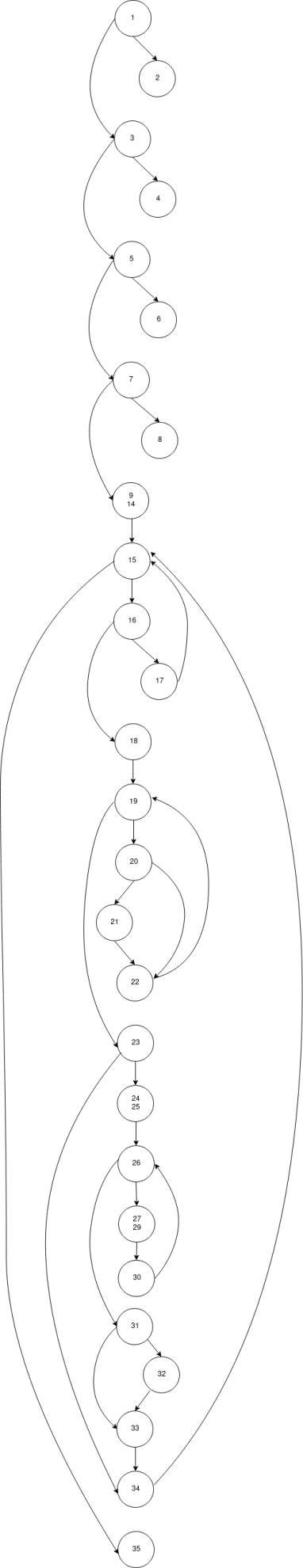
**Testarea circuitelor independente**

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:



În cadrul grafului numărul de noduri n este 27, numărul de arce este 33. Dacă adăugăm arce de la 2, 4, 6, 8 și 35 la 1, numărul de circuite linear independente este V(G) = e – n + 1 = (33 + 5) - 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

Testarea circuitelor independente poate fi realizată prin următoarele teste:

|  |
| --- |
| **public class IndependentCircuitTesting {**  **private List<Integer> resultList;**  **@Test**  **void kLessThanZero() {**  **// circuit covered: 1, 2, 1**  **try {**  **resultList = Main.findPrimes(-1, 1, 1000, 5);**  **Assertions.fail("K should be negative.");**  **}**  **catch (IllegalArgumentException e) {**  **Assertions.assertEquals("K is negative.", e.getMessage());**  **}**  **}**  **@Test**  **void sLessThanZero() {**  **// circuit covered: 1, 3, 4, 1**  **try {**  **resultList = Main.findPrimes(5, 1, 1000, -5);**  **Assertions.fail("S should be negative.");**  **}**  **catch (IllegalArgumentException e) {**  **Assertions.assertEquals("S is negative.", e.getMessage());**  **}**  **}**  **@Test**  **void aOrBLessThanZero() {**  **// circuit covered: 1, 3, 5, 6, 1**  **try {**  **resultList = Main.findPrimes(5, -1, -1000, 5);**  **Assertions.fail("Range should be negative.");**  **}**  **catch (IllegalArgumentException e) {**  **Assertions.assertEquals("Range is negative.", e.getMessage());**  **}**  **}**  **@Test**  **void aGreaterThanB() {**  **// circuit covered: 1, 3, 5, 7, 8, 1**  **try {**  **resultList = Main.findPrimes(5, 1000, 1, 5);**  **Assertions.fail("Range should be reversed.");**  **}**  **catch (IllegalArgumentException e) {**  **Assertions.assertEquals("Range is reversed.", e.getMessage());**  **}**  **}**  **@Test**  **void findPrimesWithoutException() {**  **// circuit covered from the beginning:**  **// 1, 3, 5, 7, 9...14, 15, 35, 1**  **// for number = 1 the covered circuit is:**  **// 15, 16, 17, 15**  **// for number = 2 or 3 the covered circuits are:**  **// 15, 16, 18, 19, 23, 24...25, 26, 31, 33, 34, 15**  **// 26, 27...29, 30, 26**  **// for number = 4 the covered circuits are:**  **// 15, 16, 18, 19, 23, 34, 15**  **// 19, 20, 21, 22, 19**  **// for number = 5 the covered circuits are:**  **// 15, 16, 18, 19, 23, 24...25, 26, 31, 32, 33, 34, 15**  **// 19, 20, 22, 19**  **// 26, 27...29, 30, 26**  **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());**  **}**  **}**  **}** |

**Linear Code Sequence and Jump (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 = {, , , , , , , , }, unde:

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

: (1, 1, EXIT)

acoperă LCSAJ 1

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

: (1, 3, EXIT)

acoperă LCSAJ 2

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

: (1, 5, EXIT)

acoperă LCSAJ 3

* = (k = 5, a = 500, b = 1, s = 5)

: (1, 7, EXIT)

acoperă LCSAJ 4

* = (k = 0, a = 1, b = 500, s = 5)

: (1, 15, 35) -> (35, 35, EXIT)

acoperă LCSAJ 5, 23

* = (k = 1, a = 2, b = 500, s = 5)

: (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)

acoperă LCSAJ 7, 10, 12, 13, 14, 15, 18, 19, 21, 22, 23

* = (k = 1, a = 5, b = 500, s = 5)

: (1, 20, 19) -> (19, 19, 23) -> (23, 30, 26) -> (26, 26, 31) -> (31, 34, 15) -> (15, 15, 35) -> (35, 35, EXIT)

acoperă LCSAJ 8, 10, 15, 19, 21, 22, 23

* = (k = 1, a = 18, b = 18, s = 5)

: (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)

acoperă LCSAJ 9, 10, 15, 16, 17, 19, 20, 21, 22, 23

* = (k = 1, a = 0, b = 1, s = 5)

: (1, 17, 15) -> (15, 17, 15) -> (15, 15, 35) -> (35, 35, EXIT)

acoperă LCSAJ 6, 10, 11, 23

T acoperă toate cele 23 LCSAJ.

Testele corespunzătoare lui T sunt:

|  |
| --- |
| **public class LCSAJCoverage {**  **private List<Integer> expectedList;**  **private List<Integer> resultList;**  **@Test**  **void t1Test(){**  **// t1 covers LCSAJ 1**  **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(){**  **// t2 covers LCSAJ 2**  **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(){**  **// t3 covers LCSAJ 3**  **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(){**  **// t4 covers LCSAJ 4**  **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(){**  **// t5 covers LCSAJ 5 and 23**  **try {**  **expectedList = new ArrayList<>();**  **Assertions.assertEquals(expectedList, Main.findPrimes(0, 1, 500, 5));**  **} catch (IllegalArgumentException e) {**  **Assertions.fail("Exception: " + e.getMessage());**  **}**  **}**  **@Test**  **void t6Test(){**  **// t6 covers LCSAJ 7, 10, 12, 13, 14, 15, 18, 19, 21, 22, 23**  **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(){**  **// t7 covers LCSAJ 8, 10, 15, 19, 21, 22, 23**  **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(){**  **// t8 covers LCSAJ 9, 10, 15, 16, 17, 19, 20, 21, 22, 23**  **try {**  **expectedList = new ArrayList<>();**  **Assertions.assertEquals(expectedList, Main.findPrimes(1, 18, 18, 5));**  **} catch (IllegalArgumentException e) {**  **Assertions.fail("Exception: " + e.getMessage());**  **}**  **}**  **@Test**  **void t9Test(){**  **// t9 covers LCSAJ 6, 10, 11, 23**  **try {**  **expectedList = new ArrayList<>();**  **Assertions.assertEquals(expectedList, Main.findPrimes(1, 0, 1, 5));**  **} catch (IllegalArgumentException e) {**  **Assertions.fail("Exception: " + e.getMessage());**  **}**  **}**  **}** |