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 ∈ {1, 2}
* 1 ≤ N ≤ 100.000
* gramajul fiecărei tablete este un număr natural nenul mai mic sau egal cu 10.000
* se garantează că există întodeauna soluție.
* Pentru 30 de puncte, C = 1
* Pentru 5 de puncte, C = 2 și N = 2
* Pentru 10 de puncte, C = 2 și 1 ≤ N ≤ 100
* Pentru 25 de puncte, C = 2 și 1 ≤ N ≤ 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 <= 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 | c nu in {1, 2}}

Domeniu pt n

N1 = {n | n < 1}

N2 = {n | n > 1 && n <= 100\_000}

N3 = {n | n > 100\_000}

N4 = {1}

Domeniu pt sir

SIR1 = {x | x > 0, x < 10\_000, k >= 2 in numere\_naturale, x = xk, sum(x1..xi) - sum(xj..xn) > 0 cu i < j, numere\_naturale in [1, k]}; sir ce garanteaza solutie

SIR2 = {x | x > 0, x < 10\_000, k >= 2 in numere\_naturale, x = xk, sum(x1..xi) - sum(xj..xn) = 0 cu i < j, numere\_naturale in [1, k]}; sir ce garanteaza solutie

SIR3 = {x | 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 | exista x a.i x <= 0, k in numere\_naturale, x = xk}

SIR5 = {x | exista x a.i x > 10\_000, k in numere\_naturale, x = xk}

SIR6 = {x | x > 0, x < 10\_000, k >= 2 in numere\_naturale, x = xk, sum(x1..xi) - sum(xj..xn) < 0 cu i < j, numere\_naturale in [1, k]}; sirul nu reprezinta solutie pentru c2, deci compatibil doar cu c1

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 | x > 0, x respecta cerinta}

I6 = {0}

Clasele

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

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

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

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

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

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

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

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

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

CL144 = {(c, n, sir) | c in C1, n in N4, sir in SIR4, si iesirea I3} --------> c = 1, n = 1, sir = (-1,)

CL145 = {(c, n, sir) | c in C1, n in N4, sir in SIR5, si iesirea I3} --------> c = 1, n = 1, sir = (20\_000,)

CL244 = {(c, n, sir) | c in C2, n in N4, sir in SIR4, si iesirea I3} --------> c = 2, n = 1, sir = (-1,)

CL245 = {(c, n, sir) | c in C2, n in N4, sir in SIR5, si iesirea I3} --------> c = 2, n = 1, sir = (20\_000,)

CL243 = {(c, n, sir) | c in C2, n in N4, sir in SIR3 si iesirea I4} --------> c = 2, n = 1, sir = (1,)

CL121 = {(c, n, sir) | c in C1, n in N2, sir in SIR1, si iesirea I5} --------> c = 1, n = 6, sir = (1, 2, 3, 4, 5, 7)

CL122 = {(c, n, sir) | c in C1, n in N2, sir in SIR2, si iesirea I5} --------> c = 1, n = 6, sir = (1, 2, 3, 4, 5, 6)

CL126 = {(c, n, sir) | c in C1, n in N2, sir in SIR6, si iesirea I5} --------> c = 1, n = 6, sir = (1, 2, 3, 4, 5, 300)

CL143 = {(c, n, sir) | c in C1, n in N4, sir in SIR3, si iesirea I5} --------> c = 1, n = 1, sir = (2,)

CL221 = {(c, n, sir) | c in C1, n in N2, sir in SIR1, si iesirea I5} --------> c = 2, n = 6, 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

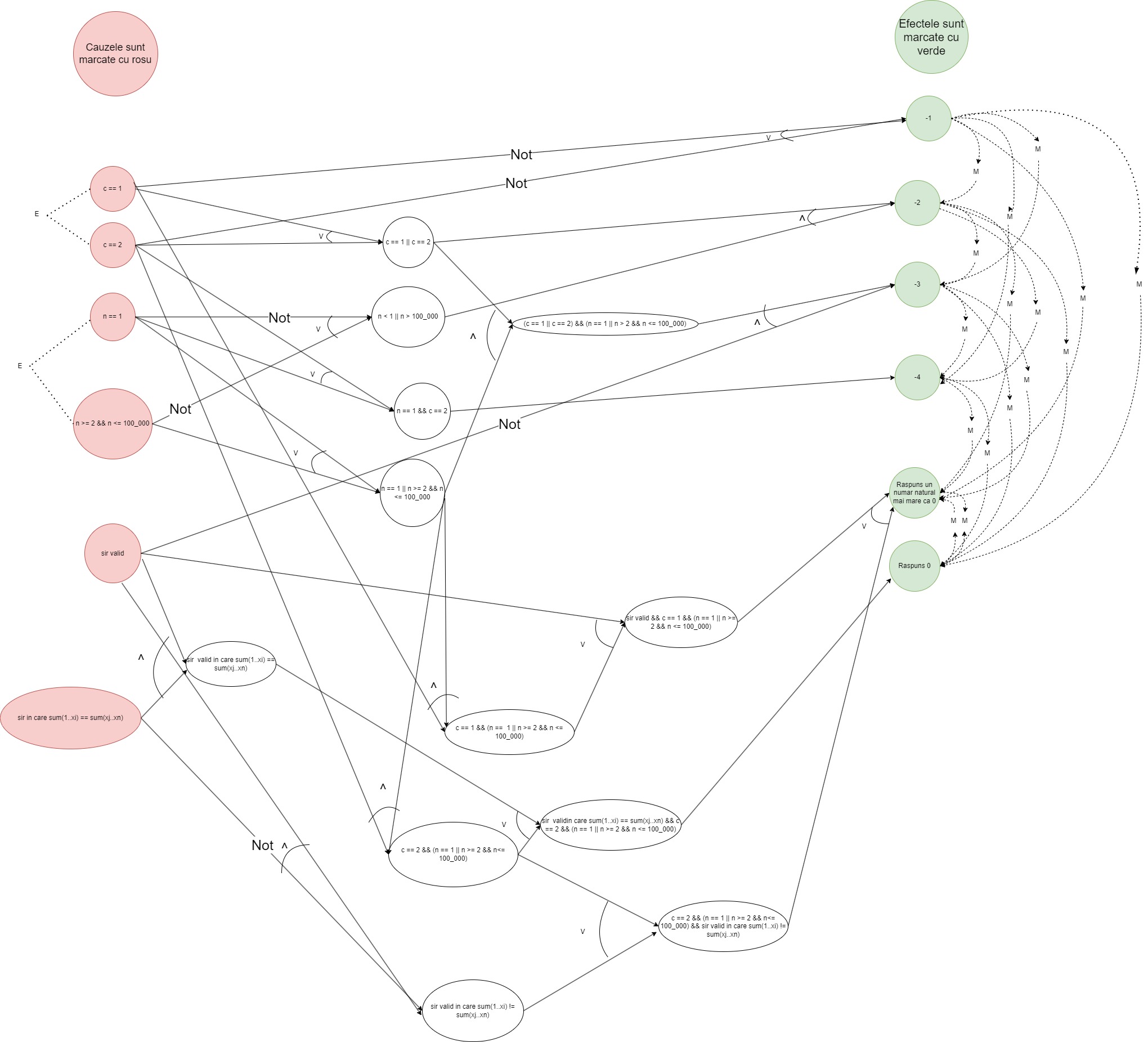
SIR1:sum(x1..xi) - sum(xj..xn) = 1 cu i < j, numere\_naturale in [1, k] valoare de frontiera cu i > 1 si j < n, adica, in contextul problemei, si Irina si Mihaela au mancat cel putin 2 tablete

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

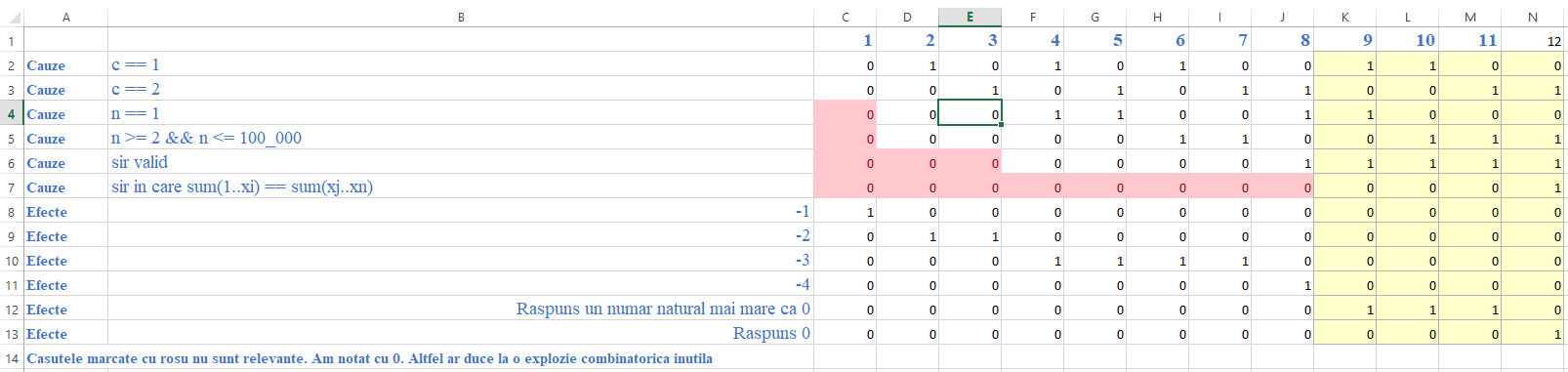
## Cod rezultat:

@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:



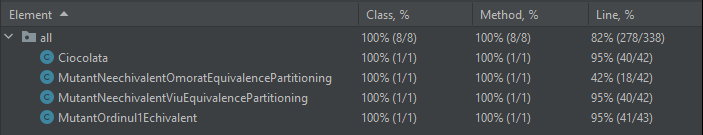
## Cod rezultat:

@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)));  
}

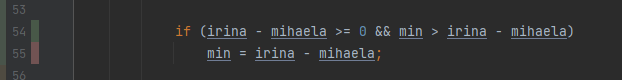
# Code coverage

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

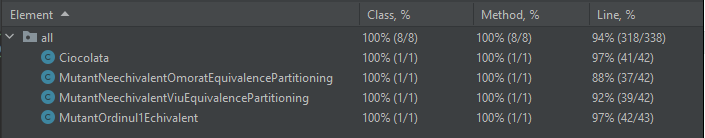
## Equivalence partitioning:



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:



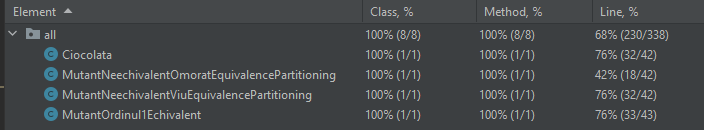
## Boundry value analysis



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:

## 

## Graful cauza-efect

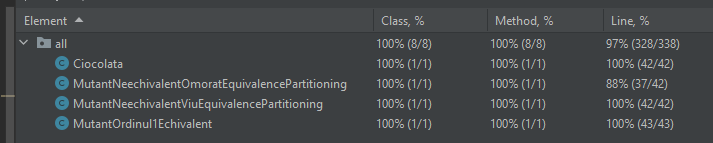


Pentru aceasta metoda se observa ca avem toate iesirile prezente, dar multe linii de cod nu sunt acoperite deoarece nu avem alte constrangeri care sa ne forteze sa avem niste valori de intrare mai extreme.

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

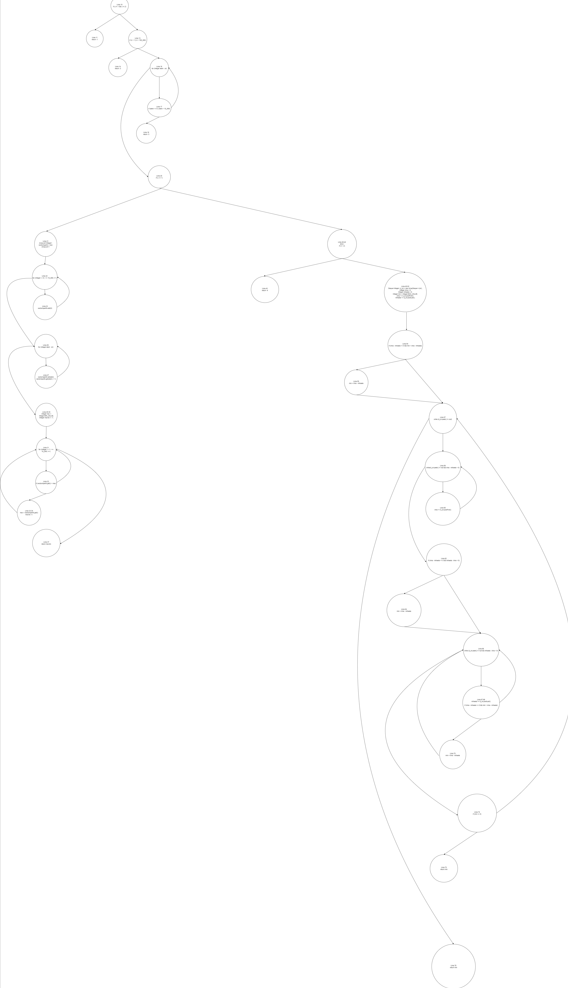


## Folosirea tuturor testelor concomitant



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

# Graful orientat al programului

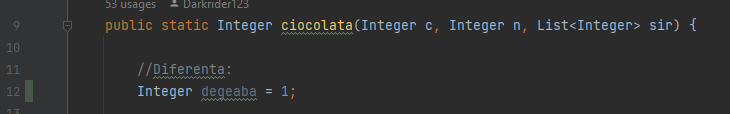


Acesta a fost realizat linie cu linie. Am specificat si continutul liniei de cod pentru o mai buna vizualizare.

# Mutanti

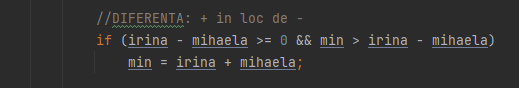
Cazul ales este **equivalence partitioning.**

## Mutant echivalent de ordinul 1

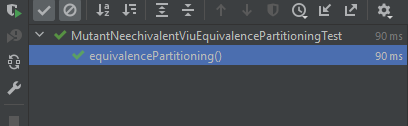


Singura modificare existenta este instantierea unui Integer care nu este apelat niciodata. Acesta nu influenteaza programul, iar toate trestele trec, in mod firesc.

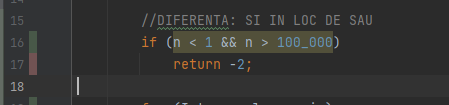
## Mutant neechivalent viu



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



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:



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

