

Testare și verificare

- Proiect individual -

Specificațiile problemei:

Să se implementeze un program care primește de la tastatură o matrice A de numere întregi pozitive, dimensiunile ei $1 < n < 5$ și $1 < m < 5$ și un număr dat x.

Input:

- A - matricea de elemente
- n - lungimea matricei
- m - înălțimea matricei
- x - numărul dat

Output: un boolean, care reprezintă valoarea de adevăr a funcției.

Testarea funcțională

1. Partiționarea de echivalență

Domeniul de intrare:

- o matrice de numere întregi
- $1 < n < 10$ lungimea matricei
- $1 < m < 10$ înălțimea matricei
- un număr întreg

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

- Pentru A:
 - $A_1 = \{a \mid a \text{ conține numai valori pozitive}\}$
 - $A_2 = \{a \mid a \text{ conține măcar o valoare negativă}\}$
- Pentru n:
 - $N_1 = 1 \dots 5$
 - $N_2 = \{n \mid n < 1\}$
 - $N_3 = \{n \mid n > 5\}$
- Pentru m:
 - $M_1 = 1 \dots 5$
 - $M_2 = \{m \mid m < 1\}$
 - $M_3 = \{m \mid m > 5\}$
- Pentru x:

- $X_1 = \{x \mid x > 0\}$
- $X_2 = \{x \mid x < 0\}$

În ceea ce privește domeniul de ieșiri, avem un singur răspuns posibil și anume aceasta va fi False dacă suma elementelor matricei nu este egala cu un număr dat.

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

- $C_{1111} = \{(a, n, m, x) \mid a \text{ in } A_1, n \text{ in } N_1, m \text{ in } M_1, x \text{ in } X_1\}$
- $C_{1112} = \{(a, n, m, x) \mid a \text{ in } A_1, n \text{ in } N_1, m \text{ in } M_1, x \text{ in } X_2\}$
- $C_{122} = \{(a, n, m, x) \mid a \text{ in } A_1, n \text{ in } N_2, m \text{ in } M_2\}$
- $C_{133} = \{(a, n, m, x) \mid a \text{ in } A_1, n \text{ in } N_3, m \text{ in } M_3\}$
- $C_2 = \{(a, n, m, x) \mid a \text{ in } A_2\}$

În total, 4 clase. Alegem următoarele date de test:

Date intrare	Răspuns
$T_{1111} = \{\{1,1\}\{2,2\}, 2, 2, 6\}$	True
$T_{1112} = \{\{1,1\}\{2,2\}, 2, 2, 0\}$	Conditions not met.
$T_{1121} = \{\{1,1\}\{2,2\}, 2, -2, 6\}$	Conditions not met.
$T_{1211} = \{\{1,1\}\{2,2\}, -2, 2, 6\}$	Conditions not met.
$T_2 = \{\{-1,1\}\{2,2\}, 2, 2, 6\}$	Conditions not met.

2. Analiza valorilor de frontieră

Valorile de frontiera sunt:

- Dimensiunile matricei: $n = 0, 1, 5, 6$; $m = 0, 1, 5, 6$
- Numărul dat: $x = -1, 0$

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

- $X_1 = \{(a, n, m, x) \mid x = -1\}$
- $X_2 = \{(a, n, m, x) \mid x = 75\}$
- $M_1 = \{(a, n, m, x) \mid m = 1 \text{ or } m = 5\}$
- $M_2 = \{(a, n, m, x) \mid m = 2\}$
- $N_1 = \{(a, n, m, x) \mid n = 1 \text{ or } n = 5\}$
- $N_2 = \{(a, n, m, x) \mid n = 2\}$

Alegem următoarele valori de test. Menționăm că deși există în total $4 * 4 * 2 = 32$ date de test, unele alegeri sunt echivalente.

Date de intrare	Răspuns
$T_{111} = \{\{1,1,1,1,1\}, 1, 5, 1\}$	Conditions not met.
$T_{112} = \{\{1,1,1,1,1\}\{2,2,2,2,2\}, 2, 5, 15\}$	True
$T_{121} = \{\{1\}\{2\}\{3\}\{4\}\{5\}, 5, 1, 1\}$	Conditions not met.
$T_{121} = \{\{1,1\}\{2,2\}\{3,3\}\{4,4\}\{5,5\}, 5, 2, 30\}$	True
$T_{122} = \{\{1,1,1,1,1\}\{2,2,2,2,2\}\{3,3,3,3,3\}\{4,4,4,4,4\}\{5,5,5,5,5\}, 5, 5, -1\}$	Conditions not met.
$T_2 = \{\{1,1\}\{2,2\}\{3,3\}\{4,4\}\{5,5\}, 5, 5, 75\}$	True

3. Partiționarea în categorii

Avem următoarea partiționare în unități a problemei:

public static boolean checkInMatrix(**int** a[][], **int** n, **int** m, **int** x)

checkInMatrix

- Categorii: suma matricei este egala cu x sau nu.
- Întoarce *true* daca numărul este suma și *false* altfel.

Date de intrare:

T_1 = {{1,1}{2,2}, 2, 2, 6} => true

T_2 = {{1,1}{2,2}, 2, 2, 6} => false

Solve

- Categorii:
 - A: daca conține elemente negative sau nu
 - n: dacă se află în intervalul valid 1 .. 5 sau nu
 - m: dacă se află în intervalul valid 1 .. 5 sau nu
 - x: este egal cu suma sau nu
- Alternative:
 - A: conține numai numere naturale sau conține cel puțin o valoare negativă
 - n: $n < 0, n = 0, n = 1, n = 2 \dots 5, n = 5, n > 6$
 - m: $n < 0, n = 0, n = 1, n = 2 \dots 5, n = 5, n > 6$
 - x: este egal cu suma sau nu este egal cu suma

Observație: Avem $2 * 6 * 6 * 2 = 144$ cazuri de testare, însă putem reduce foarte mult din acestea, prin adăugarea de constrângeri. Rezultă următoarele teste:

Date de intrare	Răspuns
T_1 = {{1,1}{2,2}, 2, 2, 1}	True
T_111 = {{1,1}{2,2}, 2, 2, 5}	True
T_1112 = {{1,1}{2,2}, 2, 2, -2}	Conditions not met.

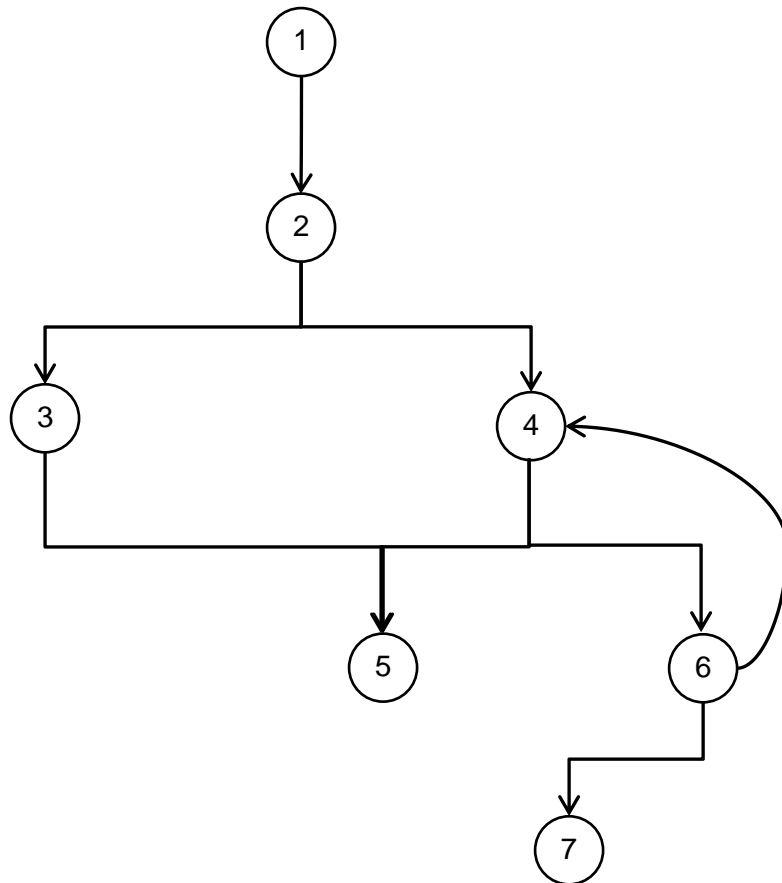
T_112 = {{1,1}{2,2}, 2, -2, 1}	Conditions not met.
T_12 = {{1,1}{2,2}, -2, 2, 1}	Conditions not met.
T_2 = {{-1,1}{2,2}, 2, 2, 5}	Conditions not met.
T_1211 = {{1,1}{2,2}, 2, 2, 6}	True
T_1111 = {{1,1}{2,2}, 2, 2, 5}	True

Testarea structurală

Dăm, mai jos, programul în Java:

#	Program
1	<code>public static boolean checkInMatrix(int a[][], int n,</code>
2	<code>int m, int x){</code>
3	<code> validateMatrix(a,n,m,x);</code>
	<code> int s = 0;</code>
	<code> for(int i=0;i<n;i++){</code>
	<code> for(int j=0;j<m;j++){</code>
	<code> s+=a[i][j];</code>
	<code> }</code>
	<code> }</code>
4	<code> if(s==x)</code>
5	<code> return true;</code>
6	<code> return false;</code>
	<code>}</code>

Graful programului este:



Statement Coverage

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

Date de intrare	Răspuns
$T_1 = \{\{1,1\}\{2,2\}, -2, 2, 2\}$	Conditions not met.
$T_{12} = \{\{1,1\}\{2,2\}, 2, 2, -1\}$	Conditions not met.
$T_{13} = \{\{1,-1\}\{2,2\}, 2, 2, 2\}$	Conditions not met.
$T_4 = \{\{1,1\}\{2,2\}, 2, 2, 6\}$	True

Branch coverage

Instrucțiuni care duc la ramuri în program:

```
if(s == x)
```

```
if(a[i][j] < 0)
```

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

Date de intrare	Răspuns
T_1 = {{1,1}{2,2}, -2, -2, 2}	Conditions not met.
T_3 = {{1,1}{2,2}, 2, 2, -2}	Conditions not met.
T_5 = {{1,-3}{2,2}, 2, 2, 2}	Conditions not met.
T_2 = {{1,1}{2,2}, 2, 2, 6}	True
T_4 = {{1,1}{2,2}, 2, 2, 3}	Conditions not met.
T_6 = {{1,1}{2,2}, 2, 2, 4}	Conditions not met.
T_7 = {{1,1}{2,2}, 2, 2, 5}	Conditions not met.

Condition coverage

Deciziile din programul Java:

Decizii	Condiții individuale
for(int i=0;i<n;i++)	i < n
for(int j=0;j<m;j++)	j < m
if(s==x)	s == x
if(a[i][j]<0)	a[i][j] < 0

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

Date de intrare	Răspuns
T_1 = {{1,1}{2,2}, -2, 2, 2}	Conditions not met.
T_2 = {{1,1}{2,2}, 2, -2, 2}	Conditions not met.
T_3 = {{1,1}{2,2}, 2, 2, -2}	Conditions not met.
T_4 = {{1,-3}{2,2}, 2, 2, 2}	True
T_5 = {{1,1}{2,2}, 2, 2, 6}	True

$T_6 = \{\{1,1\}\{2,2\}, 2, 2, 5\}$	Conditions not met.
-------------------------------------	---------------------

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:

$V(G) = e - n + 1$, 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: (5, 1) și (7, 1)

Atunci: $V(G) = 8 - 7 + 1 = 2$.

Circuite independente:

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

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:

$1.2.(3 + 4).(5+6).4*.7$

Numărul de căi: 3

Căile posibile sunt:

1.2.3.5

1.2.4.5

1.2.4.6.7

Date de test:

Date test	Răspuns
T_1 = {{1,1}{2,2}, -2, 2, 2}	Conditions not met.
T_2 = {{1,1}{2,2}, 2, 2, -2}	Conditions not met.
T_3 = {{1,1}{2,2}, 2, 2, 6}	True

Generator de mutanți

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

<http://eclipse.pitest.org/release/>

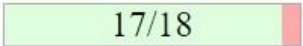
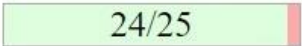
Pit Test Coverage Report

Package Summary

tss.proiect

Number of Classes	Line Coverage	Mutation Coverage
1	94%  17/18	96%  24/25

Breakdown by Class

Name	Line Coverage	Mutation Coverage
App.java	94%  17/18	96%  24/25

Report generated by [PIT](#) 1.1.9

Observăm că unul dintre testele de mai sus omoară 24 din 25 de mutanți. Unul dintre mutanții neomorâți este un ConditionalsBoundaryMutator care este generat la linia (14) în cod:

```
if(a[i][j]<0) throw new IllegalArgumentException("Conditions not met.");
```

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

```
assertTrue(checkInMatrix([0,0][0,0], 2, 2, 6));
```