

PROIECT TESTARE ȘI VERIFICARE

Student: Olteanu-Crăciunescu Ana-Maria

Grupa 505

Cuprins

1. Testare funcțională	2
a. Partiționarea în clase de echivalență (equivalence partitioning)	2
b. Analiza valorilor de frontieră	4
c. Graf cauză-efect	5
2. Nivelul de acoperire pentru fiecare dintre seturile de teste de la 1.....	8
3. Testare structurală	9
○ Modified condition/decision coverage (MC/DC).....	12
4. Generare mutant de ordinul 1 echivalent al programului	13
5. Generare mutanți ne-echivalenți	13
a. Mutant ne-echivalent care să fie omorât de către test.....	13
b. Mutant ne-echivalent care să nu fie omorât de către test	13

Se va testa problema Granita (<https://www.infoarena.ro/problema/granita>). Se va adăuga condiția suplimentară: un dispozitiv este redundant și dacă lungimea intervalului este mai mare decât un k dat.

1. Testare funcțională

a. Partiționarea în clase de echivalență (equivalence partitioning)

i. Domeniul de intrări

Date de intrare:

n – numărul de dispozitive de apărare

k – lungimea maximă a unui interval

S – mulțimea intervalelor închise $[A_i, B_i]$

- n : distingem 3 clase de echivalență (presupunem că n trebuie să fie un număr întreg din intervalul $[1, 6]$)
 $N_1 = \{n \mid n \text{ întreg}, n \geq 1, n \leq 6\}$
 $N_2 = \{n \mid n < 1\}$
 $N_3 = \{n \mid n > 6\}$
- k : distingem 3 clase de echivalență (presupunem că k trebuie să fie un număr întreg din intervalul $[1, 20]$)
 $K_1 = \{k \mid k \text{ întreg}, k \geq 1, k \leq 20\}$
 $K_2 = \{k \mid k < 1\}$
 $K_3 = \{k \mid k > 20\}$
- Presupunem că mulțimea S conține valori valide între 0 și 50, dimensiunea acesteia este egală cu n , iar intervalele din mulțime sunt disjuncte și respectă proprietatea că $A_i < B_i$.

ii. Domeniul de ieșiri

$I_1 = \text{numărul de dispozitive redundante}$

$I_2 = \text{"nu s-au gasit dispozitive redundante"}$

$I_3 = \text{"n este invalid"}$

$I_4 = \text{"k este invalid"}$

Clasele de echivalență globale se obțin ca o combinație a claselor individuale.

$C_{111} = \{(n, k, S) \mid n \text{ in } N_1, k \text{ in } K_1, \text{iesirea } I_1\}$

$C_{112} = \{(n, k, S) \mid n \text{ in } N_1, k \text{ in } K_2, \text{iesirea } I_2\}$

$C_2 = \{(n, k, S) \mid n \text{ in } N_2, \text{iesirea } I_3\}$

$C_3 = \{(n, k, S) \mid n \text{ in } N_3, \text{iesirea } I_3\}$

$C_{12} = \{(n, k, S) \mid n \text{ in } N_1, k \text{ in } K_2, \text{iesirea } I_4\}$

$C_{13} = \{(n, k, S) \mid n \text{ in } N_1, k \text{ in } K_3, \text{iesirea } I_4\}$

Set de date de test:

$c_{111} = (5, 15, \{(0, 10), (2, 9), (3, 8), (1, 15), (6, 11)\})$

$c_{112} = (2, 10, \{(0, 10), (6, 11)\})$

$c_2 = (0, _, _)$

$c_3 = (103, _, _)$

$c_{12} = (3, -1, _)$

$c_{13} = (3, 21, _)$

Nr test	Intrari			Iesire
	n	k	S	
1	5	15	$\{(0, 10), (2, 9), (3, 8), (1, 15), (6, 11)\}$	3
2	2	10	$\{(0, 10), (6, 11)\}$	nu s-au gasit dispozitive redundante
3	0			n este invalid
4	103			n este invalid
5	3	-1		k este invalid
6	3	21		k este invalid

b. Analiza valorilor de frontieră

Odată identificate clasele de echivalență, valorile de frontieră sunt următoarele:

- n: 0, 1, 6, 7
- k: 0, 1, 20, 21

Pentru mulțimea S vom alege valori arbitrare.

	Intrari			Iesire
Nr test	n	k	S	
1	0			n este invalid
2	1	0		k este invalid
3	1	1	(0, 2)	1
4	1	20	(0, 21)	1
5	1	21		k este invalid
6	6	0		k este invalid
7	6	1	{(0, 10), (2, 9), (3, 8), (1, 15), (6, 11), (1, 2)}	5
8	6	20	{(0, 10), (2, 9), (3, 8), (1, 15), (6, 11), (1, 2)}	4
9	6	21		k este invalid
10	7			n este invalid

c. Graf cauză-efect

C1 – n valid
 C2 – n invalid
 C3 – k valid
 C4 – k invalid
 C5 – Există cel puțin un dispozitiv redundant
 C6 – Nu există niciun dispozitiv redundant

E1 – n este invalid
 E2 – k este invalid
 E3 – Afișare număr dispozitive redundante
 E4 – Nu s-au găsit dispozitive redundante

Deoarece nu analizăm și mulțimea S de intervale, vom considera că C5 și C6 au loc numai pentru un n și k valid, în celelalte cazuri neputându-se determina dacă s-au găsit sau nu dispozitive redundante.

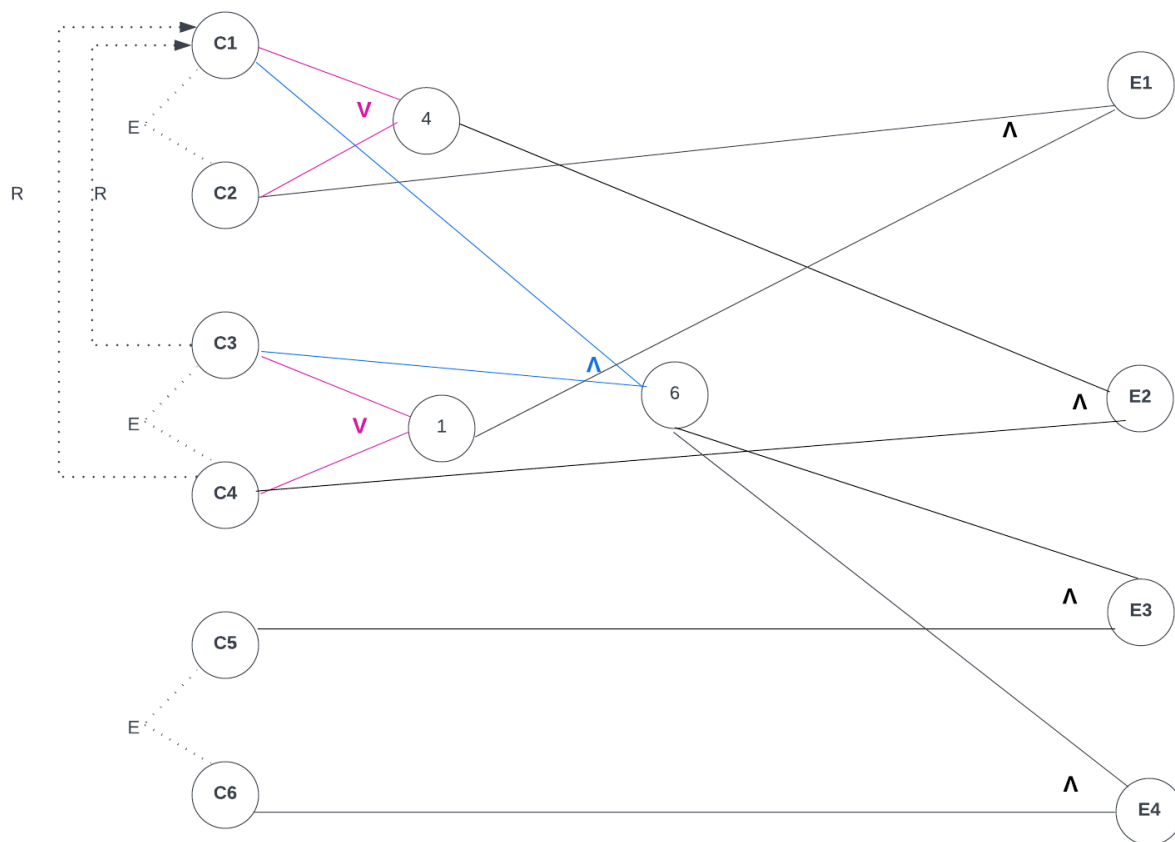


Fig 1. Graful cauză-efect

Crearea tabelului de decizie:

- i. Pentru efectul E1: $C2 \wedge (C3 \vee C4) = 1$

C2	C3	C4
1	1	1
1	1	0
1	0	1

Dar C3 și C4 nu pot exista simultan:

C2	C3	C4
1	1	0
1	0	1

Adăugăm și coloanele pentru celelalte cauze și efecte:

C2	C3	C4	C1	C4	C6	E1	E3	E4	E5
1	1	0	0	0	0	1	0	0	0
1	0	1	0	0	0	1	0	0	0

- ii. Pentru efectul E2: $(C1 \vee C2) \wedge C4 = 1$

C1	C2	C4
1	1	1
1	0	1
0	1	1

Dar C1 și C2 nu pot exista simultan, iar C4 implică C1.

C1	C2	C4
1	0	1

Adăugăm și coloanele pentru celelalte cauze și efecte:

C1	C2	C4	C3	C5	C6	E1	E2	E3	E4
1	0	1	0	0	0	0	1	0	0

- iii. Pentru efectul E3: $C1 \wedge C3 \wedge C5 = 1$

C1	C3	C5
1	1	1

Adăugăm și coloanele pentru celelalte cauze și efecte:

C1	C3	C5	C2	C4	C6	E1	E2	E3	E4
1	1	1	0	0	0	0	0	1	0

iv. Pentru efectul E4: $C1 \wedge C3 \wedge C6 = 1$

C1	C3	C6
1	1	1

Adăugăm și coloanele pentru celelalte cauze și efecte:

C1	C3	C6	C2	C4	C5	E1	E2	E3	E4
1	1	1	0	0	0	0	0	0	1

Matricea de decizie va fi următoarea:

	1	2	3	4	5
C1	0	0	1	1	1
C2	1	1	0	0	0
C3	1	0	0	1	1
C4	0	1	1	0	0
C5	0	0	0	1	0
C6	0	0	0	0	1
E1	1	1	0	0	0
E2	0	0	1	0	0
E3	0	0	0	1	0
E4	0	0	0	0	1

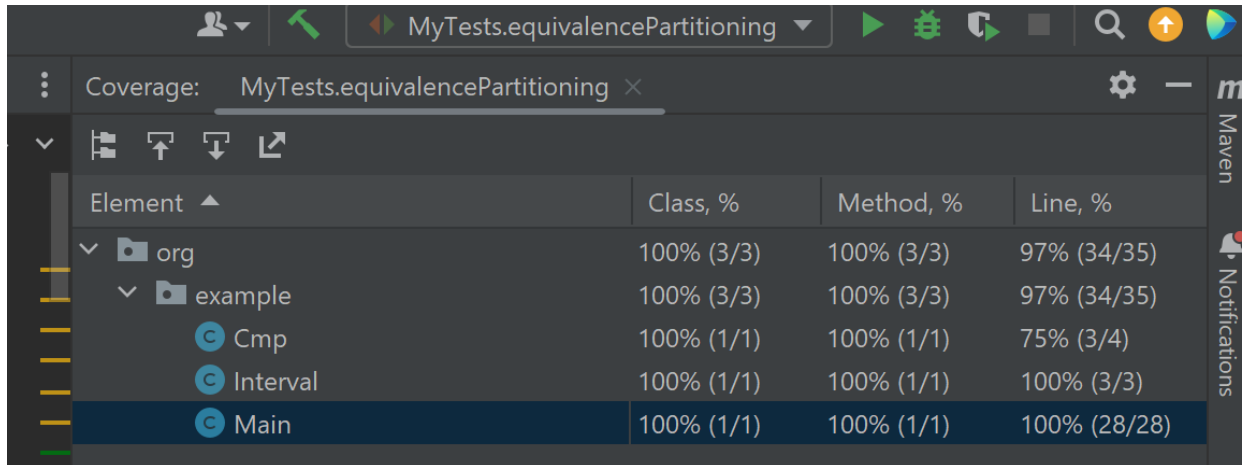
Set de date test:

Nr test	Intrari			Iesire
	n	k	S	
1	200	2	(0, 1)	n este invalid
2	200	45	(2, 10)	n este invalid
3	2	43	(0, 3), (1, 2)	k este invalid
4	2	4	(0, 3), (1, 2)	1
5	2	3	(3, 5), (7, 9)	Nu s-au gasit dispozitive redundante

2. Nivelul de acoperire pentru fiecare dintre seturile de teste de la 1

Am testat metoda *gaseste_dispozitive_redundante()* din clasa Main.

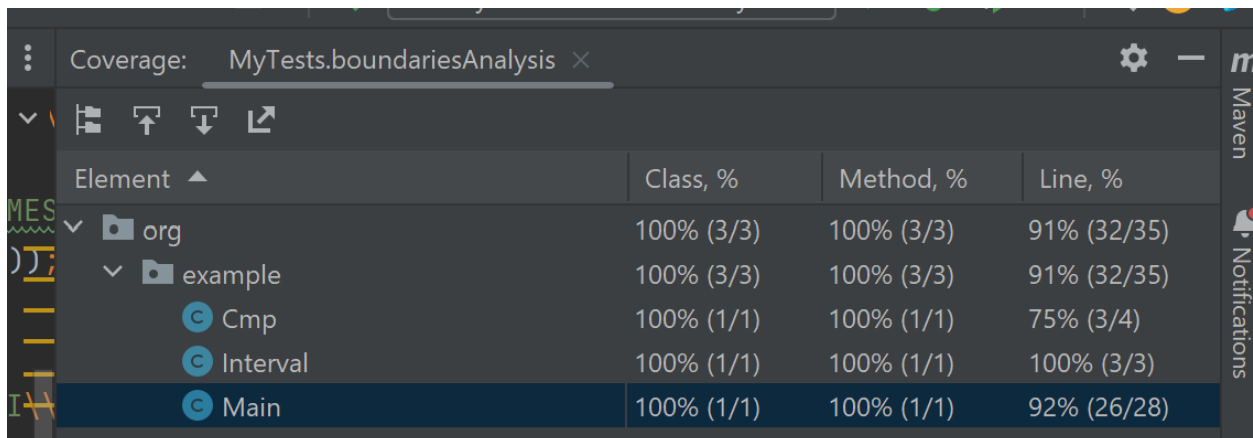
a. Pentru setul de date rezultat prin equivalence partitioning



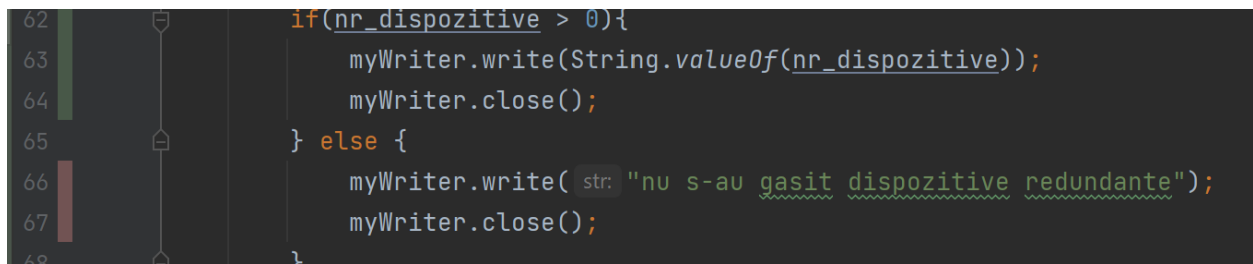
Element	Class, %	Method, %	Line, %
org	100% (3/3)	100% (3/3)	97% (34/35)
example	100% (3/3)	100% (3/3)	97% (34/35)
Cmp	100% (1/1)	100% (1/1)	75% (3/4)
Interval	100% (1/1)	100% (1/1)	100% (3/3)
Main	100% (1/1)	100% (1/1)	100% (28/28)

Se observă că acoperirea la nivel de linie este 100%, astfel că au fost acoperite toate posibilele comportamente ale programului.

b. Pentru setul de date rezultat prin boundary value analysis



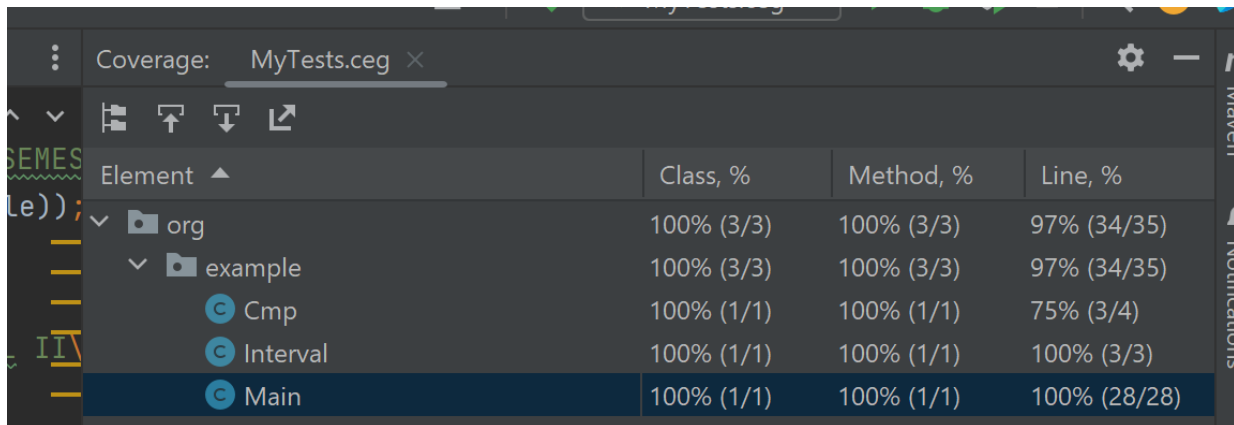
Element	Class, %	Method, %	Line, %
org	100% (3/3)	100% (3/3)	91% (32/35)
example	100% (3/3)	100% (3/3)	91% (32/35)
Cmp	100% (1/1)	100% (1/1)	75% (3/4)
Interval	100% (1/1)	100% (1/1)	100% (3/3)
Main	100% (1/1)	100% (1/1)	92% (26/28)



```
62 if(nr_dispozitive > 0){
63     myWriter.write(String.valueOf(nr_dispozitive));
64     myWriter.close();
65 } else {
66     myWriter.write(str: "nu s-au gasit dispozitive redundante");
67     myWriter.close();
68 }
```


Se observă că acoperirea la nivel de linie este 92%, ceea ce înseamnă că a fost omis cazul în care nu s-au găsit dispozitive redundante. Acest lucru se datorează setului de date de intrare care nu includea o mulțime de intervale eligibilă pentru acest caz.

c. Pentru setul de date rezultat prin cause-effect graphing



Element	Class, %	Method, %	Line, %
org	100% (3/3)	100% (3/3)	97% (34/35)
example	100% (3/3)	100% (3/3)	97% (34/35)
Cmp	100% (1/1)	100% (1/1)	75% (3/4)
Interval	100% (1/1)	100% (1/1)	100% (3/3)
Main	100% (1/1)	100% (1/1)	100% (28/28)

Se observă o acoperire de 100% la nivelul liniei, lucru așteptat deoarece au fost analizate toate cauzele și efectele programului.

3. Testare structurală

o Transformarea programului într-un graf orientat

	<pre> public static void gaseste_dispozitive_redundante(File f) throws IOException{ Scanner myScanner = new Scanner(f); FileWriter myWriter = new FileWriter("src/main/resources/granitaOut.txt"); int n, i, nr_dispozitive = 0, k; n = myScanner.nextInt(); if(n < 1 n > 6){ myWriter.write("n este invalid"); myWriter.close(); return; } k = myScanner.nextInt(); if(k < 1 k > 20){ myWriter.write("k este invalid"); myWriter.close(); return; } </pre>
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	

```

12      List<Interval> intervals = new ArrayList<>();
13
14      for(i = 0; i < n; i++){
15          int a = myScanner.nextInt();
16          int b = myScanner.nextInt();
17          Interval interval = new Interval(a, b);
18          intervals.add(interval);
19
20      Collections.sort(intervals, new Cmp());
21
22      int max_crt = -1;
23
24      for(i = 0; i < n; i++){
25          if(intervals.get(i).b > max_crt && intervals.get(i).b
26 - intervals.get(i).a <= k){
27              max_crt = intervals.get(i).b;
28          }
29          else {
30              nr_dispozitive ++;
31          }
32      }
33
34      if(nr_dispozitive > 0){
35          myWriter.write(String.valueOf(nr_dispozitive));
36          myWriter.close();
37      } else {
38          myWriter.write("nu      s-au      gasit      dispozitive
39 redundante");
40          myWriter.close();
41      }
42  }

```

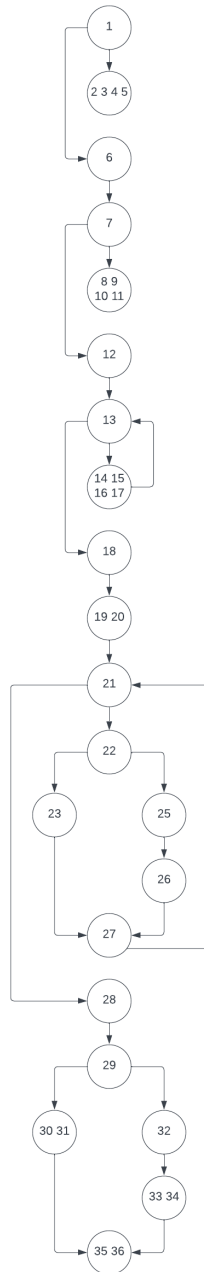


Fig. 2 Graful asociat programului

○ **Modified condition/decision coverage (MC/DC)**

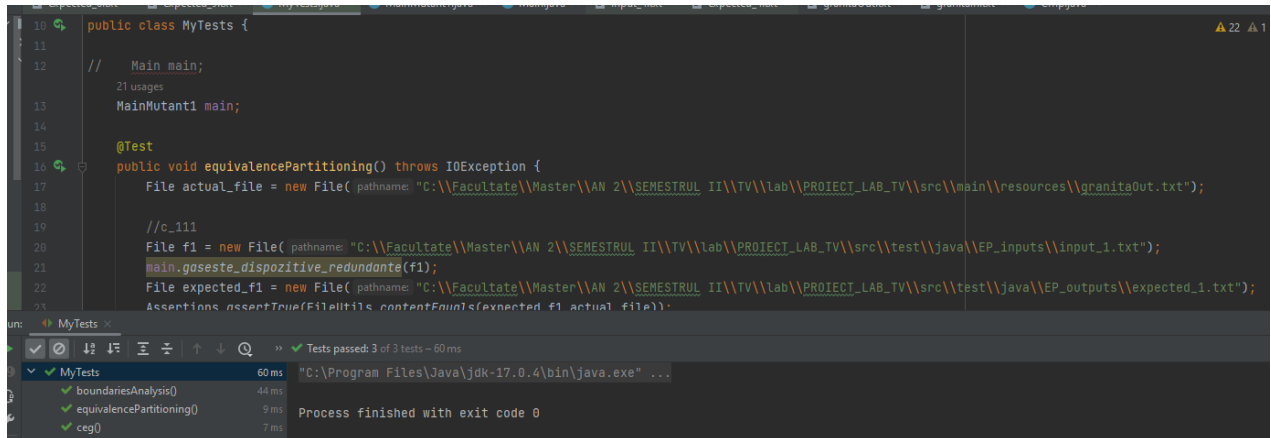
	Decizii	Conditii individuale
1	if(n < 1 n > 6)	n < 1, n > 6
2	if(k < 1 k > 20)	k < 1, k > 20
3	for(i = 0; i < n; i++)	i < n
4	for(i = 0; i < n; i++)	i < n
5	if(intervals.get(i).b > max_crt && intervals.get(i).b - intervals.get(i).a <= k)	intervals.get(i).b > max_crt, intervals.get(i).b - intervals.get(i).a <= k
6	(else 5) if(intervals.get(i).b <= max_crt intervals.get(i).b - intervals.get(i).a > k)	intervals.get(i).b <= max_crt, intervals.get(i).b - intervals.get(i).a > k
7	if(nr_dispozitive > 0)	nr_dispozitive > 0
8	(else 7) if(nr_dispozitive <= 0)	nr_dispozitive <= 0

Test	Intrări			Valoare de adevăr condiții	Decizie acoperita	Valoare de adevăr decizie	Rezultat
	n	k	S				
T1	0			True, false	1	True	n este invalid
T2	7			False, true	1	True	n este invalid
T3	2	8	(0, 1), (2, 3)	False, false	1	False	Nu s-au gasit dispozitive redundante
T4	2	0		True, false	2	True	k este invalid
T5	2	24		False, true	2	True	k este invalid
T6	2	10	(0, 1), (2, 3)	False, false	2	False	Nu s-au gasit dispozitive redundante
T7	2	10	(0, 3), (4, 7)	True, true	5	True	Nu s-au gasit dispozitive redundante
T8	2	10	(0, 10), (2, 3)	False, true	5	False	1
T9	2	3	(0, 6), (7, 20)	True, false	5	False	2
T10=T8	2	10	(0, 10), (2, 3)	True, false	6	True	1
T11=T9	2	3	(0, 6), (7, 20)	False, true	6	True	Nu s-au gasit dispozitive redundante
T12=T8	2	10	(0, 10), (2, 3)	False, false	6	False	1
T13=T8	2	10	(0, 10), (2, 3)	True	7	True	1
T14=T7	2	10	(0, 3), (4, 7)	False	7	False	Nu s-au gasit dispozitive redundante
T15=T7	2	10	(0, 3), (4, 7)	True	8	True	Nu s-au gasit dispozitive redundante
T16=T8	2	10	(0, 10), (2, 3)	False	8	False	1

4. Generare mutant de ordinul 1 echivalent al programului

Mutantul de ordin 1 al programului s-a obținut prin modificarea valorii variabilei `max_crt` din -1 în -10. Mutantul este echivalent, deoarece nu există niciun alt test care să îl omoare.

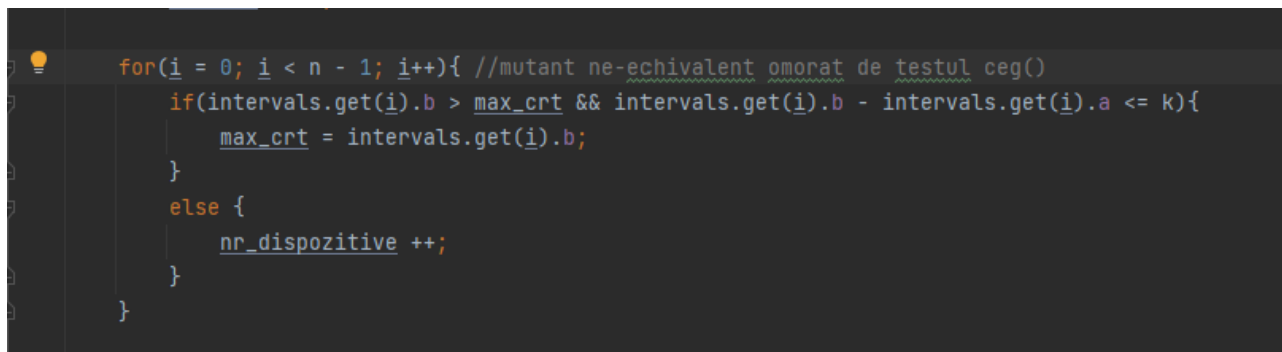
```
int max_crt = -10; // mutant de ordin 1 echivalent, in program max_crt = -1
```



5. Generare mutanți ne-echivalenți

a. Mutant ne-echivalent care să fie omorât de către test

Mutantul a fost generat prin schimbarea condiției de oprire din repetiția care parcurge mulțimea de intervale. Acest mutant a fost omorât de testul `ceg()`, de către setul de date numărul 4: {2, 4, (0, 3), (1, 2)}. Programul dă output-ul „1”, în timp ce mutantul dă output-ul „nu s-au găsit dispozitive redundante”, deoarece mulțimea de intervale nu este parcursă în întregime.



b. Mutant ne-echivalent care să nu fie omorât de către test

Mutantul a fost generat prin schimbarea operatorului `<=` în `<`, nefiind omorât de testul `ceg()`. Mutantul este ne-echivalent, deoarece există un set de date pentru care este omorât.

```

for(i = 0; i < n; i++){
    if(intervals.get(i).b > max_crt && intervals.get(i).b - intervals.get(i).a < k){ //mutant ne-echivalent neomorat de testul ceg()
        max_crt = intervals.get(i).b;
    }
    else {
        nr_dispozitive ++;
    }
}

```

Pentru setul de date:

6
 1
 0 10
 2 9
 3 8
 1 15
 6 11
 1 2

Output-ul aşteptat este 5, însă mutantul generat produce output-ul 6.