

UNIVERSITATEA DIN BUCUREȘTI
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
MASTER INGINERIE SOFTWARE

Testare și verificare

Proiect Individual

Alexandru Ilie
506

Specificațiile problemei:

Să se implementeze un program care primește de la tastatură un număr n , $1 < n < 50$, urmat de n numere naturale și un număr $i < n$. Să se întoarcă adevarat daca cel puțin i din cele n numere sunt prime altfel fals.

Input:

- n - numărul de elemente ale vectorului
- n numere naturale
- i - numărul de numere prime dorit

Output:

- s – adevarat daca cel puțin i din cele n numere sunt prime altfel fals.

Testarea funcțională

1. Partiționare de echivalență

Domeniul de intrări:

- $1 < n < 50$ - lungimea vectorului
- un vector a de n numere naturale
- un număr întreg i

Pentru fiecare intrare, distingem următoarele **clase de echivalență**:

- Pentru n :
 - $N_1 = \{1, 2, \dots, 50\}$
 - $N_2 = \{n \mid n < 1\}$
 - $N_3 = \{n \mid n > 50\}$
- Pentru a :
 - $A_1 = \{a \mid a \text{ conține numai valori pozitive}\}$
 - $A_2 = \{a \mid a \text{ conține măcar o valoare negativă}\}$
- Pentru i :
 - $I_1 = \{i \mid 1 \leq i \leq n\}$
 - $I_2 = \{i \mid i < 1\}$
 - $I_3 = \{i \mid i > n\}$

Domeniul de ieșiri consta din doua răspunsuri:

- Adevarat daca vectorul are cel puțin i numere prime
- Fals daca nu are cel puțin i numere prime

Rezulta ca domeniul de intrare este impartit in 2 clase de echivalenta:

$I_{11}(a) = \{ i \mid \text{exista cel putin } i \text{ nr prime in } a \}$
 $I_{12}(a) = \{ i \mid \text{nu exista cel putin } i \text{ nr prime in } a \}$

Astfel, obținem următoarele clase de echivalență globale:

- $C_{1111} = \{ (n, a, i) \mid n \text{ in } N_1, a \text{ in } A_1, i \text{ in } I_1, i \text{ in } I_{11}(a) \}$
- $C_{1112} = \{ (n, a, i) \mid n \text{ in } N_1, a \text{ in } A_1, i \text{ in } I_1, i \text{ in } I_{12}(a) \}$
- $C_{112} = \{ (n, a, i) \mid n \text{ in } N_1, a \text{ in } A_1, i \text{ in } I_2 \}$
- $C_{113} = \{ (n, a, i) \mid n \text{ in } N_1, a \text{ in } A_1, i \text{ in } I_3 \}$
- $C_{12} = \{ (n, a, i) \mid n \text{ in } N_1, a \text{ in } A_2 \}$
- $C_2 = \{ (n, a, i) \mid n \text{ in } N_2 \}$
- $C_3 = \{ (n, a, i) \mid n \text{ in } N_3 \}$

În total avem 6 clase de echivalență. Alegem următoarele date de test:

Date intrare	Răspuns
$T_{1111} = (4, \{2, 3, 4, 5\}, 3)$	true
$T_{1111} = (4, \{2, 3, 4, 5\}, 4)$	false
$T_{112} = (4, \{2, 3, 4, 5\}, -3)$	Conditions not met
$T_{113} = (4, \{2, 3, 4, 5\}, 5)$	Conditions not met
$T_{12} = (4, \{-2, 3, 4, 5\}, 1)$	Conditions not met
$T_2 = (0, _, _)$	Conditions not met
$T_3 = (51, _, _)$	Conditions not met

2. Analiza valorilor de frontieră

Valorile de frontieră sunt:

- Dimensiunea vectorului: $n = 0, 1, 50, 51$
- $i \in \{1, n\}$

Distingem următoarele clase din punctul de vedere al valorilor de frontieră:

- $N_1 = \{ (n, a, i) \mid n = 1 \text{ sau } n = 50 \}$
- $N_2 = \{ (n, a, i) \mid n = 0 \}$
- $N_3 = \{ (n, a, i) \mid n = 51 \}$
- $I_1 = \{ (n, a, i) \mid i = 1 \}$
- $I_2 = \{ (n, a, i) \mid i = n \}$

Alegem următoarele valori de test. In total există $3 * 2 = 6$ clase dar unele alegeri sunt echivalente.

Date intrare	Răspuns
$T11 = \{ (1, \{3\}, 1), (50, \{3, \dots, 3\}, 1) \}$	true
$T121 = \{ (1, \{2\}, 50) \}$	false
$T121 = \{ (50, \{2, \dots, 2\}, 50) \}$	true
$T21 = \{ (0, \{\}, 1) \}$	Conditions not met.
$T22 = \{ (0, \{\}, 0) \}$	Conditions not met.
$T31 = \{ (51, \{2, \dots, 2\}, 1) \}$	Conditions not met.
$T32 = \{ (51, \{2, \dots, 2\}, 51) \}$	Conditions not met.

3. Partiționarea în categorii

Avem următoarea partiționare în unități a problemei:

```
public static boolean isPrime(int x)
public static int solve(int[] a, int n, int i)
```

isPrime

- Categorii: număr prim sau nu.
- Întoarce **true** dacă numărul dat este prim, altfel **false**.

Date de intrare:

T1 = 4 => false

T2 = 3 => true

solve

- Categorii:
 - n: dacă se află în intervalul valid 1... 50 sau nu
 - array: dacă conține elemente negative sau nu
 - i: {1 ... n}
- Alternative:
 - $n < 0$, $n = 0$, $n = 1$, $n = 2 \dots 49$, $n = 50$, $n > 50$
 - a: conține numai numere naturale sau conține cel puțin o valoare negativă
 - i:
 - $i \Rightarrow 1 \ \&\& \ i \leq n$;
 - $i < 1$
 - $i > n$

Observație: Avem $6 * 2 * 3 = 36$ cazuri de testare, însă putem reduce foarte mult din acestea, prin adăugarea de constrângeri. Rezultă următoarele teste:

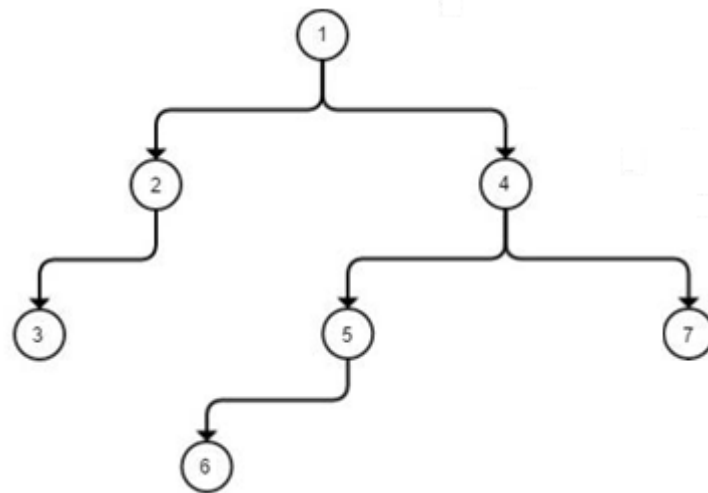
Date de intrare	Răspuns
T1 = (-1, _, _)	Conditions not met.
T2 = (0, {}, _)	Conditions not met.
T311 = (1, {2}, 1)	true
T312 = (1, {2}, 0)	Conditions not met.
T313 = (1, {2}, 5)	Conditions not met.
T32 = (1, {-1}, 1)	Conditions not met.
T411 = (2, {2, 3}, 2)	true
T412 = (2, {2, 2}, 0)	Conditions not met.
T413 = (2, {2, 2}, 3)	Conditions not met.
T42 = (2, {3, -14}, 1)	Conditions not met.
T511 = (50, {2, ..., 2}, 4)	true
T512 = (50, {2, ..., 2}, 0)	Conditions not met.
T513 = (50, {2, ..., 2}, 51)	Conditions not met.
T52 = (50, {-2, ..., -2}, 4)	Conditions not met.
T6 = (51, _, _)	Conditions not met.

Testarea structurală

Programul în Java:

#	
	<pre>public class Primes { public static int solve(int [] a, int n, int i) { 1 if (n < 1 n > 50 i < 0 i > n) { 2 System.out.println("Conditions not met"); 3 return false; } 4 if (Arrays.stream(a).anyMatch(x -> x < 0)) { 5 System.out.println("Conditions not met."); 6 return false; } 7 return Arrays.stream(a).filter(x -> isPrime(x)).count() >= i; } }</pre>

Graful programului este:



Statement Coverage

Pentru a realiza acoperirea la nivel de instrucțiune dăm următoarele teste:

Date intrare	Răspuns
T_1 = (0, {}, _)	Conditions not met.
T_2 = (1, {-4}, _)	Conditions not met.
T_3 = (1, {2}, 1)	true

Branch coverage

Instrucțiuni care duc la ramuri în program:

```
if ( n < 1 || n > 50 || i < 0 || i > n )  
if (Arrays.stream(a).anyMatch(x -> x < 0))
```

Pentru a testa acoperirea la nivel de ramuri, avem următoarele teste:

Date intrare	Răspuns
T_1 = (0, {}, _)	Conditions not met.
T_2 = (1, {9}, 1)	false
T_3 = (1, {-9}, 1)	Conditions not met.
T_4 = (1, {9}, 1)	false

Condition coverage

Deciziile din programul Java:

Decizii	Condiții individuale
<code>if (n < 1 n > 50 i < 0 i > n)</code>	<code>n < 1</code> <code>n > 50</code> <code>i < 0</code> <code>1 > n</code>
<code>if (Arrays.stream(a).anyMatch(x -> x < 0))</code>	<code>a[i] < 0</code>

Pentru a acoperi toate condițiile din setul de mai sus, folosim următoarea suită de teste:

Date intrare	Răspuns
<code>T_1 = (0, {}, _)</code>	Conditions not met.
<code>T_2 = (51, {2, ..., 2}, _)</code>	Conditions not met.
<code>T_3 = (1, {2}, 0)</code>	Conditions not met.
<code>T_4 = (1, {2}, 2)</code>	Conditions not met.
<code>T_5 = (1, {-1}, 1)</code>	Conditions not met.
<code>T_6 = (1, {2}, 1)</code>	true

Complexitatea programului

Formula lui **McCabe** pentru complexitate ciclomatică: Dat fiind un graf complet conectat G cu n noduri, atunci numărul de circuite linear independente este dat de:
 $V(G) = e - n + 1$, unde:

G - graf complet conectat (există o cale între oricare două noduri)

Circuit - cale care începe și se termină în același nod

Circuite liniar independente - niciunul nu poate fi obținut ca o combinație a celorlalte.

Adăugăm următoarele noduri în graful de mai sus pentru a deveni complet

conectat: (3, 1), (6, 1) și (7, 1)

Atunci: $V(G) = 9 - 7 + 1 = 3$.

Circuite independente:

- $1 \rightarrow 2 \rightarrow 3 \rightarrow 1$
- $1 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 1$
- $1 \rightarrow 4 \rightarrow 7 \rightarrow 1$

Acoperirea la nivel de cale

Putem descrie programul ca o expresie regulată folosind nodurile grafului, conform Paige și Holthouse.

Pentru graful de mai sus, avem expresie regulată:

$$1.(2.3 + 4.(5.6+7))$$

Numărul de căi: 3

Căile posibile sunt:

1.2.3

1.4.5.6

1.4.7

Date de test:

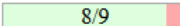
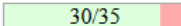
Date intrare	Răspuns
T_1 = (0, {}, _)	Conditions not met.
T_2 = (1, {-1}, _)	Conditions not met.
T_3 = (1, {2}, 1)	true

Generator de mutanți

Pentru generarea mutanților s-a folosit [Pitclipse](#)

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage
1	89%  8/9	86%  30/35

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage
P	1	89%  8/9	86%  30/35

Report generated by [PIT](#) 1.1.5

Observăm că testele de mai sus omoară 30 din 35 de mutanți.

Supraviețuiesc urmatorii:

1. ChangedConditionalBoundary

```
return n == 2 || !(n < 2 || n % 2 == 0) ...
```

devine

```
return n == 2 || !(n <= 2 || n % 2 == 0) ...
```

Nu poate fi omorât pentru că se testează anterior egalitatea lui n cu 2, deci mutația operatorului nu schimbă rezultatul.

2. Replace Integer Modulus With Multiplication

```
.filter(x -> x % 2 != 0)  
devine  
.filter(x -> x * 2 != 0)
```

Nu poate fi omorat pentru ca eliminarea numerelor pare este doar o optimizare a algoritmului, rezultatul nu este afectat daca acestea nu sunt eliminate. In plus mutatia elimina dublul fiecarui numar par dar nici acest lucru nu afecteaza integritatea algoritmului.

3. ChangedConditionalBoundary

```
Arrays.stream(a).anyMatch(x -> x < 0)  
devine  
Arrays.stream(a).anyMatch(x -> x <= 0)
```

Poate fi omorat adaugand testul:

```
@Test  
public void killChangedConditionalBoundary() {  
    assertTrue( solve( 50, IntStream.rangeClosed(1, 50).map(x-> x != 2 ? 0 : x ).toArray(), 1) );  
}
```