# Testarea Sistemelor Software

## Unit Testing (Java)


Echipa:

Atasie Oana-Andreea (341)

Burta Mihai-Catalin (342)

Matei Andreea-Madalina (341)

Stoica Liviu (342)

Tudorache Alexandru-Theodor (342)

# Cuprins

## Specificația problemei:

Date fiind numerele a, b, k, s, să se găsească primele k numere prime din intervalul [a, b] care au suma cifrelor egală cu s, având următoarele condiții:

- k >= 0
- s >= 0
- a >= 0
- b >= 0
- a <= b

## Implementarea soluției:

```
public static List<Integer> findPrimes(int k, int a, int b, int s)
throws IllegalArgumentException {
   if ( k < 0 ) throw new IllegalArgumentException("K is negative.");
   if ( s < 0 ) throw new IllegalArgumentException("S is negative.");
   if ( a < 0 || b < 0) throw new IllegalArgumentException("Range is
negative.");
   if ( a > b ) throw new IllegalArgumentException("Range is
reversed.");

   List<Integer> primes = new ArrayList<>();
   int copy;
   int digit;
   int sum;
   boolean found;
   int number, divisor;

   for(number = a; primes.size() < k && number <= b; number++)
   {
      if(number == 0 || number == 1) continue;
      found = false;
      for(divisor = 2; !found && divisor <= sqrt(number); divisor++)
      {
         if(number % divisor == 0) found = true;
      }

      if(!found)
      {
         copy = number;
         sum = 0;
         while(copy != 0)
         {
            digit = copy % 10;
            copy = copy / 10;
            sum += digit;
         }
         if(sum == s) primes.add(number);
      }
   }
   return primes;
```

```
}
```

# 1. Testare Funcțională

## a) Partiționarea de echivalență

Există 4 date de intrare:

1. un întreg pozitiv K;
2. un întreg pozitiv A;
3. un întreg pozitiv B;
4. un întreg pozitiv S;

Pentru fiecare dată de intrare se disting mai multe clase, astfel:

- Pentru K:
    1. $K < 0$
    2. $K \geq 0$
- Pentru A și B:
    1. $A \leq B$
    2. $A > B$
    3. $A < 0$ sau $B < 0$
- Pentru S:
    1. $S < 0$
    2. $S \geq 0$

Există 2 tipuri de date de ieșire:

1. Eroare
2. Listă

Pentru tipurile de ieșire se disting 5 clase, după cum urmează:

- Pentru Eroare:
    1. $K < 0 \Rightarrow$ “K is negative.”
    2. $A > B \Rightarrow$ “Range is reversed.”
    3. $S < 0 \Rightarrow$ “S is negative.”
    4. $A < 0$ sau $B < 0 \Rightarrow$ “Range is negative.”

- Pentru listă:
    1. lista cu elementele căutate

Din clasele individuale anterioare rezultă 5 clase de echivalență pentru întregul program:

```
C_1    = { (k, a, b, s) | k < 0 }

C_22   = { (k, a, b, s) | k ≥ 0, a > b }

C_211  = { (k, a, b, s) | k ≥ 0, 0 ≤ a ≤ b, s < 0 }

C_23   = { (k, a, b, s) | k ≥ 0, s ≥ 0, a < 0 sau b < 0 }

C_2122 = { (k, a, b, s) | k ≥ 0, 0 ≤ a ≤ b, s ≥ 0 }
```

Pentru fiecare clasă de echivalență vom scrie câte un test, spre exemplu:

```
C_1    = (-3, 1, 20, 3)

C_22   = (3, 20, 1, 3)

C_211  = (3, 1, 20, -5)

C_23   = (3, -5, 1, 3)

C_2122 = (5, 1, 1000, 5)
```

| **Intrări** | | | | **Expected** |
|---|---|---|---|---|
| **K** | **A** | **B** | **S** | |
| -3 | 1 | 20 | 3 | K is negative. |
| 3 | 20 | 1 | 3 | Range is reversed. |
| 3 | 1 | 20 | -5 | S is negative. |
| 3 | -5 | 1 | 3 | Range is negative. |
| 5 | 1 | 1000 | 5 | [5, 23, 41, 113, 131] |

```
@Test
void aOrBLessThanZero() {
    List expectedList;
```

```
    // a < 0 or b < 0
    try {
        expectedList = new ArrayList(Arrays.asList());
        Assertions.assertEquals(expectedList, Main.findPrimes(3, -5,
1, 3));
        Assertions.fail();
    } catch (IllegalArgumentException e) {
        Assertions.assertEquals(new IllegalArgumentException("Range is
negative.").toString(), e.toString());
    }
}

@Test
void goodFindPrimes() {
    List expectedList;

    // Good
    try {
        expectedList = new ArrayList(Arrays.asList(5, 23, 41, 113,
131));
        Assertions.assertEquals(expectedList, Main.findPrimes(5, 1,
1000, 5));
    }
    catch (IllegalArgumentException e) {
        Assertions.fail();
    }
}
```

### b) Analiza valorilor de frontieră

Conform claselor de echivalență avem urmatoarele valori de frontieră:

- Valorile -5, 5 pentru K
- A si B pot fi fie ambele <0, fie ambele >=0 si A>B, fie amebele >0 si A<B
- Valorile -7,7 pentru S

Deci vom testa urmatoarele valori:

- K_1: -5
- K_2: 5
- AB_1: -4,5 (A<0, B_ sau A_, B<0)
- AB_2: 7,5  (A,B>0, A>B)
- AB_3: 7,9  (A,B>0, A<B)
- S_1: -7
- S_2: 7

Avem urmatoarele date de test:

`C_1 : (-5,_,_,_)`

C_22: (5,7,5,_)

C_211: (5,7,9,-7)

C_23: (5,-4,5,_)

C_2122: (5,7,9,7)

| Intrări | | | | Expected |
|---|---|---|---|---|
| **K** | **A** | **B** | **S** | |
| -5 | _ | _ | _ | K is negative. |
| 5 | 7 | 5 | _ | Range is reversed. |
| 5 | 1 | 5 | -5 | S is negative. |
| 5 | -4 | 5 | 5 | Range is negative. |
| 5 | 7 | 9 | 7 | [7] |

Exemple de implementare a testelor:

```
@Test
void kIsNegative() {
	try {
		resultList = Main.findPrimes(-5, 0, 0, 0);
		Assertions.fail("K should be negative.");
	}
	catch (IllegalArgumentException e) {
		expectedErrorMessage = "K is negative.";
		Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
	}
}

 @Test
void aIsNotNegativeandsbIsNotNegative() {
	try {
		resultList = Main.findPrimes(5, 7, 5, 5);
		Assertions.fail("Range should be reversed.");
	}
	catch (IllegalArgumentException e) {
		expectedErrorMessage = "Range is reversed.";
		Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
	}
}

@Test
```

```
void sIsNegative() {
	try {
		resultList = Main.findPrimes(5, 1, 5, -5);
		Assertions.fail("S should be negative.");
	}
	catch (IllegalArgumentException e) {
		expectedErrorMessage = "S is negative.";
		Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
	}
}

@Test
void sIsNotNegative() {
	try {
		resultList = Main.findPrimes(5, -4, 5, 5);
		Assertions.fail("Range should be negative.");
	}
	catch (Exception e) {
		expectedErrorMessage = "Range is negative.";
		Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
	}
}

@Test
void notFoundAndDivisorLessThanSqrtNumber() {
	try {
		expectedList = new ArrayList<>(Arrays.asList(7));
		resultList = Main.findPrimes(5, 7, 9, 7);
		Assertions.assertEquals(expectedList, resultList);
	}
	catch (IllegalArgumentException e) {
		Assertions.fail("Encountered exception: " +
e.getMessage());
	}
}
```

### c) Partiționarea în categorii

Am parcurs pașii corespunzători metodei de testare:

1. Descompune specificația în unități: avem o singură unitate.

2. Identifică parametrii: k, a, b, s, numerele din [a, b]

3. Găsește categorii:

- k: Dacă este mai mare decât 0
- a: Dacă este mai mare decât 0
- b: Dacă este mai mare decât 0
- Relația dintre a și b: Dacă a <= b sau a > b
- Între a și b există cel puțin un număr prim/niciunul.

- s: Dacă este mai mare decât 0
- Între a și b exista cel puțin un număr cu suma cifrelor s/niciunul.

4. Partiționeaza fiecare categorie în alternative:

- k: < 0, 0, > 0
- a: < 0, 0, 1, > 1, număr prim
- b: < 0, 0, 1, > 1, număr prim
- a & b: a < b; a = b; a > b
- [a, b]: Niciun număr prim în interval; Cel puțin un număr prim în interval
- s: < 0, 0, > 0;
- [a, b] & s: Niciun număr cu suma cifrelor s în interval; Cel puțin un număr cu suma cifrelor s în interval

5. Scrie specificația de testare

- *k*

  1) {k | k < 0}

  2) k = 0

  3) {k | k > 0}

- *a*

  1) {a | a < 0}

  2) a = 0

  3) a = 1

  4) {a | a > 1, a nu e prim}

  5) {a | a > 1, a e prim}

- *b*

  1) {b | b < 0}

  2) b = 0

  3) b = 1

  4) {b | b > 1 && b < a}

5) {b | b > 1 && b = a, b nu e prim}

6) {b | b > 1 && b > a, b nu e prim}

7) {b | b > 1 && b = a, b e prim}

8) {b | b > 1 && b > a, b e prim}

- *s*

1) {s | s < 0}

2) s = 0

3) {s | s > 0}

- *n din [a, b] ->*

1) |{n | a <= n <= b, n nu e prim}| = b - a + 1

2) |{n | a <= n <= b, n e prim}| >= 1

- *n în funcție de s ->*

1) |{n | a <= n <= b, suma cifrelor lui n nu este s}| >= 1

2) |{n | a <= n <= b, suma cifrelor lui n este s}| = b - a + 1

6. Creează cazuri de testare

| | | | | |
|---|---|---|---|---|
| k1 | k2 | k3a1 | k3a2b1 | k3a2b2 |
| k3a2b3 | k3a2b6s1 | k3a2b6s2 | k3a2b6s3nab2ns1 | k3a2b6s3nab2ns2 |
| k3a2b8s1 | k3a2b8s2 | k3a2b8s3nab2ns1 | k3a2b8s3nab2ns2 | k3a3b1 |
| k3a3b2 | k3a3b3 | k3a3b6s1 | k3a3b6s2 | k3a3b6s3nab2ns1 |
| k3a3b6s3nab2ns2 | k3a3b8s1 | k3a3b8s2 | k3a3b8s3nab2ns1 | k3a3b8s3nab2ns2 |
| k3a4b1 | k3a4b2 | k3a4b3 | k3a4b4 | k3a4b5s1 |
| k3a4b5s2 | k3a4b5s3nab1ns1 | k3a4b5s3nab1ns2 | k3a4b6s1 | k3a4b6s2 |
| k3a4b6s3nab1ns1 | k3a4b6s3nab1ns2 | k3a4b6s3nab2ns1 | k3a4b6s3nab2ns2 | k3a4b8s1 |
| k3a4b8s2 | k3a4b8s3nab2ns1 | k3a4b8s3nab2ns2 | k3a5b1 | k3a5b2 |
| k3a5b3 | k3a5b4 | k3a5b4 | k3a5b6s2 | k3a5b6s3nab2ns1 |
| k3a5b6s3nab2ns2 | k3a5b7s1 | k3a5b7s2 | k3a5b7s3nab2ns1 | k3a5b7s3nab2ns2 |
| k3a5b8s1 | k3a5b8s2 | k3a5b8s3nab2ns1 | k3a5b8s3nab2ns2 | |

În total: 59 cazuri de testare

7. Creează date de test

| **Nume test** | **Intrari** | | | | **Rezultat afișat (expected)** |
|---|---|---|---|---|---|
| | **k** | **a** | **b** | **s** | |
| k1 | -1 | | | | Exceptie: K is negative |
| k2 | 0 | 0 | 0 | 0 | [] |
| k3a1 | 1 | -1 | | | Exceptie: Range is negative |
| k3a2b1 | 1 | 0 | -1 | | Exceptie: Range is negative |
| k3a2b2 | 1 | 0 | 0 | 0 | [] |
| k3a2b3 | 1 | 0 | 1 | 0 | [] |
| k3a2b6s1 | 1 | 0 | 4 | -1 | Exceptie: S is negative |
| k3a2b6s2 | 1 | 0 | 4 | 0 | [] |
| k3a2b6s3nab2ns1 | 1 | 0 | 9 | 17 | [] |
| k3a2b6s3nab2ns2 | 1 | 0 | 14 | 2 | [2] |
| k3a2b8s1 | 1 | 0 | 3 | -1 | Excepție: S is negative |
| k3a2b8s2 | 1 | 0 | 3 | 0 | [] |
| k3a2b8s3nab2ns1 | 1 | 0 | 7 | 17 | [] |
| k3a2b8s3nab2ns2 | 1 | 0 | 7 | 7 | [7] |
| k3a3b1 | 1 | 1 | -1 | 0 | Excepție: Range is negative |
| k3a3b2 | 1 | 1 | 0 | 0 | Excepție: Range is reversed |
| k3a3b3 | 1 | 1 | 1 | 1 | [] |
| k3a3b6s1 | 1 | 1 | 2 | -1 | Excepție: S is negative |
| k3a3b6s2 | 1 | 1 | 2 | 0 | [] |
| k3a3b6s3nab2ns1 | 1 | 1 | 9 | 17 | [] |
| k3a3b6s3nab2ns2 | 2 | 1 | 14 | 2 | [2, 11] |
| k3a3b8s1 | 3 | 1 | 3 | -1 | Excepție: S is negative |
| k3a3b8s2 | 5 | 1 | 3 | 0 | [] |
| k3a3b8s3nab2ns1 | 2 | 1 | 7 | 17 | [] |
| k3a3b8s3nab2ns1 | 1 | 1 | 7 | 7 | [7] |
| k3a4b1 | 3 | 4 | -1 | 0 | Excepție: Range is negative |
| k3a4b2 | 4 | 4 | 0 | 0 | Excepție: Range is reversed |
| k3a4b3 | 2 | 4 | 1 | 0 | Excepție: Range is reversed |
| k3a4b4 | 8 | 4 | 2 | 0 | Excepție: Range is reversed |

| | | | | | |
|---|---|---|---|---|---|
| k3a4b5s1 | 1 | 4 | 4 | -1 | Excepție: S is negative |
| k3a4b5s2 | 9 | 4 | 4 | 0 | [] |
| k3a4b5s3nab1ns1 | 3 | 4 | 4 | 5 | [] |
| k3a4b5s3nab1ns2 | 5 | 4 | 4 | 4 | [] |
| k3a4b6s1 | 6 | 24 | 28 | -1 | Excepție: S is negative |
| k3a4b6s2 | 2 | 24 | 28 | 0 | [] |
| k3a4b6s3nab1ns1 | 1 | 24 | 28 | 19 | [] |
| k3a4b6s3nab1ns2 | 4 | 24 | 28 | 9 | [] |
| k3a4b6s3nab2ns1 | 8 | 27 | 30 | 19 | [] |
| k3a4b6s3nab2ns2 | 11 | 27 | 30 | 11 | [29] |
| k3a4b8s1 | 3 | 4 | 5 | -1 | Excepție: S is negative |
| k3a4b8s2 | 3 | 4 | 5 | 0 | [] |
| k3a4b8s3nab2ns1 | 10 | 24 | 29 | 100 | [] |
| k3a4b8s3nab2ns2 | 8 | 24 | 29 | 11 | [29] |
| k3a5b1 | 8 | 3 | -1 | 0 | Excepție: Range is negative |
| k3a5b2 | 7 | 3 | 0 | 0 | Excepție: Range is reversed |
| k3a5b3 | 6 | 3 | 1 | 0 | Excepție: Range is reversed |
| k3a5b4 | 4 | 3 | 2 | 0 | Excepție: Range is reversed |
| k3a5b6s1 | 5 | 23 | 28 | -1 | Excepție: S is negative |
| k3a5b6s2 | 2 | 23 | 28 | 0 | [] |
| k3a5b6s3nab2ns1 | 10 | 23 | 27 | 19 | [] |
| k3a5b6s3nab2ns2 | 16 | 23 | 27 | 5 | [23] |
| k3a5b7s1 | 9 | 23 | 23 | -1 | Excepție: S is negative |
| k3a5b7s2 | 3 | 23 | 23 | 0 | [] |
| k3a5b7s3nab2ns1 | 1 | 23 | 23 | 19 | [] |
| k3a5b7s3nab2ns2 | 4 | 23 | 23 | 5 | [23] |
| k3a5b8s1 | 2 | 3 | 5 | -1 | Excepție: S is negative |
| k3a5b8s2 | 6 | 3 | 5 | 0 | [] |
| k3a5b8s3nab2ns1 | 5 | 23 | 29 | 100 | [] |
| k3a5b8s3nab2ns2 | 8 | 23 | 29 | 11 | [29] |

Exemple de implementare a testelor:

```
@Test
void k1() {
```

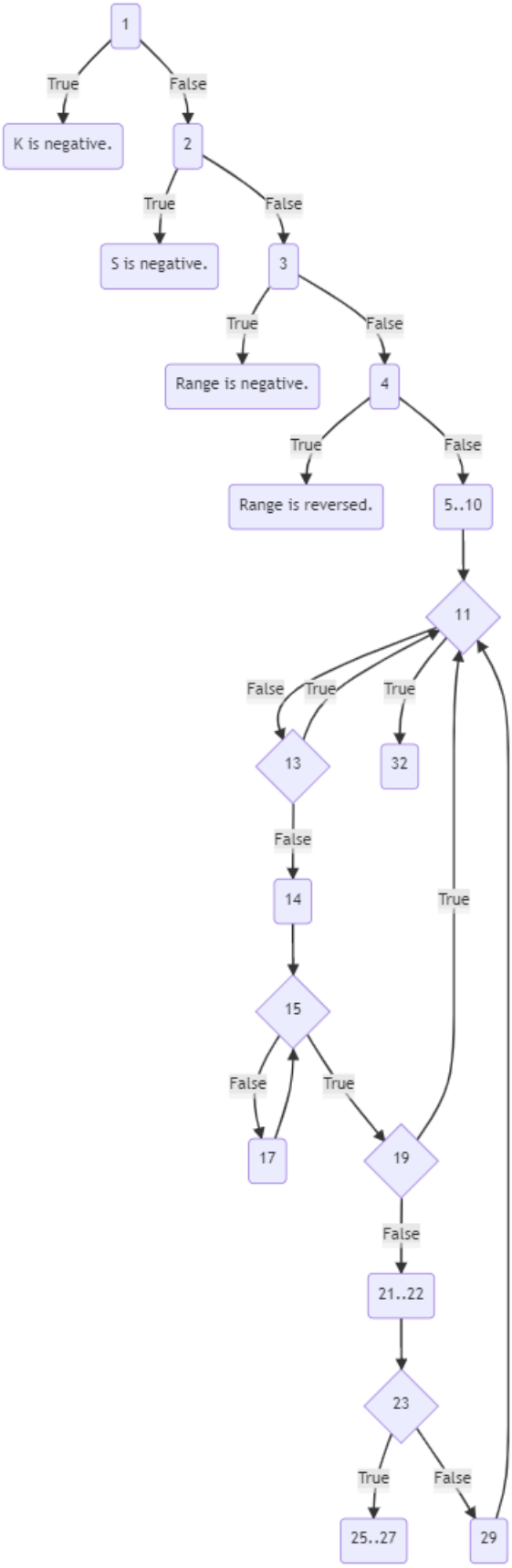
```
      try {
            resultList = Main.findPrimes(-1, 0, 0, 0);
            Assertions.fail("K should be negative.");
      } catch (IllegalArgumentException e) {
            expectedErrorMessage = "K is negative.";
            Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
      }
}

@Test
void k3a2b3() {
      try {
            expectedList = new ArrayList<>();
            resultList = Main.findPrimes(1, 0, 1, 0);
            Assertions.assertEquals(expectedList, resultList);
      } catch (IllegalArgumentException e) {
            Assertions.fail("Encountered exception: " +
e.getMessage());
      }
}

@Test
void k3a3b2() {
      try {
            resultList = Main.findPrimes(1, 1, 0, 0);
            Assertions.fail("Range should be reversed.");
      } catch (IllegalArgumentException e) {
            expectedErrorMessage = "Range is reversed.";
            Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
      }
}

@Test
void k3a4b5s2() {
      try {
            expectedList = new ArrayList<>();
            resultList = Main.findPrimes(9, 4, 4, 0);
            Assertions.assertEquals(expectedList, resultList);
      } catch (IllegalArgumentException e) {
            Assertions.fail("Encountered exception: " +
e.getMessage());
      }
}

@Test
void k3a5b7s3nab2ns2() {
      try {
            expectedList = new ArrayList<>(Arrays.asList(23));
            resultList = Main.findPrimes(4, 23, 23, 5);
            Assertions.assertEquals(expectedList, resultList);
      } catch (IllegalArgumentException e) {
            Assertions.fail("Encountered exception: " +
e.getMessage());
      }
}
```

2. Testare Structurală

**a) Acoperire la nivel de instrucțiune**

Este necesară numerotarea instrucțiunilor din cod:

```
1.  if ( k < 0 ) throw new Exception("K is negative.");
2.  if ( s < 0 ) throw new Exception("S is negative.");
3.  if ( a < 0 || b < 0) throw new Exception("Range is negative.");
4.  if ( a > b ) throw new Exception("Range is reversed.");
5.  List primes = new ArrayList();
6.  int copy;
7.  int digit;
8.  int sum;
9.  boolean found;
10. int number, divisor;
11. for(number = a; primes.size() <= k && number <= b; number++)
12. {
13.     if(number == 1) continue;
14.     found = false;
15.     for(divisor = 2; !found && divisor <= sqrt(number); divisor++)
16.     {
17.         if(number % divisor == 0) found = true;
18.     }
19.     if(!found)
20.     {
21.         copy = number;
22.         sum = 0;
23.         while(copy != 0)
24.         {
25.             digit = copy % 10;
26.             copy = copy / 10;
27.             sum += digit;
28.         }
29.         if(sum == s) primes.add(number);
30.     }
31. }
32. return primes;
```


1
True
False
K is negative.
2
True
False
S is negative.
3
True
False
Range is negative.
4
True
False
Range is reversed.
5..10
11
False
True
True
13
32
False
14
True
15
False
True
17
19
False
21..22
23
True
False
25..27
29

| Intrări | | | | Rezultat afișat | Instrucțiuni parcurse |
|---|---|---|---|---|---|
| **K** | **A** | **B** | **S** | | |
| -3 | 1 | 20 | 3 | K is negative. | 1 |
| 3 | 20 | 1 | 3 | Range is reversed. | 4 |
| 3 | 1 | 20 | -5 | S is negative. | 2 |
| 3 | -5 | 1 | 3 | Range is negative. | 3 |
| 1 | 1 | 5 | 5 | [5] | 5..10, 11..12, 13, 11..12, 14, 19..22, 23..28, 11..12, 14, 19..22, 23..28, 11..12, 15, 16..18, 11..12, 14, 15, 16, 18, 19..22, 23..28, 29, 31..32 |

```
@Test
void aOrBLessThanZero() {
    List expectedList;

    // a < 0 or b < 0
    try {
        expectedList = new ArrayList(Arrays.asList());
        Assertions.assertEquals(expectedList, Main.findPrimes(3, -5,
1, 3));
        Assertions.fail();
    } catch (IllegalArgumentException e) {
        Assertions.assertEquals(new IllegalArgumentException("Range is
negative.").toString(), e.toString());
    }
}

@Test
void goodFindPrimes() {
    List expectedList;

    // Good
    try {
        expectedList = new ArrayList(Arrays.asList(5));
        Assertions.assertEquals(expectedList, Main.findPrimes(1, 1, 5,
5));
    }
    catch (IllegalArgumentException e) {
        Assertions.fail();
    }
}
```

**b) Acoperire la nivel de decizie sau acoperire la nivel de ramură**

Am numerotat deciziile astfel:

1. `if ( k < 0 ) throw new IllegalArgumentException("K is negative.");`
2. `if ( s < 0 ) throw new IllegalArgumentException("S is negative.");`
3. `if ( a < 0 || b < 0) throw new IllegalArgumentException("Range is negative.");`
4. `if ( a > b ) throw new IllegalArgumentException("Range is reversed.");`
5. `for(number = a; primes.size() < k && number <= b; number++)`
6. `if(number == 0 || number == 1)`
7. `for(divisor = 2; !found && divisor <= sqrt(number); divisor++)`
8. `if(number % divisor == 0)`
9. `if(!found)`
10. `while(copy != 0)`
11. `if(sum == s)`

Pentru a acoperi toate cazurile, atat cu adevarat cat si cu fals, avem urmatoarele teste:

1 → A - findPrimes(-1, 0, 0, 0)

→ F - findPrimes(0, 0, 0, -1)

2 → A - findPrimes(0, 0, 0, -1)

→ F - findPrimes(0, -1, 0, 0)

3 → A - findPrimes(0, -1, 0, 0)

→ F - findPrimes(0, 2, 1, 0)

4 → A - findPrimes(0, 2, 1, 0)

→ F - findPrimes(1, 1, 3, 2)

5 → A - findPrimes(1, 1, 3, 2)

→ F - findPrimes(1, 1, 3, 2)

6 → A  - findPrimes(1, 1, 3, 2)

  → F  - findPrimes(1, 1, 3, 2)

7 → A  - findPrimes(5, 1, 5, 2)

  → F  - findPrimes(1, 1, 3, 2)

8 → A  - findPrimes(5, 1, 5, 2)

  → F  - findPrimes(5, 1, 5, 2)

9 → A  - findPrimes(1, 1, 3, 2)

  → F  - findPrimes(5, 1, 5, 2)

10 → A  - findPrimes(1, 1, 3, 2)

  → F  - findPrimes(1, 1, 3, 2)

11 → A  - findPrimes(1, 1, 3, 2)

  → F  - findPrimes(5, 1, 5, 2)

Testele ce implementeaza toate cazurile:

```
      @Test // 1 T
      void kIsNegative() {
            try {
                  Main.findPrimes(-1, 0, 0, 0);
                  Assertions.fail("K should be positive.");
            }
            catch (Exception e) {
                  String expectedErrorMessage = "K is negative.";
                  Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
            }
      }

      @Test // 1 F, 2 T
      void sIsNegative() {
            try {
                  Main.findPrimes(0, 0, 0, -1);
                  Assertions.fail("S should be positive.");
            }
            catch (Exception e) {
                  String expectedErrorMessage = "S is negative.";
                  Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
            }
      }
```

```
	@Test // 2 F, 3 T
	void aOrBIsNegative() {
		try {
			Main.findPrimes(0, -1, 0, 0);
			Assertions.fail("Range should be positive.");
		}
		catch (Exception e) {
			String expectedErrorMessage = "Range is negative.";
			Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
		}
	}

	@Test // 3 F, 4 T
	void aGreaterThanB() {
		try {
			Main.findPrimes(0, 2, 1, 0);
			Assertions.fail("B should be greater than A.");
		}
		catch (Exception e) {
			String expectedErrorMessage = "Range is reversed.";
			Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
		}
	}

	@Test // 4 F, 5 T, 5 F, 6 T, 6 F, 7 F, 9 T, 10 T, 10 F, 11 T
	void test1() {
		try {
			List<Integer> expectedResut = Main.findPrimes(1, 1,
3, 2);
			Assertions.assertEquals(List.of(2),expectedResut);
		}
		catch (Exception e) {
			Assertions.fail(e.getMessage());
		}
	}

	@Test //  7 A, 8 A, 8 F, 9 F, 11 F
	void test2() {
		try {
			List<Integer> expectedResut = Main.findPrimes(5, 1,
5, 2);
			Assertions.assertEquals(List.of(2),expectedResut);
		}
		catch (Exception e) {
			Assertions.fail(e.getMessage());
		}
	}
```

**c) Acoperire la nivel de condiție**

| Decizii | Conditii individuale |
|---|---|
| if ( k < 0 ) | k<0 |
| if ( s < 0 ) | s<0 |
| if ( a < 0 \|\| b < 0) | a<0,b<0 |
| if ( a > b ) | a>b |
| for(number = a; primes.size() < k && number <= b | Primes.size()<k,number<=b |
| if(number == 0 \|\| number == 1) | number==0,number==1 |
| for(divisor = 2; !found && divisor <= sqrt(number); | Found,divisor<=sqrt(number) |
| if(number % divisor == 0) | number==0,divisor==0 |
| if(!found) | found |
| while(copy != 0) | copy!=0 |
| if(sum == s) | sum==s |

1) k<0
   A: (-5,0,0,0)
   F: (5,0,0,-5)
2) s < 0
   A: (5,0,0,-5)
   F: (1,-4,5,5)

3) a < 0 || b < 0
   A || A : (5,-5,-5,5)
   F || F : (5,7,5,5)
4) a>b
   A: (5, 5, 1, 1)
   F: (5, 1, 5, 1)
5) primes.size() < k && number <= b
   A && F: (1, 5, 5, 10) -> de la pasul 2
   F && A: (0, 1, 1, 1) -> la primul pas
6) number == 0 || number == 1
   A || F: (5, 0, 5, 7) -> la primul pas

F || A: (5, 1, 5, 7) -> la primul pas

7) !found && divisor <= sqrt(number)

   A && F: (5, 7, 9, 7) -> la primul pas din ambele for-uri

   F && A: (5, 6, 8, 6) -> primul pas din ambele for-uri

8) number % divisor == 0

   A: (5, 2, 9, 1) -> la primul pas din ambele for-uri

   F: (1, 3, 7, 9) -> la primul pas din ambele for-uri

9) !found

   A: (1, 7, 9, 7) -> la primul pas din ambele for-uri (7 e numar prim)

   F: (1, 6, 8, 6) -> la primus pas din ambele for-uri (6%2==0)

10) copy != 0

   A: (1, 5, 7, 5) -> la primul pas din while, copy = 5

   F: (1, 5, 7, 5) -> la al doilea pas din while, copy = 0

11) sum == s

   A: (1, 5, 9, 5)

   F: (1, 5, 9, 9)

Exemple de implementare a testelor:

```
@Test
    void kIsNegative() {
      try {
            resultList = Main.findPrimes(-5, 0, 0, 0);
            Assertions.fail("K should be negative.");
      }
      catch (IllegalArgumentException e) {
            expectedErrorMessage = "K is negative.";
            Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
      }
}

@Test
void sIsNegative() {
      try {
            resultList = Main.findPrimes(5, 0, 0, -5);
            Assertions.fail("S should be negative.");
      }
      catch (IllegalArgumentException e) {
            expectedErrorMessage = "S is negative.";
            Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
      }
}

@Test
```

```
void aIsNegativeandsbIsNegative() {
	try {
		resultList = Main.findPrimes(5, -5, -5, 5);
		Assertions.fail("Range should be negative.");
	}
	catch (IllegalArgumentException e) {
		expectedErrorMessage = "Range is negative.";
		Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
	}
}

@Test
void aIsGreaterThanb() {
	try {
		resultList = Main.findPrimes(5, 5, 1, 1);
		Assertions.fail("Range should be reversed.");
	}
	catch (IllegalArgumentException e) {
		expectedErrorMessage = "Range is reversed.";
		Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
	}
}

@Test
void notFoundAndDivisorLessThanSqrtNumber() {
	try {
		expectedList = new ArrayList<>(Arrays.asList(7));
		resultList = Main.findPrimes(5, 7, 9, 7);
		Assertions.assertEquals(expectedList, resultList);
	}
	catch (IllegalArgumentException e) {
		Assertions.fail("Encountered exception: " +
e.getMessage());
	}
}
```

### d) Acoperire la nivel de condiție/decizie

Pentru instructiunile simple sunt aceleasi exemple, pentru cele compuse:

1) primes.size() < k && number <= b

   -> A && A: (5, 1, 5, 1) -> la primul pas -> decizie A

   -> F && F: (1, 1, 2, 2) -> la pasul 3 -> -> decizie F

2) number == 0 || number == 1

   cazurile precendete +

   F && F: (1, 6, 8, 5) -> la pasul 1 -> decizie F

3) !found && divisor <= sqrt(number)

   A && A: (1, 4, 4, 4) – la primul pas -> decizie A

F && F: (1, 9, 9, 9) la primul pas din primul for, al treilea pas din al doilea for -> decizie F

Exemple de implementare a testelor:

```
@Test
void sizeLowerThankNumberLowerThanb() {
      try {
            expectedList = new ArrayList<>(Arrays.asList());
            resultList = Main.findPrimes(5, 1, 5, 1);
            Assertions.assertEquals(expectedList, resultList);
      }
      catch (IllegalArgumentException e) {
            Assertions.fail("Encountered exception: " +
e.getMessage());
      }
}

@Test
void notFoundDivisorLessThanSqrtNumber() {
      try {
            expectedList = new ArrayList<>();
            resultList = Main.findPrimes(1, 4, 4, 4);
            Assertions.assertEquals(expectedList, resultList);
      }
      catch (IllegalArgumentException e) {
            Assertions.fail("Encountered exception: " +
e.getMessage());
      }
}

@Test
void FoundAndDivisorHigherThanSqrtNumberFalseFalse() {
      try {
            expectedList = new ArrayList<>();
            resultList = Main.findPrimes(1, 9, 9, 9);
            Assertions.assertEquals(expectedList, resultList);
      }

      catch (IllegalArgumentException e) {
            Assertions.fail("Encountered exception: " +
e.getMessage());
      }
}
```

### e) Acoperire la nivel de condiții multiple

În programul nostru avem 11 decizii:

- k < 0
- s < 0
- a < 0 || b < 0
- a > b

- primes.size() < k && number <= b
- number == 0 || number == 1
- !found && divisor <= sqrt(number)
- number % divisor == 0
- !found
- copy != 0
- sum == s

Trebuie generate date de test astfel încât să fie parcurse toate combinațiile posibile de adevărat și fals ale conditiilor individuale.

Pentru a realiza acest lucru, am generat următoarele date de test:

| **Condiția** | **Valorile de adevăr** | **Intrări** | | | | **Când se ajunge la respectivele valori de adevăr** |
|---|---|---|---|---|---|---|
| | | ***k*** | ***a*** | ***b*** | ***s*** | |
| k < 0 | A | -1 | 0 | 0 | -1 | |
| k < 0 | F | 1 | 0 | 0 | -1 | |
| s < 0 | A | 1 | 0 | 0 | -1 | |
| s < 0 | F | 1 | -1 | 0 | 1 | |
| a < 0 \|\| b < 0 | A \|\| A | 1 | -1 | -1 | 1 | |
| a < 0 \|\| b < 0 | A \|\| F | 1 | -1 | 0 | 1 | |
| a < 0 \|\| b < 0 | F \|\| A | 1 | 0 | -1 | 1 | |
| a < 0 \|\| b < 0 | F \|\| F | 1 | 1 | 0 | 1 | |
| a > b | A | 1 | 1 | 0 | 1 | |
| a > b | F | 1 | 0 | 0 | 1 | |
| primes.size() < k && number <= b | A && A | 1 | 1 | 2 | 2 | Primul pas din *for* |
| primes.size() < k && number <= b | A && F | 1 | 1 | 2 | 1 | Al doilea pas din *for* |
| primes.size() < k && number <= b | F && A | 0 | 1 | 1 | 1 | Primul pas din *for* |
| primes.size() < k && number <= b | F && F | 1 | 1 | 2 | 2 | Al treilea pas din *for* |
| number == 0 \|\| number == 1 | A \|\| A | | | | | Niciodata (este imposibil) |
| number == 0 \|\| number == 1 | A \|\| F | 1 | 0 | 2 | 1 | Primul pas din *for* |
| number == 0 \|\| number == 1 | F \|\| A | 1 | 1 | 2 | 1 | Primul pas din *for* |
| number == 0 \|\| number == 1 | F \|\| F | 1 | 2 | 2 | 2 | Primul pas din *for* |
| !found && divisor <= sqrt(number) | A && | 1 | 4 | 4 | 4 | Primul pas din primul *for*, primul pas |

| | | | | | | |
|---|---|---|---|---|---|---|
| | A | | | | | din al doilea *for* |
| !found && divisor <= sqrt(number) | A && F | 1 | 2 | 2 | 2 | Primul pas din primul *for*, primul pas din al doilea *for* |
| !found && divisor <= sqrt(number) | F && A | 1 | 12 | 12 | 3 | Primul pas din primul *for*, al doilea pas din al doilea *for* |
| !found && divisor <= sqrt(number) | F && F | 1 | 9 | 9 | 9 | Primul pas din primul *for*, al treilea pas din al doilea *for* |
| number % divisor == 0 | A | 1 | 4 | 4 | 4 | Primul pas din primul *for*, primul pas din al doilea *for* |
| number % divisor == 0 | F | 1 | 9 | 9 | 9 | Primul pas din primul *for*, primul pas din al doilea *for* |
| !found | A | 1 | 3 | 3 | 3 | Primul pas din primul *for* |
| !found | F | 1 | 4 | 4 | 4 | Primul pas din primul *for*, dupa primul pas din al doilea *for* |
| copy != 0 | A | 1 | 3 | 3 | 3 | Primul pas din primul *for*, primul pas din *while* |
| copy != 0 | F | 1 | 5 | 5 | 5 | Primul pas din primul *for*, al doilea pas din *while* |
| sum == s | A | 1 | 7 | 7 | 7 | |
| sum == s | F | 1 | 7 | 7 | 9 | |

Exemple de implementare a testelor:

```
@Test
void kLessThanZeroTrue() {
	try {
		resultList = Main.findPrimes(-1, 0, 0, -1);
		Assertions.fail("K should be negative.");
	}
	catch (IllegalArgumentException e) {
		expectedErrorMessage = "K is negative.";
		Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
	}
}

@Test
void aGreaterThanBTrue() {
	try {
		resultList = Main.findPrimes(1, 1, 0, 1);
		Assertions.fail("Range should be reversed.");
	}
	catch (IllegalArgumentException e) {
```

```
            expectedErrorMessage = "Range is reversed.";
            Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
      }
}

@Test
void numberEqualsZeroOrOneTrueFalse() {
      try {
            expectedList = new ArrayList<>();
            resultList = Main.findPrimes(1, 0, 2, 1);
            Assertions.assertEquals(expectedList, resultList);
      }
      catch (IllegalArgumentException e) {
            Assertions.fail("Encountered exception: " +
e.getMessage());
      }
}

@Test
void numberModDivisorTrue() {
      try {
            expectedList = new ArrayList<>();
            resultList = Main.findPrimes(1, 4, 4, 4);
            Assertions.assertEquals(expectedList, resultList);
      }
      catch (IllegalArgumentException e) {
            Assertions.fail("Encountered exception: " +
e.getMessage());
      }
}
```

**f) MC/DC**

În programul nostru avem 11 decizii:

- k < 0
- s < 0
- a < 0 || b < 0
- a > b
- primes.size() < k && number <= b
- number == 0 || number == 1
- !found && divisor <= sqrt(number)
- number % divisor == 0
- !found
- copy != 0
- sum == s

Trebuie generate date de test astfel încât:

- Fiecare condiție individuală dintr-o decizie ia atât valoare True cât și valoare False
- Fiecare decizie ia atât valoare True cât și valoare False
- Fiecare condiție individuală influențează în mod independent decizia din care face parte

Pentru a realiza acest lucru, am generat următoarele date de test:

| **Condiția** | **Valorile de adevăr** | **Intrări** | | | | **Când se ajunge la respectivele valori de adevăr** |
|---|---|---|---|---|---|---|
| | | ***k*** | ***a*** | ***b*** | ***s*** | |
| k < 0 | A | -1 | 0 | 0 | -1 | |
| k < 0 | F | 1 | 0 | 0 | -1 | |
| s < 0 | A | 1 | 0 | 0 | -1 | |
| s < 0 | F | 1 | -1 | 0 | 1 | |
| a < 0 \|\| b < 0 | A \|\| F | 1 | -1 | 0 | 1 | |
| a < 0 \|\| b < 0 | F \|\| A | 1 | 0 | -1 | 1 | |
| a < 0 \|\| b < 0 | F \|\| F | 1 | 1 | 0 | 1 | |
| a > b | A | 1 | 1 | 0 | 1 | |
| a > b | F | 1 | 0 | 0 | 1 | |
| primes.size() < k && number <= b | A && A | 1 | 1 | 2 | 2 | Primul pas din *for* |
| primes.size() < k && number <= b | A && F | 1 | 1 | 2 | 1 | Al doilea pas din *for* |
| primes.size() < k && number <= b | F && A | 0 | 1 | 1 | 1 | Primul pas din *for* |
| number == 0 \|\| number == 1 | A \|\| F | 1 | 0 | 2 | 1 | Primul pas din *for* |
| number == 0 \|\| number == 1 | F \|\| A | 1 | 1 | 2 | 1 | Primul pas din *for* |
| number == 0 \|\| number == 1 | F \|\| F | 1 | 2 | 2 | 2 | Primul pas din *for* |
| !found && divisor <= sqrt(number) | A && A | 1 | 4 | 4 | 4 | Primul pas din primul *for*, primul pas din al doilea *for* |
| !found && divisor <= sqrt(number) | A && F | 1 | 2 | 2 | 2 | Primul pas din primul *for*, primul pas din al doilea *for* |
| !found && divisor <= sqrt(number) | F && A | 1 | 12 | 12 | 3 | Primul pas din primul *for*, al doilea pas din al doilea *for* |
| number % divisor == 0 | A | 1 | 4 | 4 | 4 | Primul pas din primul *for*, primul pas din al doilea *for* |
| number % divisor == 0 | F | 1 | 9 | 9 | 9 | Primul pas din primul *for*, primul pas din al doilea *for* |

| | | | | | | |
|---|---|---|---|---|---|---|
| !found | A | 1 | 3 | 3 | 3 | Primul pas din primul *for* |
| !found | F | 1 | 4 | 4 | 4 | Primul pas din primul *for*, dupa primul pas din al doilea *for* |
| copy != 0 | A | 1 | 3 | 3 | 3 | Primul pas din primul *for*, primul pas din *while* |
| copy != 0 | F | 1 | 5 | 5 | 5 | Primul pas din primul *for*, al doilea pas din *while* |
| sum == s | A | 1 | 7 | 7 | 7 | |
| sum == s | F | 1 | 7 | 7 | 9 | |

Exemple de implementare a testelor:

```
@Test
void sLessThanZeroTrue() {
	try {
		resultList = Main.findPrimes(1, 0, 0, -1);
		Assertions.fail("S should be negative.");
	}
	catch (IllegalArgumentException e) {
		expectedErrorMessage = "S is negative.";
		Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
	}
}

@Test
void aLessThanZeroOrBLessThanZeroFalseTrue() {
	try {
		resultList = Main.findPrimes(1, 0, -1, 1);
		Assertions.fail("Range should be negative.");
	}
	catch (IllegalArgumentException e) {
		expectedErrorMessage = "Range is negative.";
		Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
	}
}

@Test
void notFoundAndDivisorLessThanSqrtNumberTrueTrue() {
	try {
		expectedList = new ArrayList<>();
		resultList = Main.findPrimes(1, 4, 4, 4);
		Assertions.assertEquals(expectedList, resultList);
	}
	catch (IllegalArgumentException e) {
		Assertions.fail("Encountered exception: " +
e.getMessage());
	}
}
```

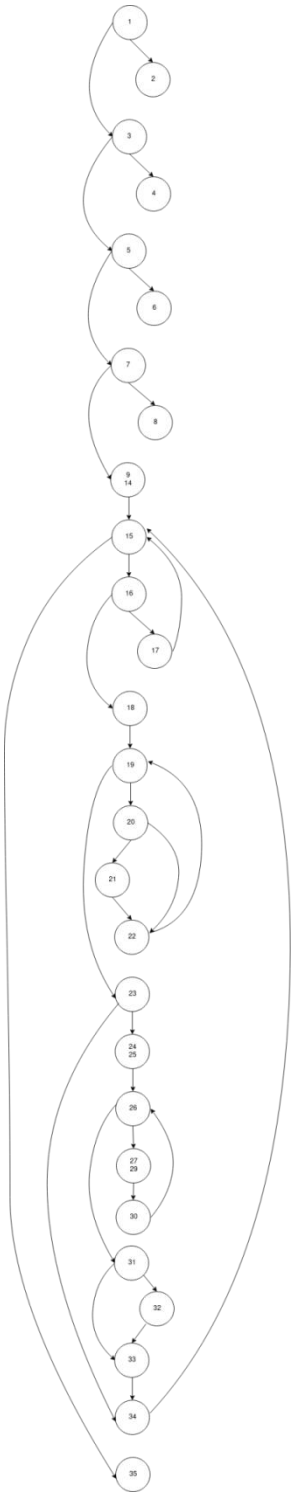
```
@Test
void sumEqualsSTrue() {
      try {
            expectedList = new ArrayList<>(Arrays.asList(7));
            resultList = Main.findPrimes(1, 7, 7, 7);
            Assertions.assertEquals(expectedList, resultList);
      }
      catch (IllegalArgumentException e) {
            Assertions.fail("Encountered exception: " +
e.getMessage());
      }
}
```

### g) Testarea circuitelor independente

Este necesară numerotarea instrucțiunilor din cod:

```
if ( k < 0 ) // 1
       throw new IllegalArgumentException("K is negative."); // 2
if ( s < 0 ) // 3
       throw new IllegalArgumentException("S is negative."); // 4
if ( a < 0 || b < 0) // 5
       throw new IllegalArgumentException("Range is negative."); // 6
if ( a > b ) // 7
       throw new IllegalArgumentException("Range is reversed."); // 8

List<Integer> primes = new ArrayList<>(); // 9
int copy; // 10
int digit; // 11
int sum; // 12
boolean found; // 13
int number, divisor; // 14

for(number = a; primes.size() < k && number <= b; number++){ // 15
       if(number == 0 || number == 1) // 16
              continue; // 17
       found = false; // 18
       for(divisor = 2; !found && divisor <= sqrt(number); divisor++){
// 19
              if(number % divisor == 0) // 20
                     found = true; // 21
       } // 22
       if(!found){ // 23
              copy = number; // 24
              sum = 0; // 25
              while(copy != 0) { // 26
                     digit = copy % 10; // 27
                     copy = copy / 10; // 28
                     sum += digit; // 29
               } // 30
               if(sum == s) // 31
                     primes.add(number); // 32
        } // 33
} // 34
return primes; // 35
```

Se transformă programul într-un graf orientat pe baza numerotării precedente:




Dacă adăugăm arce de la nodurile 2, 4, 6, 8 și 35 la nodul 1, numărul de noduri n este 27 și numărul de arce e devine 38. Numărul de circuite linear independente este V(G) = e – n + 1 = 38 - 27 + 1 = 12. Circuitele independente sunt:

1) 1, 2, 1
2) 1, 3, 4, 1
3) 1, 3, 5, 6, 1

4) 1, 3, 5, 7, 8, 1
5) 1, 3, 5, 7, 9...14, 15, 35, 1
6) 15, 16, 17, 15
7) 15, 16, 18, 19, 23, 24...25, 26, 31, 32, 33, 34, 15
8) 15, 16, 18, 19, 23, 24...25, 26, 31, 33, 34, 15
9) 15, 16, 18, 19, 23, 34, 15
10) 19, 20, 21, 22, 19
11) 19, 20, 22, 19
12) 26, 27...29, 30, 26

Pentru a testa aceste circuite sunt suficiente teste pentru următoarele date de intrare:

- k < 0 pentru a acoperi circuitul 1)

```
@Test
void kLessThanZero() {
    try {
        resultList = Main.findPrimes(-1, 1, 1000, 5);
        Assertions.fail("K should be negative.");
    } catch (IllegalArgumentException e) {
        Assertions.assertEquals("K is negative.", e.getMessage());
    }
}
```

- s < 0 pentru a acoperi circuitul 2)

```
@Test
void sLessThanZero() {
    try {
        resultList = Main.findPrimes(5, 1, 1000, -5);
        Assertions.fail("S should be negative.");
    } catch (IllegalArgumentException e) {
        Assertions.assertEquals("S is negative.", e.getMessage());
    }
}
```

- a < 0 sau b < 0 pentru a acoperi circuitul 3)

```
@Test
void aOrBLessThanZero() {
    try {
        resultList = Main.findPrimes(5, -1, -1000, 5);
        Assertions.fail("Range should be negative.");
    } catch (IllegalArgumentException e) {
        Assertions.assertEquals("Range is negative.", e.getMessage());
    }
}
```

- a > b pentru a acoperi circuitul 4)

```
@Test
void aGreaterThanB() {
    try {
```

```
        resultList = Main.findPrimes(5, 1000, 1, 5);
        Assertions.fail("Range should be reversed.");
    } catch (IllegalArgumentException e) {
        Assertions.assertEquals("Range is reversed.", e.getMessage());
    }
}
```

- date de intrare pentru care k > 0 și să existe în intervalul [a, b] numărul 0 sau 1, numere prime cu suma egală sau diferită de s, cât și numere care nu sunt prime pentru a acoperi circuitele 5), 6), 7), 8), 9), 10), 11), 12)

```
@Test
void findPrimesWithoutException() {
    try {
        List<Integer> expectedList = new ArrayList<>(List.of(5));
        Assertions.assertEquals(expectedList,   Main.findPrimes(1,   1,
500, 5));
    } catch (IllegalArgumentException e) {
        Assertions.fail("Exception: " + e.getMessage());
    }
}
```

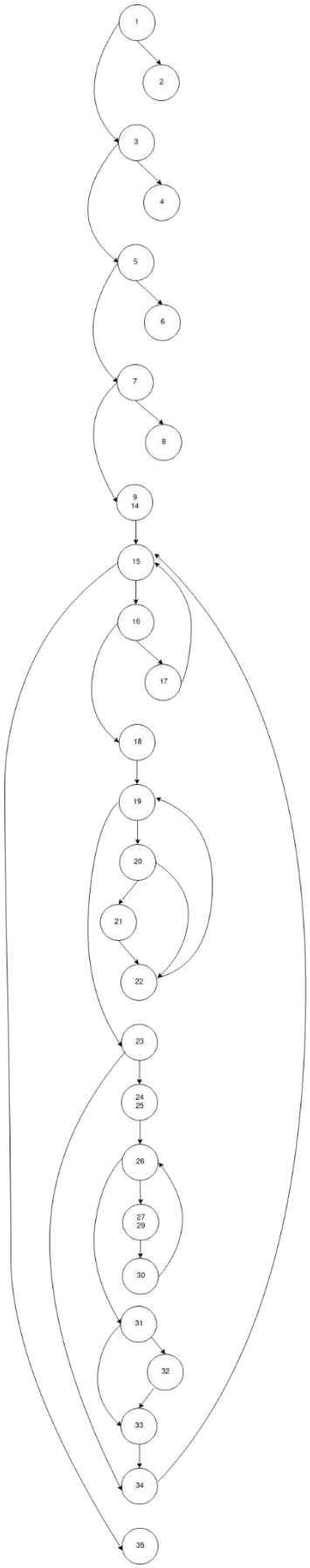
### h) Testare la nivel de cale

Avem numerotate liniile astfel, pentru a reprezenta nodurile grafului:

```
if ( k < 0 ) // 1
       throw new IllegalArgumentException("K is negative."); // 2
if ( s < 0 ) // 3
       throw new IllegalArgumentException("S is negative."); // 4
if ( a < 0 || b < 0) // 5
       throw new IllegalArgumentException("Range is negative."); // 6
if ( a > b ) // 7
       throw new IllegalArgumentException("Range is reversed."); // 8

List<Integer> primes = new ArrayList<>(); // 9
int copy; // 10
int digit; // 11
int sum; // 12
boolean found; // 13
int number, divisor; // 14

for(number = a; primes.size() < k && number <= b; number++){ // 15
       if(number == 0 || number == 1) // 16
              continue; // 17
       found = false; // 18
       for(divisor = 2; !found && divisor <= sqrt(number); divisor++){
// 19
              if(number % divisor == 0) // 20
                     found = true; // 21
       } // 22
       if(!found){ // 23
              copy = number; // 24
              sum = 0; // 25
              while(copy != 0) { // 26
                     digit = copy % 10; // 27
                     copy = copy / 10; // 28
```

```
                    sum += digit; // 29
                } // 30
                if(sum == s) // 31
                    primes.add(number); // 32
        } // 33
} // 34
return primes; // 35
```

Pe baza acestei notari, rezulta graful urmator:



Expresia regulata obtinuta:

9.15.(16.((null+17).15)*.18.19.(20.((21+null).22.19)*.23.((34.15)*+24.26.(27.30.26)*.31.(32.33 +33).34.15)*.35

Pentru n=0 si n=1 avem:

1.1.(1.((1 + 1).1 + 1).1.1.(1.((1+ 1).1.1 + 1).1.((1.1 + 1) + 1.1.(1.1.1 + 1).1.(1.1+1).1.1 + 1).1

= (2+1) * (2+1) * (1+1) * (1+1) * (1+1) = 3 * 3 * 2 * 2 * 2 = 72 cai

**i) LCSAJ Coverage**

| **LCSAJ** | **Start** | **End** | **Jump to** |
|---|---|---|---|
| 1 | 1 | 1 | EXIT |
| 2 | 1 | 3 | EXIT |
| 3 | 1 | 5 | EXIT |
| 4 | 1 | 7 | EXIT |
| 5 | 1 | 15 | 35 |
| 6 | 1 | 17 | 15 |
| 7 | 1 | 19 | 23 |
| 8 | 1 | 20 | 19 |
| 9 | 1 | 22 | 19 |
| 10 | 15 | 15 | 35 |
| 11 | 15 | 17 | 15 |
| 12 | 15 | 19 | 23 |
| 13 | 15 | 20 | 19 |
| 14 | 15 | 22 | 19 |
| 15 | 19 | 19 | 23 |
| 16 | 19 | 22 | 19 |
| 17 | 19 | 20 | 19 |
| 18 | 23 | 34 | 15 |
| 19 | 23 | 30 | 26 |
| 20 | 26 | 30 | 26 |

| | | | |
|---|---|---|---|
| 21 | 26 | 26 | 31 |
| 22 | 31 | 34 | 15 |
| 23 | 35 | 35 | EXIT |

Considerăm T = {$t_1$, $t_2$, $t_3$, $t_4$, $t_5$, $t_6$, $t_7$, $t_8$, $t_9$}, unde:

- $t_1$ = (k = -1, a = 1, b = 500, s = 5)

$t_1$: (1, 1, EXIT); $t_1$ acoperă LCSAJ 1

- $t_2$ = (k = 5, a = 1, b = 500, s = -5)

$t_2$: (1, 3, EXIT); $t_2$ acoperă LCSAJ 2

- $t_3$ = (k = 5, a = -1, b = 500, s = 5)

$t_3$: (1, 5, EXIT); $t_3$ acoperă LCSAJ 3

- $t_4$ = (k = 5, a = 500, b = 1, s = 5)

$t_4$: (1, 7, EXIT); $t_4$ acoperă LCSAJ 4

- $t_5$ = (k = 0, a = 1, b = 500, s = 5)

$t_5$: (1, 15, 35) -> (35, 35, EXIT); $t_5$ acoperă LCSAJ 5, 23

- $t_6$ = (k = 1, a = 2, b = 500, s = 5)

$t_6$: (1, 19, 23) -> (23, 30, 26) -> (26, 26, 31) -> (31, 34, 15) -> (15, 19, 23) -> (23, 30, 26) -> (26, 26, 31) -> (31, 34, 15) -> (15, 22, 19) -> (19, 19, 23) -> (23, 34, 15) -> (15, 20, 19) -> (19, 19, 23) -> (23, 30, 26) -> (26, 26, 31) -> (31, 34, 15) -> (15, 15, 35) -> (35, 35, EXIT); $t_6$ acoperă LCSAJ 7, 10, 12, 13, 14, 15, 18, 19, 21, 22, 23

- $t_7$ = (k = 1, a = 5, b = 500, s = 5)

$t_7$: (1, 20, 19) -> (19, 19, 23) -> (23, 30, 26) -> (26, 26, 31) -> (31, 34, 15) -> (15, 15, 35) -> (35, 35, EXIT); $t_7$ acoperă LCSAJ 8, 10, 15, 19, 21, 22, 23

- $t_8$ = (k = 1, a = 18, b = 18, s = 5)

$t_8$: (1, 22, 19) -> (19, 22, 19) -> (19, 20, 19) -> (19, 19, 23) -> (23, 30, 26) -> (26, 30, 26) -> (26, 26, 31) -> (31, 34, 15) -> (15, 15, 35) -> (35, 35, EXIT); $t_8$ acoperă LCSAJ 9, 10, 15, 16, 17, 19, 20, 21, 22, 23

- $t_9$ = (k = 1, a = 0, b = 1, s = 5)

$t_9$: (1, 17, 15) -> (15, 17, 15) -> (15, 15, 35) -> (35, 35, EXIT); $t_9$ acoperă LCSAJ 6, 10, 11, 23

T acoperă toate cele 23 LCSAJ. Testele corespunzătoare lui T sunt:

```
@Test
void t1Test(){
    try {
        resultList = Main.findPrimes(-1, 1, 500, 5);
        Assertions.fail("K should be negative.");
    } catch (IllegalArgumentException e) {
        Assertions.assertEquals("K is negative.", e.getMessage());
    }
}
@Test
void t2Test(){
    try {
        resultList = Main.findPrimes(5, 1, 500, -5);
        Assertions.fail("S should be negative.");
    } catch (IllegalArgumentException e) {
        Assertions.assertEquals("S is negative.", e.getMessage());
    }
}
@Test
void t3Test(){
    try {
        resultList = Main.findPrimes(5, -1, 500, 5);
        Assertions.fail("Range should be negative.");
    } catch (IllegalArgumentException e) {
        Assertions.assertEquals("Range is negative.", e.getMessage());
    }
}
@Test
void t4Test(){
    try {
        resultList = Main.findPrimes(5, 500, 1, 5);
        Assertions.fail("Range should be reversed.");
    } catch (IllegalArgumentException e) {
        Assertions.assertEquals("Range is reversed.", e.getMessage());
    }
}
@Test
void t5Test(){
    try {
        expectedList = new ArrayList<>();
        Assertions.assertEquals(expectedList,   Main.findPrimes(0,   1,
500, 5));
    } catch (IllegalArgumentException e) {
        Assertions.fail("Exception: " + e.getMessage());
    }
}
@Test
void t6Test(){
    try {
```

```
        expectedList = new ArrayList<>(List.of(5));
        Assertions.assertEquals(expectedList,   Main.findPrimes(1,   2,
500, 5));
    } catch (IllegalArgumentException e) {
        Assertions.fail("Exception: " + e.getMessage());
    }
}
@Test
void t7Test(){
    try {
        expectedList = new ArrayList<>(List.of(5));
        Assertions.assertEquals(expectedList,   Main.findPrimes(1,   5,
500, 5));
    } catch (IllegalArgumentException e) {
        Assertions.fail("Exception: " + e.getMessage());
    }
}
@Test
void t8Test(){
    try {
        expectedList = new ArrayList<>();
        Assertions.assertEquals(expectedList,  Main.findPrimes(1,  18,
18, 5));
    } catch (IllegalArgumentException e) {
        Assertions.fail("Exception: " + e.getMessage());
    }
}
@Test
void t9Test(){
    try {
        expectedList = new ArrayList<>();
        Assertions.assertEquals(expectedList, Main.findPrimes(1, 0, 1,
5));
    } catch (IllegalArgumentException e) {
        Assertions.fail("Exception: " + e.getMessage());
    }
}
```

## 3. Testarea Mutanților

Pentru testarea mutanților am folosit plugin-ul “PIT” din IDE-ul IDEA IntelliJ.

După rularea testelor cu acest plugin am obținut următoarele rezultate:

# Pit Test Coverage Report

### Project Summary

| Number of Classes | Line Coverage | | Mutation Coverage | | Test Strength | |
|---|---|---|---|---|---|---|
| 1 | 95% | 19/20 | 100% | 30/30 | 100% | 30/30 |

### Breakdown by Package

| Name | Number of Classes | Line Coverage | | Mutation Coverage | | Test Strength | |
|---|---|---|---|---|---|---|---|
| cod | 1 | 95% | 19/20 | 100% | 30/30 | 100% | 30/30 |

---

Toți mutanții au fost “omorâți”, deci nu există nimic de analizat.