

# Testare și verificare

- Proiect individual -

## Specificațiile problemei:

Să se implementeze un program care primește de la tastatură un număr  $n$ ,  $1 < n < 100$ , urmat de  $n$  numere **întregi pozitive** și doi indici  $low$  și  $high$ ,  $low < high$ . Să se întoarcă suma numerelor pătrate perfecte dintre cei doi indici.

### Input:

- $n$  - numărul de elemente ale vectorului
- $n$  numere naturale
- $low$  - primul indice
- $high$  - cel de-al doilea indice

**Output:** un număr  $s$ , reprezentând suma numerelor pătrate perfecte din vector.

## Testarea funcțională

### 1. Partiționare de echivalență

Domeniul de intrări:

- $1 < n < 100$  - lungimea vectorului
- un vector de numere **întregi pozitive**
- un număr întreg (indice)  $low$
- un număr întreg (indice)  $high$

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

- Pentru  $n$ :
  - $N_1 = 1 \dots 100$
  - $N_2 = \{ n / n < 1 \}$
  - $N_3 = \{ n / n > 100 \}$
- Pentru  $a$ :
  - $A_1 = \{ a / a \text{ conține numai valori pozitive} \}$
  - $A_2 = \{ a / a \text{ conține măcar o valoare negativă} \}$
- Pentru  $low$ :
  - $L_1 = \{ low / 0 \leq low < n \}$

- $L\_2 = \{low / low < 0\}$
- $L\_3 = \{low / low \geq n\}$
- Pentru *high*, analog *low*:
  - $H\_1 = \{high / 0 \leq high < n\}$
  - $H\_2 = \{high / high < 0\}$
  - $H\_3 = \{high / high \geq n\}$

În ceea ce privește domeniul de ieșiri, avem un singur răspuns posibil și anume  $s$  - suma pătratelor perfecte dintre cei doi indici din vector. Aceasta va fi 0 dacă vectorul nu conține niciun element pătrat perfect între *low* și *high*.

Astfel, obținem următoarele clase de echivalență globale (pentru întregul program):

- $C\_1111 = \{(n, a, high, low) / n \text{ in } N\_1, a \text{ in } A\_1, low \text{ in } L\_1, high \text{ in } H\_1\}$
- $C\_1112 = \{(n, a, high, low) / n \text{ in } N\_1, a \text{ in } A\_1, low \text{ in } L\_1, high \text{ in } H\_2\}$
- $C\_1113 = \{(n, a, high, low) / n \text{ in } N\_1, a \text{ in } A\_1, low \text{ in } L\_1, high \text{ in } H\_3\}$
- ~~$C\_1122 = \{(n, a, high, low) / n \text{ in } N\_1, a \text{ in } A\_1, low \text{ in } L\_2, high \text{ in } H\_2\}$~~
- ~~$C\_1123 = \{(n, a, high, low) / n \text{ in } N\_1, a \text{ in } A\_1, low \text{ in } L\_2, high \text{ in } H\_3\}$~~
- ~~$C\_1133 = \{(n, a, high, low) / n \text{ in } N\_1, a \text{ in } A\_1, low \text{ in } L\_3, high \text{ in } H\_3\}$~~
- $C\_12 = \{(n, a, high, low) / n \text{ in } N\_1, a \text{ in } A\_2\}$
- $C\_2 = \{(n, a, high, low) / n \text{ in } N\_2\}$
- $C\_3 = \{(n, a, high, low) / n \text{ in } N\_3\}$

În total, 8 clase. Alegem următoarele date de test:

Date intrare	Răspuns
$T\_1111 = \{5, \{3, 4, 5, 8, 9\}, 1, 4\}$	13
$T\_1112 = \{5, \{3, 4, 5, 8, 9\}, 3, -1\}$	Conditions not met.
$T\_1113 = \{5, \{3, 4, 5, 8, 9\}, 2, 101\}$	Conditions not met.
<del><math>T\_1122 = \{5, \{3, 4, 5, 8, 9\}, -1, -1\}</math></del>	Conditions not met.
<del><math>T\_1123 = \{5, \{3, 4, 5, 8, 9\}, -2, 101\}</math></del>	<del>Conditions not met.</del>
<del><math>T\_1133 = \{5, \{3, 4, 5, 8, 9\}, 101, 101\}</math></del>	Conditions not met.
$T\_12 = \{5, \{-1, 2, 3, -5, 0\}, \_, \_ \}$	Conditions not met.
$T\_2 = \{0, \_, \_, \_ \}$	Conditions not met.
$T\_3 = \{101, \_, \_, \_ \}$	Conditions not met.

## 2. Analiza valorilor de frontieră

Valorile de frontieră sunt:

- Dimensiunea array-ului:  $n = 0, 1, 100, 101$
- $\text{Low} \in \{-1, 0, n - 1, n\}$
- $\text{High} \in \{-1, 0, n - 1, n\}$

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

- $N\_1 = \{ (n, a, \text{high}, \text{low}) / n = 1 \text{ or } n = 100 \}$
- $N\_2 = \{ (n, a, \text{high}, \text{low}) / n = 0 \}$
- $N\_3 = \{ (n, a, \text{high}, \text{low}) / n = 101 \}$
- $L\_1 = \{ (n, a, \text{high}, \text{low}) / \text{low} = 0, n-1 \}$
- $L\_2 = \{ (n, a, \text{high}, \text{low}) / \text{low} = -1 \}$
- $L\_3 = \{ (n, a, \text{high}, \text{low}) / \text{low} = n-1 \}$
- ~~$L\_4 = \{ (n, a, \text{high}, \text{low}) / \text{low} = n \}$~~
- $H\_1 = \{ (n, a, \text{high}, \text{low}) / \text{high} = 0, n-1 \}$
- $H\_2 = \{ (n, a, \text{high}, \text{low}) / \text{high} = -1 \}$
- $H\_3 = \{ (n, a, \text{high}, \text{low}) / \text{high} = n-1 \}$
- ~~$H\_4 = \{ (n, a, \text{high}, \text{low}) / \text{high} = n \}$~~

Alegem următoarele valori de test. Menționăm că deși există în total  $3 * 4 * 4 = 48$  date de test, unele alegeri sunt echivalente.

Date intrare	Răspuns
$T\_11h = \{(n, \{3\}, -1, h) / n \in \{1, 100\}, h \in H\_h\}$	Conditions not met.
$T\_121 = \{(1, \{4\}, 0, -1), (100, \{1, \dots, 1\}, 0, -1)\}$	Conditions not met.
$T\_122 = \{(1, \{4\}, 0, 0), (100, \{4, \dots, 4\}, 0, 0)\}$	4
$T\_123 = \{(100, \{1, \dots, 1\}, 0, 99), (1, \{1\}, 0, 0)\}$	100, respectiv 1
$T\_124 = \{(\_, \_, \_, n)\}$	Conditions not met.
$T\_133 = \{(1, \{4\}, n-1, n-1)\}$	4
$T\_14 = \{(\_, \_, n, \_)\}$	Conditions not met.
$T\_2 = \{0, \{ \}, 0, 0\}$	Conditions not met.
$T\_3 = \{101, \{1, \dots, 1\}, \_, \_ \}$	Conditions not met.

### 3. Partiționarea în categorii

Avem următoarea partiționare în unități a problemei:

```
public static boolean isPerfectSquare(int x)
public static int solve(int n, int[] a, int low, int high)
```

**isPerfectSquare**

- Categorii: număr pătrat perfect sau nu.
- Întoarce **true** dacă numărul dat este pătrat perfect și **false** altfel.

Date de intrare:

T\_1 = 4 => true

T\_2 = 3 => false

**solve**

- Categorii:
  - n: dacă se află în intervalul valid 1.. 100 sau nu
  - array: dacă conține elemente negative sau nu
  - low: {0 .. n - 1}
  - high: {0 .. n - 1}
- Alternative:
  - $n < 0$ ,  $n = 0$ ,  $n = 1$ ,  $n = 2 \dots 100$ ,  $n = 100$ ,  $n > 101$
  - array: conține numai numere naturale sau conține cel puțin o valoare negativă
  - low:
    - $low \geq 0 \ \&\& \ low < n$ ;
    - $low < 0$
    - $low > n$
    - $low = n - 1$
  - high: analog low

**Observație:** Avem  $6 * 2 * 5 * 5 = 300$  cazuri de testare, însă putem reduce foarte mult din acestea, prin adăugarea de constrângeri. Rezultă următoarele teste:

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

T_311 = (1, {4}, 0, 0)	4
T_312 = (1, {4}, -1, _)	Conditions not met.
T_313 = (1, {4}, 3, _)	Conditions not met.
T_3142 = (1, {4}, 0, -1)	Conditions not met.
T_4111 = (5, {3, 4, 5, 8, 9}, 3, 4)	9
T_4122 = (5, {3, 4, 5, 8, 9}, 3, [-1   5   6])	Conditions not met.
T_5111 = (100, {1, ..., 1}, 0, 99)	100
T_61 = (101, _, _, _)	Conditions not met.
T_4113 = (78, {1, ..., 1}, 0, 101)	Conditions not met.
T_5144 = (100, {1, ..., 1}, 99, 99)	1
T_32 = (1, {-1}, _, _)	Conditions not met.
T_42 = (5, {3, 4, -5, -8, 9}, _, _)	Conditions not met.
T_52 = (100, {-1, ..., -1}, _, _)	Conditions not met.

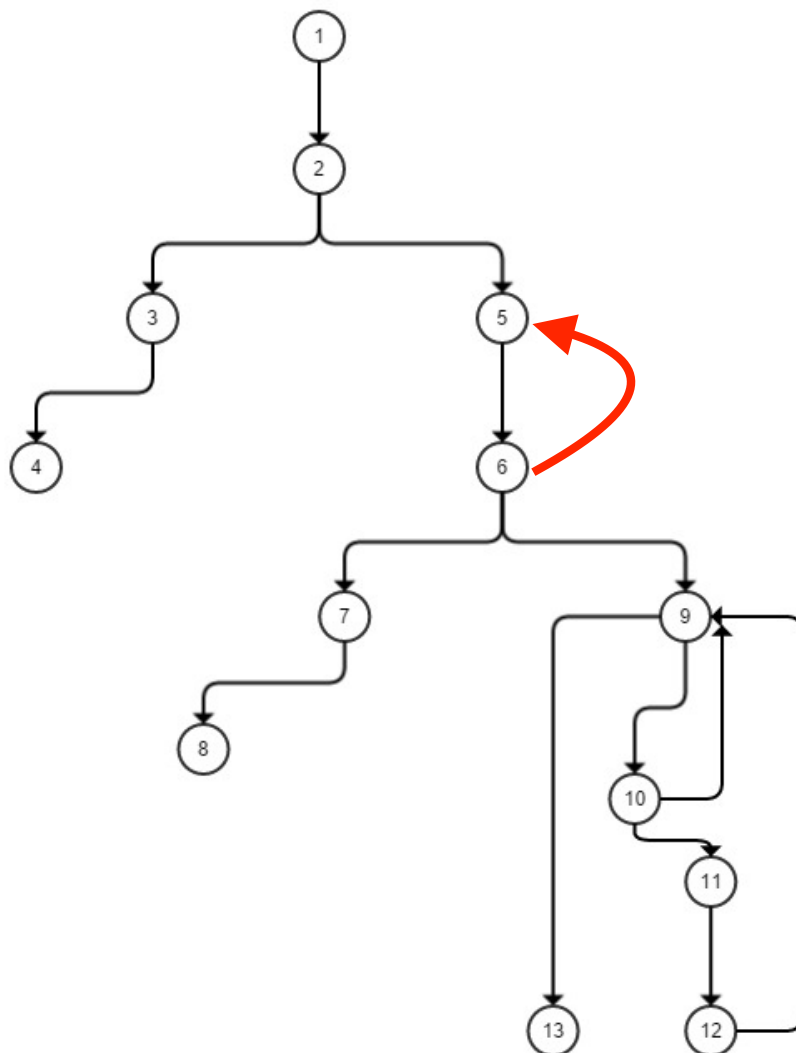
## Testarea structurală

Dăm, mai jos, programul în Java:

#	Program
1	<code>public class PerfectSquares {</code>
2	<code>    public static int solve(int n, int[] a, int low, int high) {</code>
3	<code>        int s = 0;</code>
4	<code>        if(low &lt; 0    low &gt;= n    high &lt; 0    high &gt;= n    n &lt; 1</code>
5	<code>   n &gt; 100) {</code>
6	<code>            System.out.println("Conditions not met.");</code>
7	<code>            return -1;</code>
8	<code>        }</code>
9	<code>        for(int i = 0; i &lt; n; ++i) {</code>
10	<code>            if(a[i] &lt; 0) {</code>
11	<code>                System.out.println("Conditions not met.");</code>
12	<code>                return -1;</code>
13	<code>            }</code>
14	<code>        }</code>
15	<code>    }</code>
16	<code>}</code>

	<pre>         }     }     9   for(int i = low; i &lt;= high; ++i) {     10       if(isPerfectSquare(a[i])) {     11           System.out.println(a[i]);     12           s += a[i];         }     }     13   return s;     }     }</pre>
--	--

Graful programului este:



### Statement Coverage

Pentru a realiza acoperirea la nivel de instrucțiune, ne concentrăm asupra nodurilor. Dăm următoarele teste:

Date intrare	Răspuns
T_1 = (0, {}, 0, 0)	Conditions not met.
T_2 = (1, {4}, -1, 0)	Conditions not met.
T_3 = (1, {4}, 0, 5)	Conditions not met.
T_4 = (6, {3, 4, -5, -4, 8, 9}, 0, 4)	Conditions not met.
T_5 = (5, {3, 4, 5, 8, 9}, 0, 4)	13
T_6 = (5, {3, 4, 5, 8, 9}, 0, 0)	0

### Branch coverage

Instrucțiuni care duc la ramuri în program:

<code>if(low &lt; 0    low &gt;= n    high &lt; 0    high &gt;= n    n &lt; 1    n &gt; 100)</code>
<code>if(a[i] &lt; 0)</code>
<code>if(isPerfectSquare(a[i]))</code>

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

Date intrare	Răspuns
T_1 = (0, {}, 0, 0)	Conditions not met.
T_2 = (5, {1, 3, 5, 4, 9}, 6, 3)	Conditions not met.
T_3 = (5, {1, 3, 5, 4, 9}, 2, -2)	Conditions not met.
T_4 = (5, {1, 3, 5, 4, 9}, 2, 10)	Conditions not met.
T_5 = (5, {1, -1, -4, 9, 3}, 2, 4)	Conditions not met.
T_6 = (1, {4}, 0, 0)	4

T_7 = (1, {3}, 0, 0)	0
T_8 = (5, {1, 3, 4, 5, 9}, 0, 1)	1

### Condition coverage

Deciziile din programul Java:

Decizii	Condiții individuale
<code>if( n &lt; 1    n &gt; 100    low &lt; 0    low &gt;= n    high &lt; 0    high &gt;= n)</code>	n < 1 n > 100 low < 0 low >= n high < 0 high >= n
<code>for(int i = 0; i &lt; n; ++i)</code>	i < n
<code>if(a[i] &lt; 0)</code>	a[i] < 0
<code>for(int i = low; i &lt;= high; ++i)</code>	i >= low i <= high
<code>if(isPerfectSquare(a[i]))</code>	<code>isPerfectSquare(a[i]) == true</code>

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

Date intrare	Răspuns
T_1 = (0, {}, 0, 0)	Conditions not met.
T_2 = (101, {}, 0, 1)	Conditions not met.
T_3 = (3, {1, 2, 3}, -5, 0)	Conditions not met.
T_4 = (3, {1, 2, 3}, 10, 2)	Conditions not met.
T_5 = (3, {1, 2, 3}, 2, -5)	Conditions not met.
T_6 = (3, {1, 2, 3}, 2, 10)	Conditions not met.
T_7 = (1, {-1}, 0, 0)	Conditions not met.
T_8 = (1, {2}, 0, 0)	0
T_9 = (1, {4}, 0, 0)	4



$T_{10} = (3, \{1, 4, 9\}, 1, 2)$	13
-----------------------------------	----

## Complexitatea programului

Formula lui **McCabe** pentru complexitate ciclomatică: Dat fiind un graf complet conectat  $G$  cu  $e$  arce și  $n$  noduri, atunci numărul de circuite linear independente este dat de:

$$V(G) = e - n + 1, \text{ unde:}$$

$G$  - 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:

$(4, 1)$ ,  $(8, 1)$  și  $(13, 1)$

Atunci:  $V(G) = 17 - 13 + 1 = 5$ .

Circuite independente:

- $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$
- $1 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 1$
- $1 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow 9 \rightarrow 13 \rightarrow 1$
- $9 \rightarrow 10 \rightarrow 9$
- $9 \rightarrow 10 \rightarrow 11 \rightarrow 12 \rightarrow 9$

## 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:

$$1.2.(3.4+5.6.(7.8+9.10.(9+11.12)*.13))$$

Numărul de căi: 4

Căile posibile sunt:

1.2.3.4

1.2.5.6.7.8

1.2.5.6.9.10.9.13

1.2.5.6.9.10.11.12.9.13

Date de test:

Date test	Răspuns
T_1 = (0, {}, 0, 0)	Conditions not met.
T_2 = (3, {-1, 1, 0}, 0, 1)	Conditions not met.
T_3 = (3, {2, 3, 5}, 0, 1)	0
T_4 = (3, {2, 3, 4}, 0, 2)	4

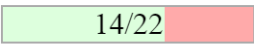
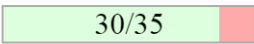
## Generator de mutanți

Pentru generarea mutanților s-a folosit plugin-ul PIT pentru gradle:

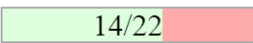
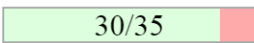
<http://gradle-pitest-plugin.solidsoft.info/>

# Pit Test Coverage Report

## Project Summary

Number of Classes	Line Coverage	Mutation Coverage
1	64% 	86% 

## Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage
<a href="#">perfectsquares</a>	1	64% 	86% 

---

Report generated by [PIT](#) 1.1.2

Observăm că unul dintre testele de mai sus omoară 30 din 35 de mutanți. Unul dintre mutanții neomorâți este un *ConditionalsBoundaryMutator* care este generat la linia (14) în cod:

```
if(a[i] < 0) {  
if(a[i] <= 0) {  
    System.out.println("Conditions not met.");
```

Pentru a omorî acest mutant, adăugăm un test care să conțină elemente nule în array:

```
int[] d = {0, 1, 2};  
assertEquals(0, PerfectSquares.solve(1, d, 0, 0));
```

Ceilalți mutanți pot fi omorâți foarte ușor prin adăugarea unui test care să testeze mesajele printate la Standard output, de exemplu folosind librăria:

<http://stefanbirkner.github.io/system-rules/>

