));

    Assertions.assertEquals(1, Ciocolata.ciocolata(1, 6, Arrays.asList(1, 2, 3, 4, 5, 7)));
    Assertions.assertEquals(1, Ciocolata.ciocolata(1, 6, Arrays.asList(1, 2, 3, 4, 5, 6)));
    Assertions.assertEquals(1, Ciocolata.ciocolata(1, 6, Arrays.asList(1, 2, 3, 4, 5, 300)));
    Assertions.assertEquals(2, Ciocolata.ciocolata(1, 1, Arrays.asList(2)));

    Assertions.assertEquals(3, Ciocolata.ciocolata(2, 6, Arrays.asList(1, 2, 3, 4, 5, 7)));
    Assertions.assertEquals(0, Ciocolata.ciocolata(2, 6, Arrays.asList(1, 2, 3, 4, 5, 6)));
}
```

Boundary value analysis

C1: 1 valoare de frontiera

C2: 2 valoare de frontiera

C3: 3, 4 valori de frontiera

N1: 0 valoare de frontiera

N2: 2,100_000 valori de frontiera

N3: 100_001 valoare de frontiera

N4: 1 valoare de frontiera

SIR1: 1, 10_000 valori de frontiera pentru fiecare element x din sir, lungimea sirului=2
valoare de frontiera, $\sum(x1..xi) - \sum(xj..xn) = 1$ cu $i < j$, numere_naturale in $[1, k]$ valoare de frontiera

SIR2: 1, 10_000 valori de frontiera pentru fiecare element x din sir, lungimea sirului=2
valoare de frontiera, $\sum(x1..xi) - \sum(xj..xn) = 0$ cu $i < j$, numere_naturale in $[1, k]$ valoare de frontiera

SIR3: 1, 10_000 valori de frontiera pentru fiecare element x din sir, lungimea sirului=1
valoare de frontiera

SIR4: 0 valoare de frontiera pentru cel putin un element x din sir

SIR5: 10_001 valoare de frontiera pentru cel putin un element x din sir

SIR6: 1, 10_000 valori de frontiera pentru fiecare element x din sir, lungimea sirului=2
valoare de frontiera, $\sum(x1..xi) - \sum(xj..xn) = -1$ cu $i < j$, numere_naturale in $[1, k]$ valoare de frontiera

alte valori speciale:

In cadrul C2

SIR1: $\sum(x1..xi) - \sum(xj..xn) = 1$ cu $i < j$, numere_naturale in $[1, k]$ valoare de frontiera
cu $i > 1$ si $j < n$, adica, in contextul problemei, si Irina si Mihaela au mancat cel putin 2 tablete

SIR2: $\sum(x1..xi) - \sum(xj..xn) = 0$ cu $i < j$, numere_naturale in $[1, k]$, adica, in contextul
problemei, si Irina si Mihaela au mancat cel putin 2 tablete

Cod rezultat:

```
@Test
void boundryValueAnalysis() {
    //Pentru C
    Assertions.assertEquals(1, Ciocolata.ciocolata(1, 3, Arrays.asList(1, 1, 2)));
    Assertions.assertEquals(1, Ciocolata.ciocolata(2, 5, Arrays.asList(1, 2, 3, 1, 4)));
    Assertions.assertEquals(-1, Ciocolata.ciocolata(3, 3, Arrays.asList(1, 2, 3)));
    Assertions.assertEquals(-1, Ciocolata.ciocolata(4, 3, Arrays.asList(1, 2, 3)));

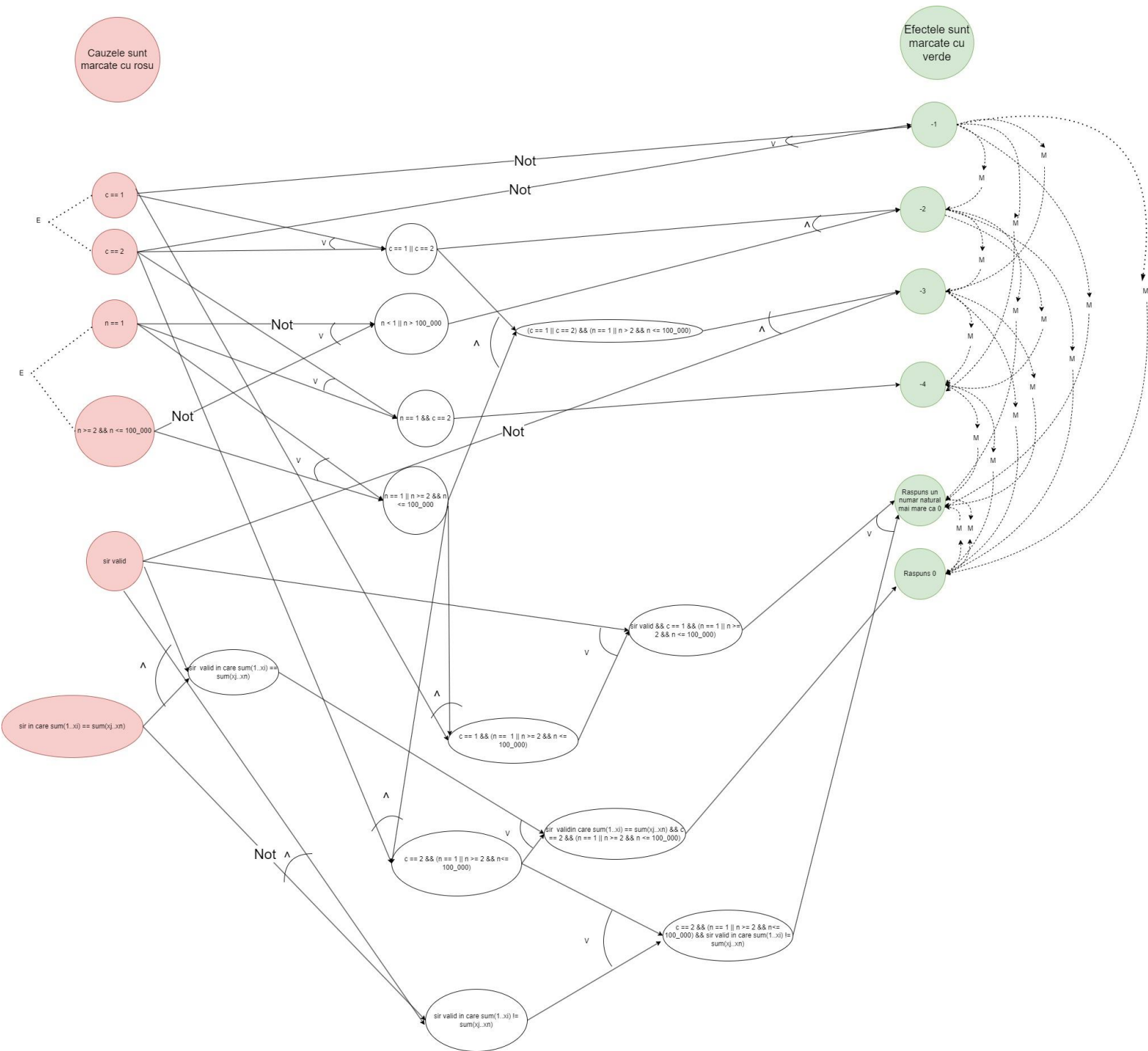
    //Pentru N
    Assertions.assertEquals(-2, Ciocolata.ciocolata(1, 0, Arrays.asList(1, 2, 3)));
    Assertions.assertEquals(1, Ciocolata.ciocolata(1, 2, Arrays.asList(1, 2)));
    List<Integer> longList = new ArrayList<>();
    for (Integer i = 0; i < 100_000; i++)
        longList.add(2);
    Assertions.assertEquals(2, Ciocolata.ciocolata(1, 100_000, longList));
    Assertions.assertEquals(-2, Ciocolata.ciocolata(1, 100_001, Arrays.asList(1, 2, 3)));
    Assertions.assertEquals(1, Ciocolata.ciocolata(1, 1, Arrays.asList(1)));

    //Pentru SIR
    Assertions.assertEquals(9999, Ciocolata.ciocolata(2, 2, Arrays.asList(10_000, 1)));
    Assertions.assertEquals(1, Ciocolata.ciocolata(2, 2, Arrays.asList(2, 1)));
}
```

```
    Assertions.assertEquals(9999, Ciocolata.ciocolata(2, 2,
Arrays.asList(10_000, 1)));
    Assertions.assertEquals(0, Ciocolata.ciocolata(2, 2, Arrays.asList(1,
1)));
    Assertions.assertEquals(1, Ciocolata.ciocolata(1, 1, Arrays.asList(1)));
    Assertions.assertEquals(10_000, Ciocolata.ciocolata(1, 1,
Arrays.asList(10_000)));
    Assertions.assertEquals(-3, Ciocolata.ciocolata(1, 1, Arrays.asList(0)));
    Assertions.assertEquals(-3, Ciocolata.ciocolata(1, 1,
Arrays.asList(10_001)));
    Assertions.assertEquals(1, Ciocolata.ciocolata(1, 2,
Arrays.asList(10_000, 1)));
    Assertions.assertEquals(1, Ciocolata.ciocolata(1, 2, Arrays.asList(2,
1)));

    //Pentru alte valori speciale
    Assertions.assertEquals(1, Ciocolata.ciocolata(2, 4, Arrays.asList(1, 33,
3, 30)));
    Assertions.assertEquals(0, Ciocolata.ciocolata(2, 4, Arrays.asList(1, 32,
3, 30)));
}
```

Graful cauza-efect:



Tabelul cauza-efect rezultat:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1			1	2	3	4	5	6	7	8	9	10	11	12
2	Cauze	c == 1	0	1	0	1	0	1	0	0	1	1	0	0
3	Cauze	c == 2	0	0	1	0	1	0	1	1	0	0	1	1
4	Cauze	n == 1	0	0	0	1	1	0	0	1	1	0	0	0
5	Cauze	n >= 2 && n <= 100_000	0	0	0	0	0	1	1	0	0	1	1	1
6	Cauze	sir valid	0	0	0	0	0	0	0	1	1	1	1	1
7	Cauze	sir in care sum(1..xi) == sum(xj..xn)	0	0	0	0	0	0	0	0	0	0	0	1
8	Efecte		-1	1	0	0	0	0	0	0	0	0	0	0
9	Efecte		-2	0	1	1	0	0	0	0	0	0	0	0
10	Efecte		-3	0	0	0	1	1	1	1	0	0	0	0
11	Efecte		-4	0	0	0	0	0	0	0	1	0	0	0
12	Efecte	Raspuns un numar natural mai mare ca 0	0	0	0	0	0	0	0	0	1	1	1	0
13	Efecte	Raspuns 0	0	0	0	0	0	0	0	0	0	0	0	1
14	Casutele marcate cu rosu nu sunt relevante. Am notat cu 0. Altfel ar duce la o explozie combinatorica inutila													

Cod rezultat:

```
@Test
void causeEffectGraphing() {
    Assertions.assertEquals(-1, Ciocolata.ciocolata(3, 2, Arrays.asList(2, 2)));
    Assertions.assertEquals(-2, Ciocolata.ciocolata(1, -1, Arrays.asList(2, 3, 4)));
    Assertions.assertEquals(-2, Ciocolata.ciocolata(2, -1, Arrays.asList(2, 3, 4)));
    Assertions.assertEquals(-3, Ciocolata.ciocolata(1, 1, Arrays.asList(-1)));
    Assertions.assertEquals(-3, Ciocolata.ciocolata(2, 1, Arrays.asList(-1)));
    Assertions.assertEquals(-3, Ciocolata.ciocolata(1, 3, Arrays.asList(-1, 2, 3)));
    Assertions.assertEquals(-3, Ciocolata.ciocolata(2, 3, Arrays.asList(-1, 2, 3)));
    Assertions.assertEquals(-4, Ciocolata.ciocolata(2, 1, Arrays.asList(3)));

    Assertions.assertEquals(30, Ciocolata.ciocolata(1, 1, Arrays.asList(30)));
    Assertions.assertEquals(30, Ciocolata.ciocolata(1, 4, Arrays.asList(30, 30, 63, 89)));
    Assertions.assertEquals(30, Ciocolata.ciocolata(2, 2, Arrays.asList(60, 30)));
    Assertions.assertEquals(0, Ciocolata.ciocolata(2, 2, Arrays.asList(2, 2)));
}
```

Code coverage

Testele sunt executate atat pentru clasa “Ciocolata” cat si pentru mutanti, dar ne vom axa pe programul nemodificat “Ciocolata”.

Equivalence partitioning:

Element ▲	Class, %	Method, %	Line, %
▼ all	100% (8/8)	100% (8/8)	82% (278/338)
Ciocolata	100% (1/1)	100% (1/1)	95% (40/42)
MutantNeechivalentOmoratEquivalencePartitioning	100% (1/1)	100% (1/1)	42% (18/42)
MutantNeechivalentViuEquivalencePartitioning	100% (1/1)	100% (1/1)	95% (40/42)
MutantOrdinul1Echivalent	100% (1/1)	100% (1/1)	95% (41/43)

Se poate observa ca natura mai libera a acestei metode de testare nu ne forteaza sa acoperim toate liniile de cod. Astfel, cazurile extreme pot lipsi din teste:

53			
54			if (<u>irina</u> - <u>mihaela</u> >= 0 && <u>min</u> > <u>irina</u> - <u>mihaela</u>)
55			<u>min</u> = <u>irina</u> - <u>mihaela</u> ;
56			

Boundry value analysis

Element ▲	Class, %	Method, %	Line, %
▼ all	100% (8/8)	100% (8/8)	94% (318/338)
Ciocolata	100% (1/1)	100% (1/1)	97% (41/42)
MutantNeechivalentOmoratEquivalencePartitioning	100% (1/1)	100% (1/1)	88% (37/42)
MutantNeechivalentViuEquivalencePartitioning	100% (1/1)	100% (1/1)	92% (39/42)
MutantOrdinul1Echivalent	100% (1/1)	100% (1/1)	97% (42/43)

Se poate observa ca apare un code coverage mai mare, dar dispar anumite combinatii de valori de intrare cum ar fi $c = 2$ si $n = 1$. Aceasta disparitie duce la lipsa verificarii urmatoarei linii de cod:

41

42

43

```
if (n < 2)
    return -4;
```


Graful cauza-efect

Element ▲	Class, %	Method, %	Line, %
▼ all	100% (8/8)	100% (8/8)	68% (230/338)
Ciocolata	100% (1/1)	100% (1/1)	76% (32/42)
MutantNeechivalentOmoratEquivalencePartitioning	100% (1/1)	100% (1/1)	42% (18/42)
MutantNeechivalentViuEquivalencePartitioning	100% (1/1)	100% (1/1)	76% (32/42)
MutantOrdinul1Echivalent	100% (1/1)	100% (1/1)	76% (33/43)

Pentru aceasta metoda se observa ca avem toate iesirile prezente, dar multe linii de cod nu sunt acoperite deoarece nu avem alte constrangeri care sa ne forteze sa avem niste valori de intrare mai extreme.

Se poate observa ca simplitatea datelor de intrare fac urmatoarele linii sa nu fie acoperite:

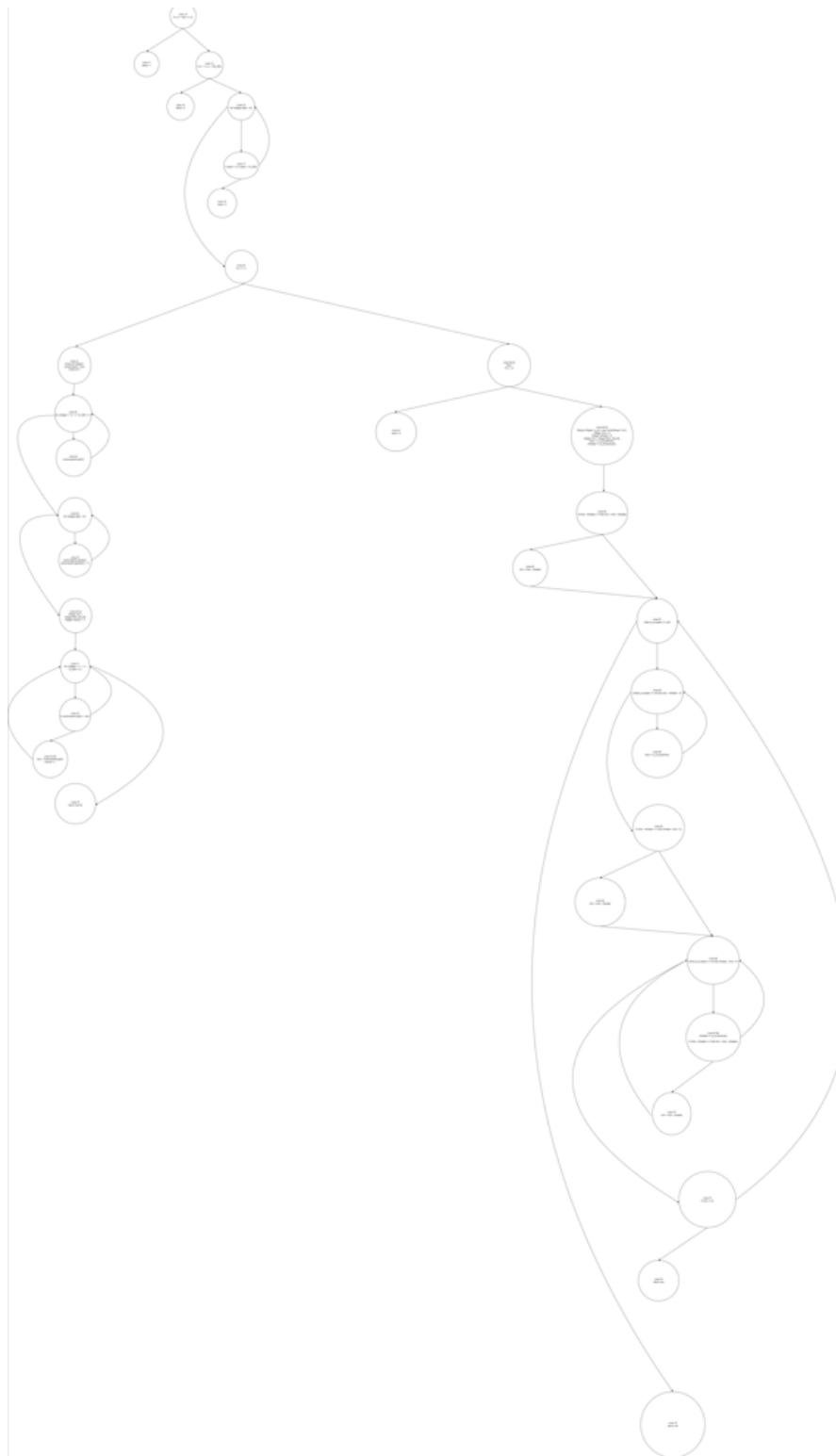
```
54     if (irina - mihaela >= 0 && min > irina - mihaela)
55         min = irina - mihaela;
56
57     while (d_sir.peek() != null) {
58
59         while (d_sir.peek() != null && irina - mihaela < 0)
60             irina += d_sir.pollFirst();
61
62         if (irina - mihaela >= 0 && min > irina - mihaela) {
63             min = irina - mihaela;
64         }
65
66         while (d_sir.peek() != null && mihaela - irina < 0) {
67             mihaela += d_sir.pollLast();
68
69             if (irina - mihaela >= 0 && min > irina - mihaela) {
70                 min = irina - mihaela;
71             }
72         }
73
74         if (min == 0)
75             return min;
76
77     }
78     return min;
79 }
80 }
81 }
82 }
```

Folosirea tuturor testelor concomitant

Element ▲	Class, %	Method, %	Line, %
▼ all	100% (8/8)	100% (8/8)	97% (328/338)
☉ Ciocolata	100% (1/1)	100% (1/1)	100% (42/42)
☉ MutantNeechivalentOmoratEquivalencePartitioning	100% (1/1)	100% (1/1)	88% (37/42)
☉ MutantNeechivalentViuEquivalencePartitioning	100% (1/1)	100% (1/1)	100% (42/42)
☉ MutantOrdinul1Echivalent	100% (1/1)	100% (1/1)	100% (43/43)

De departe cea mai buna alegere, avand o acoperire de 100% pentru toate liniile. Testele isi acopera reciproc dezavantajele.

Graful orientat al programului



Acesta a fost realizat linie cu linie. Am specificat si continutul liniei de cod pentru o mai buna vizualizare.

Mutanti

Cazul ales este **equivalence partitioning**.

Mutant echivalent de ordinul 1

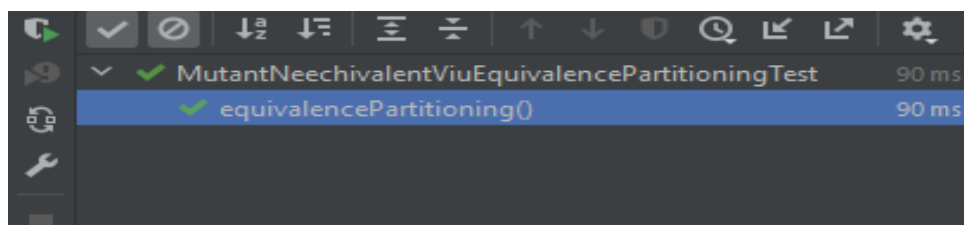
```
53 usages Darknider123
9      public static Integer ciocolata(Integer c, Integer n, List<Integer> sir) {
10
11          //Diferenta:
12          Integer degeaba = 1;
13
```

Singura modificare existenta este instantierea unui Integer care nu este apelat niciodata. Acesta nu influenteaza programul, iar toate trestele trec, in mod firesc.

Mutant neechivalent viu

```
//DIFERENTA: + in loc de -
if (irina - mihaela >= 0 && min > irina - mihaela)
    min = irina + mihaela;
```

Cum aceasta linie nu este acoperita de testele generate prin metoda equivalence partitioning, daca se creaza un mutant prin alteratia ei, acesta nu va fi omorat de catre teste.



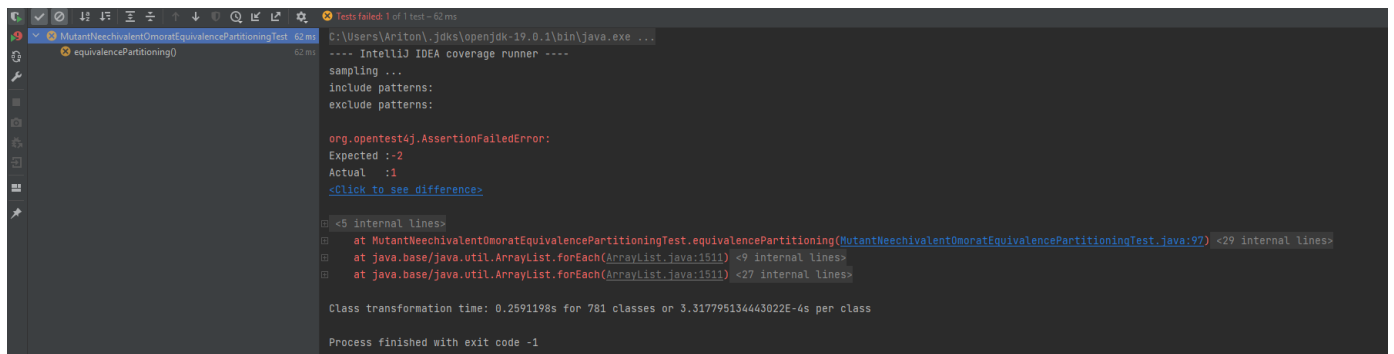
Mutant neechivalent omorat

```
15 //DIFERENTA: SI IN LOC DE SAU  
16 if (n < 1 && n > 100_000)  
17     return -2;  
18
```

Cum linia de cod este verificata de catre testele generate prin metoda equivalence partitioning, mutantul a fost omorat. Este de mentionat ca o alta modificare, desi linia este acoperita, ar putea lasa mutantul viu, daca mutatia ar fii mai subtila, spre exemplu:

```
if (n < 1 || n >= 100_000)  
    return -2;
```

Aici modificandu-se un caz extrem, equivalence partitioning nu garanteaza acoperirea cazului. Totusi aceste cazuri sunt bine acoperite de boundry value analysis.



```
Tests failed: 1 of 1 test - 62 ms  
E:\Users\Arton\jdk\openjdk-19.0.1\bin\java.exe ...  
---- IntelliJ IDEA coverage runner ----  
sampling ...  
include patterns:  
exclude patterns:  
  
org.opentest4j.AssertionFailedError:  
Expected :-2  
Actual :1  
Click to see difference  
  
5 internal lines  
at MutantNeechivalentOmoratEquivalencePartitioningTest.equivalencePartitioning(MutantNeechivalentOmoratEquivalencePartitioningTest.java:97) <29 internal lines>  
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>  
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <27 internal lines>  
  
Class transformation time: 0.2591198s for 781 classes or 3.317795134443022E-4s per class  
  
Process finished with exit code -1
```