UNIVERSITATEA DIN BUCURESTI

FACULTATEA DE MATEMATICA SI INFORMATICA

TESTAREA SISTEMELOR SOFTWARE

AUTOR: Spinu Valeria Gabriela

GRUPA: 451

SPECIALIZAREA: Calculatoare si Tehnologia Informatiei

PROFESOR COORDONATOR: Predut Sorina Nicoleta

2020 – 2021

1. Problema aleasa

Examen https://www.infoarena.ro/problema/examen

„Să treci un examen e ca trecutul pe zebră,

Te uiți mai întai în stânga și în dreapta.”

La un examen, N studenți sunt așezați la o masă circulară. Aceștia au de rezolvat o singură problemă. După multe calcule, fiecare a ajuns la rezultatul lui propriu. Desigur, nimeni nu e sigur dacă acest rezultat e bun sau nu. Astfel, instinctul i-a făcut pe toți să se uite în stânga și în dreapta. În loc să își treacă propriul rezultat, ei au trecut suma rezultatelor celor 2 vecini ai săi.

În ultimul minut, un val imens de conștiință i-a lovit pe toți deodată. Repede ei au realizat că nu vor ajunge nicăieri în viață dacă trișează și în final s-au hotărât să își treacă propriul lor rezultat. Problema e ca toată lumea a pierdut ciorna cu rezultatul inițial. Singura informație pe care o au la dispoziție este suma rezultatelor celor 2 vecini. Ajutați studenții să ajungă pe calea cea dreaptă prin reconstituirea rezultatelor inițiale.

Date de intrare

Fişierul de intrare examen.in va conține un număr natural N, numărul de studenți. Pe următoarea linie se află N valori, cea de-a i-a reprezentând informația studentului i.

Date de ieşire

Fişierul de ieşire examen.out va conține pe o singură linie N numere naturale reprezentând rezultatele inițiale ale celor N elevi. În cazul în care soluția nu este unică, afișați -1.

Restricţii

$4 \leq N \leq 100.000$

Rezultatele inițiale sunt numere întregi din intervalul [-1.000.000.000, 1.000.000.000]

Vecinii studentului 1 sunt 2 și N. Vecinii studentului N sunt 1 și N - 1

Se garantează că există cel puțin o configurație a rezultatelor inițiale care să satisfacă condițiile din datele de intrare.

Exemplu

examen.in

5

6 13 11 10 10

examen.out

5 4 5 9 6 1

2. Testare functionala

2.1.Partiționare de echivalență

Domeniul de intrare:

- Un numar natural N
- N numere intregi reprezentand suma rezultatelor celor 2 vecini pentru fiecare student

Domeniul de iesire:

- N numere intregi reprezentand rezultatele initiale ale fiecarui student

N trebuie sa fie un numar intreg intre 4 si 100.000. De aici distingem 3 clase de echivalenta:

- N_1 = [4, 100.000]
- N_2 = {N | N < 4}
- N_3 = {N | N > 100.000}

Fiecare rezultat trebuie sa fie un numar intreg intre –1.000.000.000 si 1.000.000.000. Din aceasta informatie distingem din nou 3 clase de echivalenta:

- R_1 = [-1.000.000.000; 1.000.000.000]
- R_2 = {r | r < -1.000.000.000}
- R_3 = {r | r > 1.000.000.000}

In ceea ce priveste output-ul, exista 2 clase de echivalenta :

- O_1 = {rezultatele corecte | solutia este unica}
- O_2 = {-1 | solutia nu este unica}

Clasele de echivalenta globale se obtin din combinatia claselor individuale. Pentru acest pas se vor obtine 8 clase de ehivalenta.

- C_111 = {(N, R, O) | N in N_1, R in R_1, O in O_1}
- C_112 = {(N, R, O) | N in N_1, R in R_1, O in O_2}
- C_121 = {(N, R, O) | N in N_1, R in R_2, O in O_1}
- C_122 = {(N, R, O) | N in N_1, R in R_2, O in O_2}
- C_131 = {(N, R, O) | N in N_1, R in R_3, O in O_1}
- C_132 = {(N, R, O) | N in N_1, R in R_3, O in O_2}
- C_2 = {(N, R, O) | N in N_2}
- C_3 = {(N, R, O) | N in N_3}

Setul de date de test pentru acest pas a fost alcatuit alegandu-se cate o valoare pentru fiecare clasa de echivalenta:

- C_111: (5, [6, 13, 11, 10, 10], sol_unica)
- C_112: (4, [4, 5, 9, 6], sol_neunica)
- C_121: (5, [-1.000.000.002, -1.000.000.002, -1.000.000.002, -1.000.000.002, -1.000.000.002], sol_unica)
- C_122: (4, [-1.000.000.008, -1.000.000.002, -1.000.000.008, -1.000.000.002], sol_neunica)
- C_131: (5, [6, 1.000.000.006, 3, 9, 1.000.000.002], sol_unica)
- C_132: (4, [3, 1.000.000.006, 3, 1.000.000.006], sol_neunica)
- C_2: (0, _, _)
- C_3: (100.005, _, _)

Intrari		**Rezultat afisat (Expected)**
N	**Rezultate**	

5	[6, 13, 11, 10, 10]	[4, 5, 9, 6, 1], solutia este unica
4	[4, 5, 9, 6]	[-1], solutia nu este unica
5	[-1.000.000.002, -1.000.000.002, -1.000.000.002, -1.000.000.002, -1.000.000.002]	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
4	[-1.000.000.008, -1.000.000.002, -1.000.000.008, -1.000.000.002]	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
5	[1.000.000.004, 1.000.000.006, 1.000.000.018, 1.000.000.012, 1.000.000.002]	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
4	[1.000.000.002, 1.000.000.006, 1.000.000.002, 1.000.000.006]	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
0	null	N trebuie sa aiba o valoare din intervalul [4; 100 000]
100005	Null	N trebuie sa aiba o valoare din intervalul [4; 100 000]

```
@Test
public void equivalencePartitioning(){
    ArrayList<Integer> arrayList1 = new ArrayList<>();
    arrayList1.add(6);
    arrayList1.add(13);
    arrayList1.add(11);
    arrayList1.add(10);
    arrayList1.add(10);
```

```
    ArrayList<Integer> arrayList2 = new ArrayList<>();
    arrayList2.add(4);
    arrayList2.add(5);
    arrayList2.add(9);
    arrayList2.add(6);

    ArrayList<Integer> arrayList3 = new ArrayList<>();
    arrayList3.add(-1000000002);
    arrayList3.add(-1000000002);
    arrayList3.add(-1000000002);
    arrayList3.add(-1000000002);
    arrayList3.add(-1000000002);

    ArrayList<Integer> arrayList4 = new ArrayList<>();
    arrayList4.add(-1000000008);
    arrayList4.add(-1000000002);
    arrayList4.add(-1000000008);
    arrayList4.add(-1000000002);

    ArrayList<Integer> arrayList5 = new ArrayList<>();
    arrayList5.add(1000000004);
    arrayList5.add(1000000006);
    arrayList5.add(1000000018);
    arrayList5.add(1000000012);
    arrayList5.add(1000000002);

    ArrayList<Integer> arrayList6 = new ArrayList<>();
    arrayList6.add(1000000002);
    arrayList6.add(1000000006);
    arrayList6.add(1000000002);
    arrayList6.add(1000000006);

    assertEquals("[4, 5, 9, 6,
1]",tester.find(5,arrayList1));
    assertEquals("[-1]",tester.find(4,arrayList2));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
```

```
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(5,arrayList3));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(4,arrayList4));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(5,arrayList5));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(4,arrayList6));
    assertEquals("N trebuie sa aiba o valoare din intervalul
[4; 100 000]",tester.find(0,null));
    assertEquals("N trebuie sa aiba o valoare din intervalul
[4; 100 000]",tester.find(100005,null));

}
```

2.2.Analiza valorilor de frontieră

Valorile de frontiera pentru clasele definite la punctul anterior sunt:

Pentru N(numarul de studenti) putem avea valorile 3, 4, 100.000, 100.001:

- N_1: 4, 100.000
- N_2: 3
- N_3: 100.001

Pentru R(rezultate) putem avem valorile –1.000.000.001, -1.000.000.000, 1.000.000.000, 1.000.000.001

- R_1: -1.000.000.000, 1.000.000.000
- R_2: -1.000.000.001
- R_3: 1.000.000.001

Pentru O (outputul) putem avea drept rezultate –1 pentru o solutie care nu este unica sau vectorul de raspunsuri corecte pentru o solutie unica

- O_1: -1

- O_2: sol_unica

Pentru acest subpunct vor exista 10 date de test deoarece, capetele invervalului fiind divizibile cu 4, nu pot obtine solutii unice:

C_111: (4, [-1000000000, -1000000000, -1000000000, -1000000000], [-1]); (4, [1000000000, 1000000000, 1000000000, 1000000000], [-1]); (100000, [-1000000000, -1000000000, ..., -1000000000, -1000000000], [-1]); (100000, [1000000000, 1000000000, ..., 1000000000, 1000000000], [-1])

C_121: (4, [-1000000008, -1000000001, - 1000000008, -1000000001], [-1]); (100000, [-1000000001, -1000000001, ..., -1000000001, -1000000001], [-1]);

C_131: (4, [1000000008, 1000000001, 1000000008, 1000000001], [-1]); (100000, [1000000001, 1000000001, ..., 1000000001, 1000000001], [-1]);

C_2: (3, _, _)

C_3: (100001, _, _)

Intrari		**Rezultat afisat(expected)**
N	**Rezultate**	
3		N trebuie sa aiba o valoare din intervalul [4; 100 000]
100.001		N trebuie sa aiba o valoare din intervalul [4; 100 000]
4	[-1000000000, -1000000000, -1000000000, -1000000000]	[-1]
4	[1000000000, 1000000000, 1000000000, 1000000000]	[-1]
100.000	[-1000000000, -1000000000, ..., -1000000000, -1000000000] |R|=100.000	[-1]
100.000	[1000000000, 1000000000, ..., 1000000000, 1000000000] |R|=100.000	[-1]

4	[-1000000008, -1000000001, -1000000008, -1000000001]	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
100.000	[-1000000001, -1000000001, ..., -1000000001, -1000000001] |R|=100.000	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
4	[1000000008, 1000000001, 1000000008, 1000000001]	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
100.000	[1000000001, 1000000001, ..., 1000000001, 1000000001] |R|=100.000	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]

```
@Test
public void boundaryValueAnalysis(){

    ArrayList<Integer> arrayList1 = new ArrayList<>();
    arrayList1.add(-1000000000);
    arrayList1.add(-1000000000);
    arrayList1.add(-1000000000);
    arrayList1.add(-1000000000);

    ArrayList<Integer> arrayList2 = new ArrayList<>();
    arrayList2.add(1000000000);
    arrayList2.add(1000000000);
    arrayList2.add(1000000000);
    arrayList2.add(1000000000);

    ArrayList<Integer> arrayList3 = new ArrayList<>();
    ArrayList<Integer> arrayList4 = new ArrayList<>();
    ArrayList<Integer> arrayList5 = new ArrayList<>();
    ArrayList<Integer> arrayList6 = new ArrayList<>();
    for(Integer i=1;i<=100000;i++){
        arrayList3.add(-1000000000);
```

```
        arrayList4.add(1000000000);
        arrayList5.add(-1000000001);
        arrayList6.add(1000000001);
    }

    ArrayList<Integer> arrayList7 = new ArrayList<>();
    arrayList7.add(-1000000008);
    arrayList7.add(-1000000001);
    arrayList7.add(-1000000008);
    arrayList7.add(-1000000001);

    ArrayList<Integer> arrayList8 = new ArrayList<>();
    arrayList8.add(1000000008);
    arrayList8.add(1000000001);
    arrayList8.add(1000000008);
    arrayList8.add(1000000001);

    assertEquals("N trebuie sa aiba o valoare din intervalul
[4; 100 000]",tester.find(3,null));
    assertEquals("N trebuie sa aiba o valoare din intervalul
[4; 100 000]",tester.find(100001,null));
    assertEquals("[-1]",tester.find(4,arrayList1));
    assertEquals("[-1]",tester.find(4,arrayList2));
    assertEquals("[-1]",tester.find(100000,arrayList3));
    assertEquals("[-1]",tester.find(100000,arrayList4));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(4,arrayList7));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(100000,arrayList5));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(4,arrayList8));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
```

```
000]",tester.find(100000,arrayList6));
}
```

2.3.Partiționarea în categorii

I. Descompune specificatia in unitati: pentru aceasta problema, exista o singura specificatie

II. Identificarea parametrilor: N, R, O

III. Gasirea categoriilor:

- N trebuie sa apartina intervalului [4, 100.000]
- Rezultatele trebuie sa fie in intervalul [-1000000000, 1000000000]
- O poate fi solutie unica sau [-1] in cazul unei solutii care nu e unica

IV. Partitionarea categoriilor in alternative

- N: <0, 3, 4, 5… 99.999, 100000, 100001, 100005>
- R: <-1.000.000.005, -1.000.000.001, -1.000.000.000, -999.999.999 … 999.999.999, 1.000.000.000, 1.000.000.001, 1.000.000.005>
- O: sol_unica, [-1]

V. Specificatii de testare

- N
  1) {N | N < 3}
  2) 3
  3) 4 [ok, valoare4]
  4) 5 … 99.999 [ok, valoare intermediare]
  5) 100.000 [ok, valoare 100.000]
  6) 100.001
  7) {N | N > 100.001}
- R
  8) {R | R < -1.000.000.001}
  9) -1.000.000.001
  10) -1.000.000.000 [ok, valoare -1.000.000.000]
  11) -999.999.999 … 999.999.999 [ok, valoare medie]
  12) 1.000.000.000 [ok, valoare 1.000.000.000]
  13) 1.000.000.001
  14) {R | R > 1.000.000.001}

- O

15) [-1]

16) Sol_unica

Din specificatia de testare ar trebui sa rezulte 7 * 7 * 2 = 98 de cazuri de testare. De asemenea exista anumite cazuri de testare care nu au sens si pot fi eliminate. In urma eliminarilor, vom obtine 32 de cazuri de testare.

VI. Crearea cazurilor de testare

N1, N2, N3R1O1, N3R2O1, N3R3O1, N3R4O1, N3R5O1, N3R6O1, N3R7O1, N4R1O1, N4R1O2, N4R2O1, N4R2O2, N4R3O1, N4R3O2, N4R4O1, N4R4O2, N4R5O1, N4R5O2, N4R6O1, N4R6O2, N4R7O1, N4R7O2, N5R1O1, N5R2O1, N5R3O1, N5R4O1, N5R5O1, N5R6O1, N5R7O1, N6, N7

VII. Crearea datelor de test

Intrari		Rezultat afisat(expected)
N	R	
0	Null	N trebuie sa aiba o valoare din intervalul [4; 100 000]
3	Null	N trebuie sa aiba o valoare din intervalul [4; 100 000]
4	[-1.000.000.005, -1.000.000.005, -1.000.000.005, -1.000.000.005]	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
4	[-1.000.000.001, -1.000.000.001, -1.000.000.001, -1.000.000.001]	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
4	[-1.000.000.000, -1.000.000.000, -1.000.000.000, -1.000.000.000]	[-1]
4	[-1, 29, -1, 29]	[-1]
4	[1.000.000.000, 1.000.000.000, 1.000.000.000, 1.000.000.000]	[-1]

4	[1.000.000.001, 1.000.000.001, 1.000.000.001, 1.000.000.001]	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
4	[1.000.000.005, 1.000.000.005, 1.000.000.005, 1.000.000.005]	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
8	[-1.000.000.005, -1.000.000.005, … -1.000.000.005], |R|=8	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
5	[-1.000.000.005, -1.000.000.005, -1.000.000.005, -1.000.000.005, -1.000.000.005]	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
8	[-1.000.000.001, -1.000.000.001, -1.000.000.001, -1.000.000.001, -1.000.000.001, -1.000.000.001, -1.000.000.001, -1.000.000.001]	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
5	[-1.000.000.001, -1.000.000.001, -1.000.000.001, -1.000.000.001, -1.000.000.001]	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
8	[-1.000.000.000, -1.000.000.000, -1.000.000.000, -1.000.000.000, -1.000.000.000, -1.000.000.000, -1.000.000.000, -1.000.000.000]	[-1]
5	[-1.000.000.000, -1.000.000.000, -1.000.000.000, -1.000.000.000, -1.000.000.000]	[-500000000, -500000000, -500000000, -500000000, -500000000]

8	[5, 17, -2, 19, 5, 7 12, 5]	[-1]
5	[30, 3, 15,14,10]	[7, 12, -4, 3, 18]
8	[1.000.000.000, 1.000.000.000, 1.000.000.000, 1.000.000.000, 1.000.000.000, 1.000.000.000, 1.000.000.000, 1.000.000.000]	[-1]
5	[1.000.000.000, 1.000.000.000, 1.000.000.000, 1.000.000.000, 1.000.000.000]	[500000000, 500000000, 500000000, 500000000, 500000000]
8	[1.000.000.001, 1.000.000.001, 1.000.000.001, 1.000.000.001, 1.000.000.001, 1.000.000.001, 1.000.000.001, 1.000.000.001]	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
5	[1.000.000.001, 1.000.000.001, 1.000.000.001, 1.000.000.001, 1.000.000.001]	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
8	[1.000.000.006, 1.000.000.006, 1.000.000.006, 1.000.000.006, 1.000.000.006, 1.000.000.006, 1.000.000.006, 1.000.000.006]	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
5	[1.000.000.006, 1.000.000.006, 1.000.000.006, 1.000.000.006, 1.000.000.006]	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
100.000	[-1.000.000.005, -1.000.000.005, …, -1.000.000.005, -1.000.000.005] |R|=100.000	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
100.000	[-1.000.000.001, -1.000.000.001, …, -1.000.000.001, -1.000.000.001] |R|=100.000	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
100.000	[-1.000.000.000, -1.000.000.000, …, -	[-1]

	1.000.000.000, -1.000.000.000] |R|=100.000	
100.000	[100, 100, …, 100, 100] |R|=100.000	[-1]
100.000	[1.000.000.000, 1.000.000.000, …, 1.000.000.000, 1.000.000.000] |R|=100.000	[-1]
100.000	[1.000.000.001, 1.000.000.001, …, 1.000.000.001, 1.000.000.001] |R|=100.000	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
100.000	[1.000.000.005, 1.000.000.005, …, 1.000.000.005, 1.000.000.005] |R|=100.000	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]
100.001	Null	N trebuie sa aiba o valoare din intervalul [4; 100 000]
100.005	Null	N trebuie sa aiba o valoare din intervalul [4; 100 000]

```
@Test
public void categoryPartitioning(){
    ArrayList<Integer> arrayList1 = new ArrayList<>();
    ArrayList<Integer> arrayList2 = new ArrayList<>();
    ArrayList<Integer> arrayList3 = new ArrayList<>();
    ArrayList<Integer> arrayList4 = new ArrayList<>();
    ArrayList<Integer> arrayList5 = new ArrayList<>();
    ArrayList<Integer> arrayList6 = new ArrayList<>();
    ArrayList<Integer> arrayList7 = new ArrayList<>();
    ArrayList<Integer> arrayList8 = new ArrayList<>();
    ArrayList<Integer> arrayList9 = new ArrayList<>();
    ArrayList<Integer> arrayList10 = new ArrayList<>();
    ArrayList<Integer> arrayList11 = new ArrayList<>();
    ArrayList<Integer> arrayList12 = new ArrayList<>();
    ArrayList<Integer> arrayList13 = new ArrayList<>();
```

```
    ArrayList<Integer> arrayList14 = new ArrayList<>();
    ArrayList<Integer> arrayList15 = new ArrayList<>();
    ArrayList<Integer> arrayList16 = new ArrayList<>();
    ArrayList<Integer> arrayList17 = new ArrayList<>();
    ArrayList<Integer> arrayList18 = new ArrayList<>();
    ArrayList<Integer> arrayList19 = new ArrayList<>();
    ArrayList<Integer> arrayList20 = new ArrayList<>();
    ArrayList<Integer> arrayList21 = new ArrayList<>();
    ArrayList<Integer> arrayList22 = new ArrayList<>();
    ArrayList<Integer> arrayList23 = new ArrayList<>();
    ArrayList<Integer> arrayList24 = new ArrayList<>();
    ArrayList<Integer> arrayList25 = new ArrayList<>();
    ArrayList<Integer> arrayList26 = new ArrayList<>();
    ArrayList<Integer> arrayList27 = new ArrayList<>();
    ArrayList<Integer> arrayList28 = new ArrayList<>();


    for(Integer i= 1; i<= 4;i++){
        arrayList1.add(-1000000005);
        arrayList2.add(-1000000001);
        arrayList3.add(-1000000000);
        arrayList5.add(1000000000);
        arrayList6.add(1000000001);
        arrayList7.add(1000000005);
    }
    arrayList4.add(-1);
    arrayList4.add(29);
    arrayList4.add(-1);
    arrayList4.add(29);

    for(Integer i= 1; i<= 8;i++){
        arrayList8.add(-1000000005);
        arrayList10.add(-1000000001);
        arrayList12.add(-1000000000);
        arrayList16.add(1000000000);
        arrayList18.add(1000000001);
        arrayList20.add(1000000005);
```

```
    }
    for(Integer i= 1; i<= 5;i++){
        arrayList9.add(-1000000005);
        arrayList11.add(-1000000001);
        arrayList13.add(-1000000000);
        arrayList17.add(1000000000);
        arrayList19.add(1000000001);
        arrayList21.add(1000000005);
    }
    arrayList14.add(5);
    arrayList14.add(17);
    arrayList14.add(-2);
    arrayList14.add(19);
    arrayList14.add(5);
    arrayList14.add(7);
    arrayList14.add(12);
    arrayList14.add(5);

    arrayList15.add(30);
    arrayList15.add(3);
    arrayList15.add(15);
    arrayList15.add(14);
    arrayList15.add(10);

    for(Integer i=1;i<= 100000;i++){
        arrayList22.add(-1000000005);
        arrayList23.add(-1000000001);
        arrayList24.add(-1000000000);
        arrayList25.add(100);
        arrayList26.add(1000000000);
        arrayList27.add(1000000001);
        arrayList28.add(1000000005);
    }
    assertEquals("N trebuie sa aiba o valoare din intervalul
[4; 100 000]",tester.find(0,null));
    assertEquals("N trebuie sa aiba o valoare din intervalul
[4; 100 000]",tester.find(3,null));
```

```
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(4,arrayList1));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(4,arrayList2));
    assertEquals("[-1]",tester.find(4,arrayList3));
    assertEquals("[-1]",tester.find(4,arrayList4));
    assertEquals("[-1]",tester.find(4,arrayList5));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(4,arrayList6));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(4,arrayList7));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(8,arrayList8));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(5,arrayList9));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(8,arrayList10));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(5,arrayList11));
    assertEquals("[-1]",tester.find(8,arrayList12));
    assertEquals("[-500000000, -500000000, -500000000, -
500000000, -500000000]",tester.find(5,arrayList13));
    assertEquals("[-1]",tester.find(8,arrayList14));
    assertEquals("[7, 12, -4, 3,
18]",tester.find(5,arrayList15));
    assertEquals("[-1]",tester.find(8,arrayList16));
    assertEquals("[500000000, 500000000, 500000000,
500000000, 500000000]",tester.find(5,arrayList17));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
```

```
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(8,arrayList18));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(5,arrayList19));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(8,arrayList20));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(5,arrayList21));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(100000,arrayList22));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(100000,arrayList23));
    assertEquals("[-1]",tester.find(100000,arrayList24));
    assertEquals("[-1]",tester.find(100000,arrayList25));
    assertEquals("[-1]",tester.find(100000,arrayList26));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(100000,arrayList27));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(100000,arrayList28));
    assertEquals("N trebuie sa aiba o valoare din intervalul
[4; 100 000]",tester.find(100001,null));
    assertEquals("N trebuie sa aiba o valoare din intervalul
[4; 100 000]",tester.find(100005,null));
}
```

3. Testare structurala

```
public class MyClass {
    public static String find(Integer N, ArrayList<Integer>
results) {
        ArrayList<Integer> finalResults = new ArrayList<>();



 1        if(N>100000 || N< 4){
 2            System.out.println("N trebuie sa aiba o valoare din
intervalul [4; 100 000]");
 3            return "N trebuie sa aiba o valoare din intervalul
[4; 100 000]";
 4        }
 5        for (Integer i=0;i< N;i++){
 6            if(results.get(i) > 1000000000 || results.get(i)<-
1000000000){
 7                System.out.println("Rezultatele initiale
trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000
000]");
 8                return "Rezultatele initiale trebuie sa aiba
valori din intervalul [- 1 000 000 000; 1 000 000 000]";
 9            }
10        }



11        if (N % 4 == 0) {
12            finalResults.add(-1);
13            System.out.println("[-1]");
14            return String.valueOf(finalResults);
15        } else {
16            if (N % 4 == 1) {
17                Integer sum = 0;
18                int rez[] = new int[N];
19                for (Integer i = 0; i < N; i++) {
20                    if ((i + 1) % 4 == 1) {
21                        sum += results.get(i);
22                    }
```

```
23                      if ((i + 1) % 4 == 2) {
24                          sum += results.get(i);
25                      }
26                      if ((i + 1) % 4 == 3) {
27                          sum -= results.get(i);
28                      }
29                      if ((i + 1) % 4 == 0) {
30                          sum -= results.get(i);
31                      }
32                  }
33                  rez[0] = (sum / 2);
34                  for (Integer i = 2; i < N; i += 2) {
35                      rez[i] = results.get(i - 1) - rez[i - 2];
36                  }
37                  rez[1] = results.get(0) - rez[N - 1];
38                  for (Integer i = 3; i < N; i += 2) {
39                      rez[i] = results.get(i - 1) - rez[i - 2];
40                  }
41                  for (Integer i = 0; i < N; i++) {
42                      finalResults.add(rez[i]);
43
44                  }

45                  System.out.println("final results are:" +
finalResults);
46              }
47              else if (N % 4 == 2) {
48                  Integer sum = 0;
49                  int rez[] = new int[N];
50                  for (Integer i = 1; i < N; i += 2) {
51                      if ((i + 1) % 4 == 2) {
52                          sum += results.get(i);
53                      } else {
54                          sum -= results.get(i);
55                      }
56                  }
```

```
57                rez[0] = (sum / 2);

58                for (Integer i = 2; i < N; i += 2) {
59                    rez[i] = results.get(i - 1) - rez[i - 2];
60                }
61                sum = results.get(0);
62                for (Integer i = 2; i < N; i += 2) {
63                    if ((i + 1) % 4 == 3) {
64                        sum += results.get(i);
65                    } else {
66                        sum -= results.get(i);
67                    }
68                }
69                rez[1] = sum / 2;
70                for (Integer i = 3; i < N; i += 2) {
71                    rez[i] = results.get(i - 1) - rez[i - 2];
72                }
73                for (Integer i = 0; i < N; i++) {
74                    finalResults.add(rez[i]);
75                }

76                System.out.println("final results are:" +
finalResults);
77            }
78            else if (N % 4 == 3) {
79                Integer sum = 0;
80                int rez[] = new int[N];
81                for (Integer i = 0; i < N; i++) {
82                    if ((i + 1) % 4 == 2) {
83                        sum += results.get(i);
84                    }
85                    if ((i + 1) % 4 == 3) {
86                        sum += results.get(i);
87                    }
88                    if ((i + 1) % 4 == 0) {
89                        sum -= results.get(i);
90                    }
```

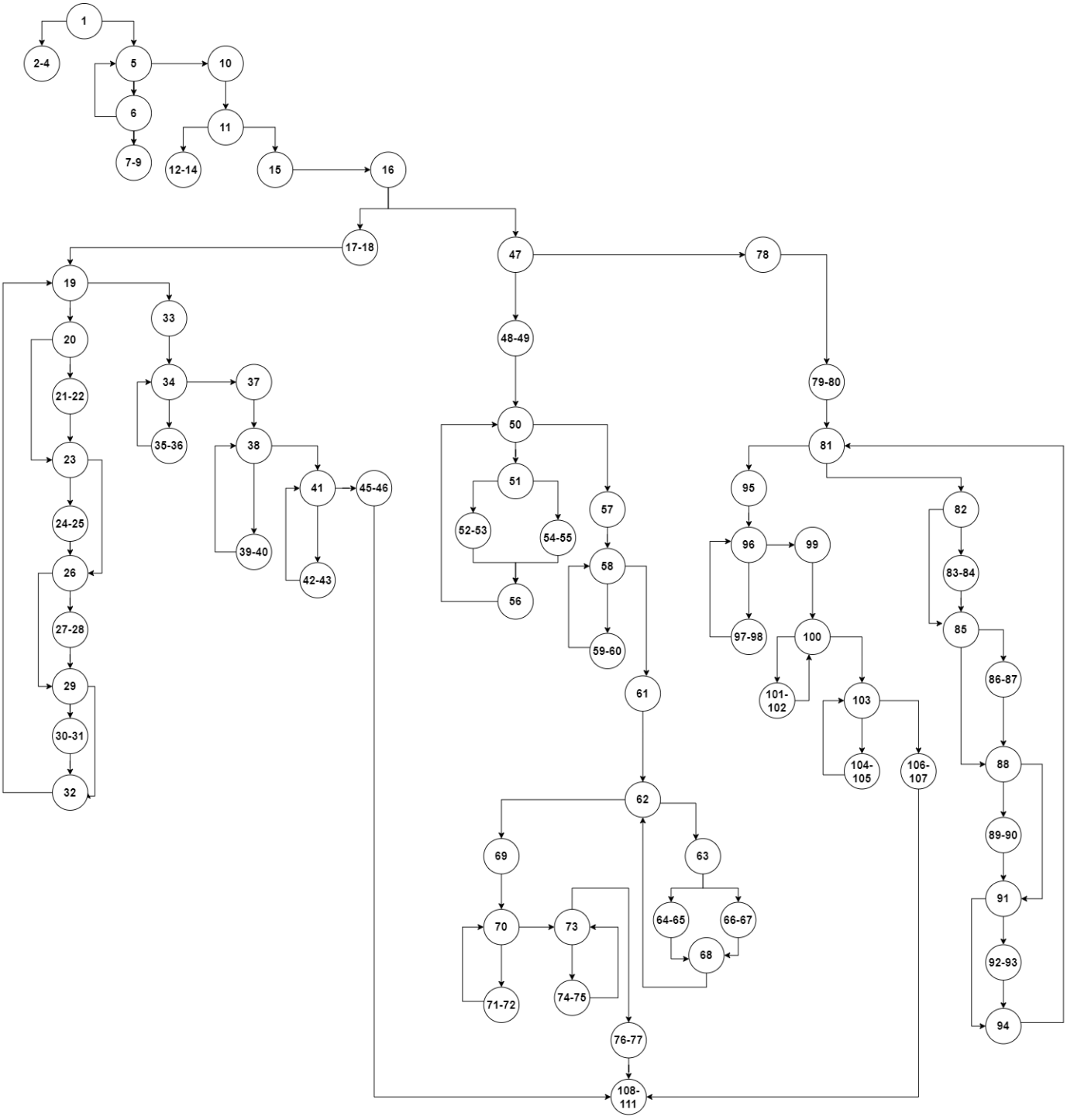
```
91                      if ((i + 1) % 4 == 1) {
92                          sum -= results.get(i);
93                      }
94                  }
95                  rez[0] = (sum / 2);
96                  for (Integer i = 2; i < N; i++) {
97                      rez[i] = results.get(i - 1) - rez[i - 2];
98                  }

99                  rez[1] = results.get(0) - rez[N - 1];
100                  for (Integer i = 3; i < N; i += 2) {
101                      rez[i] = results.get(i - 1) - rez[i - 2];
102                  }
103                  for (Integer i = 0; i < N; i++) {
104                      finalResults.add(rez[i]);
105                  }

106                  System.out.println("final results are:" +
finalResults);
107              }



108          }
109          return String.valueOf(finalResults);
110      }
111}
```


1
2-4
5
6
7-9
10
11
12-14
15
16
17-18
19
20
21-22
23
24-25
26
27-28
29
30-31
32
33
34
35-36
37
38
39-40
41
42-43
45-46
47
48-49
50
51
52-53
54-55
56
57
58
59-60
61
62
63
64-65
66-67
68
69
70
71-72
73
74-75
76-77
78
79-80
81
82
83-84
85
86-87
88
89-90
91
92-93
94
95
96
97-98
99
100
101-102
103
104-105
106-107
108-111

3.1.Statement coverage

Prin statement coverage sau testarea la nivel de instructiune ne asiguram ca toate instructiunile sunt parcurse cel putin o data.

Intrari		**Rezultat Afisat**	**Instructiuni parcurse**
N	**R**		
3	NULL	N trebuie sa aiba o valoare din intervalul [4; 100 000]	1, 2-4
4	[-1.000.000.001, -1.000.000.001, -1.000.000.001, -1.000.000.001]	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]	1, 5, 6, 7-9
4	[1,3,1,3]	[-1]	1, 5, 6, 5, 6, 5, 6, 5, 6, 5, 10, 11, 12-14
5	[30, 3, 15, 14, 10]	[7, 12, -4, 3, 18]	1,5,6,5,6,5,6,5,6,5,6,5,10, 11, 15,16, 17-18, 19, 20, 21-22, 23, 26, 29, 32, 19, 20, 23, 24-25, 26, 29, 32, 19, 20, 23, 26, 27-28, 29, 32, 19, 20, 23, 26, 29, 30-31, 32, 19, 20, 21-22, 23, 26, 29, 32, 19, 33, 34, 35-36, 34, 35-36, 34, 35-36, 34,37, 38, 39-40, 38, 39-40, 38, 41, 42-43, 41, 42-43, 41, 42-43, 41, 42-43, 41, 45-46, 108-111

6	[-3, 9, 4, 5, 9,10]	[7, -4, 2, 8, 3, 1]	1, 5, 6, 5, 6, 5, 6, 5, 6, 5, 6, 5, 6, 5, 10, 11, 15, 16, 47, 48-49, 50, 51, 52-53, 56, 50, 51, 54-55, 56, 50, 51, 54-55, 56, 50, 51, 54-55, 56, 50, 51, 52-53, 56, 50, 57, 58, 59-60, 58, 59-60, 58, 59-60, 58, 59-60, 58, 61, 62, 63, 64-65, 68, 62, 63, 66-67, 68, 62, 63, 66-67, 68, 62, 63, 66-67, 68, 62, 69, 70, 71-72, 70, 71-72, 70, 71-72, 70, 73, 74-75, 73, 74-74, 73, 74-75, 73, 74-75, 73, 74-75, 73, 74-75, 73, 76-77, 108-111
7	[18, 8, 6, 5, 3, 14, 12]	[7, 8, 1, -2, 4, 5, 10]	1, 5, 6, 5, 6, 5, 6, 5, 6, 5, 6, 5, 6, 5, 10, 11, 15, 16, 47, 78, 79-80, 81, 82, 85, 88, 91, 92-93, 94, 81, 82, 83-84, 85, 88, 91, 94, 81, 82, 85, 86-87, 88, 91, 94, 81, 82, 85, 88, 89-90, 91, 94, 81, 82, 85, 88, 91, 92-93, 94, 81, 82, 83-84, 85, 88, 91, 94, 81, 82, 85, 86-87, 88, 91, 94, 81, 95, 96, 97-98, 96, 97-98, 96, 97-98, 96, 97-98, 96, 97-98, 96, 99, 100, 101-102, 100, 101-102, 100, 101-102, 100, 101-102, 100,

			103, 104-105, 103, 104-105, 103, 104-105, 103, 104-105, 103, 104-105, 103, 104-105, 103, 104-105, 103, 106-107, 108-111

```
@Test
public void statementCoverage(){
    ArrayList<Integer> arrayList1 = new ArrayList<>();
    ArrayList<Integer> arrayList2 = new ArrayList<>();
    ArrayList<Integer> arrayList3 = new ArrayList<>();
    ArrayList<Integer> arrayList4 = new ArrayList<>();
    ArrayList<Integer> arrayList5 = new ArrayList<>();

    arrayList1.add(-1000000001);
    arrayList1.add(-1000000001);
    arrayList1.add(-1000000001);
    arrayList1.add(-1000000001);

    arrayList2.add(1);
    arrayList2.add(3);
    arrayList2.add(1);
    arrayList2.add(3);

    arrayList3.add(30);
    arrayList3.add(3);
    arrayList3.add(15);
    arrayList3.add(14);
    arrayList3.add(10);

    arrayList4.add(-3);
    arrayList4.add(9);
    arrayList4.add(4);
    arrayList4.add(5);
    arrayList4.add(9);
```

```
    arrayList4.add(10);

    arrayList5.add(18);
    arrayList5.add(8);
    arrayList5.add(6);
    arrayList5.add(5);
    arrayList5.add(3);
    arrayList5.add(14);
    arrayList5.add(12);


    assertEquals("N trebuie sa aiba o valoare din intervalul
[4; 100 000]",tester.find(3,null));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(4,arrayList1));
    assertEquals("[-1]",tester.find(4,arrayList2));
    assertEquals("[7, 12, -4, 3,
18]",tester.find(5,arrayList3));
    assertEquals("[7, -4, 2, 8, 3,
1]",tester.find(6,arrayList4));
    assertEquals("[7, 8, 1, -2, 4, 5,
10]",tester.find(7,arrayList5));
}
```

3.2.Decision coverage

Decision coverage(acoperirea la nivel de de decizie) sau branch coverage(acoperirea la nivel de ramura) genereaza date de test care testeaza cazurile cand fiecare decizie este adevarata sau falsa.

	Decizii
1	if(N>100000 || N< 4) (**1**)
2	for (Integer i=0; i< N;i++) (**5**)
3	if(results.get(i) > 1000000000 || results.get(i)<-1000000000) (**6**)

4	if (N % 4 == 0) (**7**)
5	if (N % 4 == 1) (**16**)
6	for (Integer i = 0; i < N; i++) (**19**)
7	if ((i + 1) % 4 == 1) (**20**)
8	if ((i + 1) % 4 == 2) (**23**)
9	if ((i + 1) % 4 == 3) (**26**)
10	if ((i + 1) % 4 == 0) (**29**)
11	for (Integer i = 2; i < N; i += 2) (**34**)
12	for (Integer i = 3; i < N; i += 2) (**38**)
13	for (Integer i = 0; i < N; i++) (**41**)
14	if (N % 4 == 2) (**47**)
15	for (Integer i = 1; i < N; i += 2) (**50**)
16	if ((i + 1) % 4 == 2) (**51**)
17	for (Integer i = 2; i < N; i += 2) (**58**)
18	for (Integer i = 2; i < N; i += 2) (**62**)
19	if ((i + 1) % 4 == 3) (**63**)
20	for (Integer i = 3; i < N; i += 2) (**70**)
21	for (Integer i = 0; i < N; i++) (**73**)
22	if (N % 4 == 3) (**78**)
23	for (Integer i = 0; i < N; i++) (**81**)
24	if ((i + 1) % 4 == 2) (**82**)
25	if ((i + 1) % 4 == 3) (**85**)
26	if ((i + 1) % 4 == 0) (**88**)
27	if ((i + 1) % 4 == 1) (**91**)
28	for (Integer i = 2; i < N; i++) (**96**)
29	for (Integer i = 3; i < N; i += 2) (**100**)
30	for (Integer i = 0; i < N; i++) (**103**)

Intrari		**Rezultat Afisat**	**Decizii acoperite**
N	**R**		
3	NULL	N trebuie sa aiba o valoare	1

		din intervalul [4; 100 000]	
4	[-1.000.000.001, -1.000.000.001, -1.000.000.001, -1.000.000.001]	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]	1-2-3
4	[1,3,1,3]	[-1]	1-2-3-4
5	[30, 3, 15, 14, 10]	[7, 12, -4, 3, 18]	1-2-3-4-5-6-7-8-9-10-11-12-13
6	[-3, 9, 4, 5, 9,10]	[7, -4, 2, 8, 3, 1]	1-2-3-4-5-14-15-16-17-18-19-20-21
7	[18, 8, 6, 5, 3, 14, 12]	[7, 8, 1, -2, 4, 5, 10]	1-2-3-4-5-14-22-23-24-25-26-27-28-29-30

```
@Test
public void branchCoverage() {
    ArrayList<Integer> arrayList1 = new ArrayList<>();
    ArrayList<Integer> arrayList2 = new ArrayList<>();
    ArrayList<Integer> arrayList3 = new ArrayList<>();
    ArrayList<Integer> arrayList4 = new ArrayList<>();
    ArrayList<Integer> arrayList5 = new ArrayList<>();

    arrayList1.add(-1000000001);
    arrayList1.add(-1000000001);
    arrayList1.add(-1000000001);
    arrayList1.add(-1000000001);

    arrayList2.add(1);
    arrayList2.add(3);
    arrayList2.add(1);
    arrayList2.add(3);
```

```
        arrayList3.add(30);
        arrayList3.add(3);
        arrayList3.add(15);
        arrayList3.add(14);
        arrayList3.add(10);

        arrayList4.add(-3);
        arrayList4.add(9);
        arrayList4.add(4);
        arrayList4.add(5);
        arrayList4.add(9);
        arrayList4.add(10);

        arrayList5.add(18);
        arrayList5.add(8);
        arrayList5.add(6);
        arrayList5.add(5);
        arrayList5.add(3);
        arrayList5.add(14);
        arrayList5.add(12);


        assertEquals("N trebuie sa aiba o valoare din
intervalul [4; 100 000]",tester.find(3,null));
        assertEquals("Rezultatele initiale trebuie sa aiba
valori din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(4,arrayList1));
        assertEquals("[-1]",tester.find(4,arrayList2));
        assertEquals("[7, 12, -4, 3,
18]",tester.find(5,arrayList3));
        assertEquals("[7, -4, 2, 8, 3,
1]",tester.find(6,arrayList4));
        assertEquals("[7, 8, 1, -2, 4, 5,
10]",tester.find(7,arrayList5));

    }
```

3.3.Condition coverage

Condition coverage sau acoperirea la nivel de conditie genereaza date de test astfel incat o conditie sa poata avea atat valoarea true cat si valoarea false.

	Decizii	**Conditii individuale**
1	if(N>100000 || N< 4) (**1**)	N>100000, N< 4
2	for (Integer i=0; i< N;i++) (**5**)	i< N
3	if(results.get(i) > 1000000000 || results.get(i)<-1000000000) (**6**)	results.get(i) > 1000000000, results.get(i)<-1000000000
4	if (N % 4 == 0) (**7**)	N % 4 == 0
5	if (N % 4 == 1) (**16**)	N % 4 == 1
6	for (Integer i = 0; i < N; i++) (**19**)	i< N
7	if ((i + 1) % 4 == 1) (**20**)	(i + 1) % 4 == 1
8	if ((i + 1) % 4 == 2) (**23**)	(i + 1) % 4 == 2
9	if ((i + 1) % 4 == 3) (**26**)	(i + 1) % 4 == 3
10	if ((i + 1) % 4 == 0) (**29**)	(i + 1) % 4 == 0
11	for (Integer i = 2; i < N; i += 2) (**34**)	i< N
12	for (Integer i = 3; i < N; i += 2) (**38**)	i< N
13	for (Integer i = 0; i < N; i++) (**41**)	i< N
14	if (N % 4 == 2) (**47**)	N % 4 == 2
15	for (Integer i = 1; i < N; i += 2) (**50**)	i< N
16	if ((i + 1) % 4 == 2) (**51**)	(i + 1) % 4 == 2
17	for (Integer i = 2; i < N; i += 2) (**58**)	i< N
18	for (Integer i = 2; i < N; i += 2) (**62**)	i< N
19	if ((i + 1) % 4 == 3) (**63**)	(i + 1) % 4 == 3
20	for (Integer i = 3; i < N; i += 2) (**70**)	i< N
21	for (Integer i = 0; i < N; i++) (**73**)	i< N
22	if (N % 4 == 3) (**78**)	N % 4 == 3
23	for (Integer i = 0; i < N; i++) (**81**)	i< N
24	if ((i + 1) % 4 == 2) (**82**)	(i + 1) % 4 == 2
25	if ((i + 1) % 4 == 3) (**85**)	(i + 1) % 4 == 3
26	if ((i + 1) % 4 == 0) (**88**)	(i + 1) % 4 == 0

27	if ((i + 1) % 4 == 1) (**91**)	(i + 1) % 4 == 1
28	for (Integer i = 2; i < N; i++) (**96**)	i< N
29	for (Integer i = 3; i < N; i += 2) (**100**)	i< N
30	for (Integer i = 0; i < N; i++) (**103**)	i< N

Intrari		**Rezultat Afisat**	**Decizii acoperite**
N	**R**		
3	NULL	N trebuie sa aiba o valoare din intervalul [4; 100 000]	1: N< 4
100001	NULL	N trebuie sa aiba o valoare din intervalul [4; 100 000]	1: N>100000
4	[-1.000.000.001, -1.000.000.001, -1.000.000.001, -1.000.000.001]	Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]	1: N>100000, N< 4 2: i< N 3: results.get(i)<-1000000000
4	[1.000.000.001, 1.000.000.001, 1.000.000.001, 1.000.000.001]	initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]	1: N>100000, N< 4 2: i< N 3: results.get(i) > 1000000000
4	[1,3,1,3]	[-1]	1: N>100000, N< 4 2: i< N 3: results.get(i) > 1000000000, results.get(i)<-1000000000 4: N % 4 == 0
5	[30, 3, 15, 14, 10]	[7, 12, -4, 3, 18]	1: N>100000, N< 4 2: i< N

			3: results.get(i) > 1000000000, results.get(i)<- 1000000000 4: N % 4 == 0 5: N % 4 == 1 6: i< N 7: (i + 1) % 4 == 1 8: (i + 1) % 4 == 2 9: (i + 1) % 4 == 3 10: (i + 1) % 4 == 0 11: i< N 12: i< N 13 i< N
6	[-3, 9, 4, 5, 9,10]	[7, -4, 2, 8, 3, 1]	1: N>100000, N< 4 2: i< N 3: results.get(i) > 1000000000, results.get(i)<- 1000000000 4: N % 4 == 0 5: N % 4 == 1 14: N % 4 == 2 15: i< N 16: (i + 1) % 4 == 2 17: i< N 18: i< N 19: (i + 1) % 4 == 3 20: i< N 21: i< N
7	[18, 8, 6, 5, 3, 14, 12]	[7, 8, 1, -2, 4, 5, 10]	1: N>100000, N< 4 2: i< N

			3: results.get(i) > 1000000000, results.get(i)<- 1000000000 4: N % 4 == 0 5: N % 4 == 1 14: N % 4 == 2 22: N % 4 == 3 23: i< N 24: (i + 1) % 4 == 2 25: (i + 1) % 4 == 3 26: (i + 1) % 4 == 0 27: (i + 1) % 4 == 1 28: i< N 29: i< N 30: i< N

```
@Test
public void conditionCoverage() {
    ArrayList<Integer> arrayList1 = new ArrayList<>();
    ArrayList<Integer> arrayList1b = new ArrayList<>();
    ArrayList<Integer> arrayList2 = new ArrayList<>();
    ArrayList<Integer> arrayList3 = new ArrayList<>();
    ArrayList<Integer> arrayList4 = new ArrayList<>();
    ArrayList<Integer> arrayList5 = new ArrayList<>();

    arrayList1.add(-1000000001);
    arrayList1.add(-1000000001);
    arrayList1.add(-1000000001);
    arrayList1.add(-1000000001);

    arrayList1b.add(1000000001);
    arrayList1b.add(1000000001);
    arrayList1b.add(1000000001);
    arrayList1b.add(1000000001);
```

```
    arrayList2.add(1);
    arrayList2.add(3);
    arrayList2.add(1);
    arrayList2.add(3);

    arrayList3.add(30);
    arrayList3.add(3);
    arrayList3.add(15);
    arrayList3.add(14);
    arrayList3.add(10);

    arrayList4.add(-3);
    arrayList4.add(9);
    arrayList4.add(4);
    arrayList4.add(5);
    arrayList4.add(9);
    arrayList4.add(10);

    arrayList5.add(18);
    arrayList5.add(8);
    arrayList5.add(6);
    arrayList5.add(5);
    arrayList5.add(3);
    arrayList5.add(14);
    arrayList5.add(12);

    assertEquals("N trebuie sa aiba o valoare din intervalul
[4; 100 000]",tester.find(3,null));
    assertEquals("N trebuie sa aiba o valoare din intervalul
[4; 100 000]",tester.find(100001,null));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(4,arrayList1));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(4,arrayList1b));
    assertEquals("[-1]",tester.find(4,arrayList2));
```

```
    assertEquals("[7, 12, -4, 3,
18]",tester.find(5,arrayList3));
    assertEquals("[7, -4, 2, 8, 3,
1]",tester.find(6,arrayList4));
    assertEquals("[7, 8, 1, -2, 4, 5,
10]",tester.find(7,arrayList5));

}
```

4. Complexitatea metodei – formula lui McCabe

Formula lui McCabe este : V(G)= e-n+1 unde G este un graf complet conectat, e reprezinta numarul de arce si n numarul de noduri.

N = 74

E = 86

V(G) = 86 –74 +1 = 13

Circuitele independente pentru graful de mai sus sunt:

a) 1, 2-4
b) 1, 5 ,6 ,7-9
c) 1,5,10,11,12-14
d) 1,5,10,11,15,16,17-18,19,33,34,37,38,41,45-46,108-111
e) 1,5,10,11,15,16,47,48-49,50,57,58,61,62,69,70,73,76-77,108-111
f) 1,5,10,11,15,16,47,78,79-80,81,95,96,99,100,103,106-107,108-111
g) 5,6,5
h) 50,51,52-53,56,50
i) 50,51,54-55,56,50
j) 62, 63, 64-65,68,62
k) 62, 63, 66-67,68,62
l) 19,20,23,26,29,32,19
m) 81,82,85,88,91,94,81

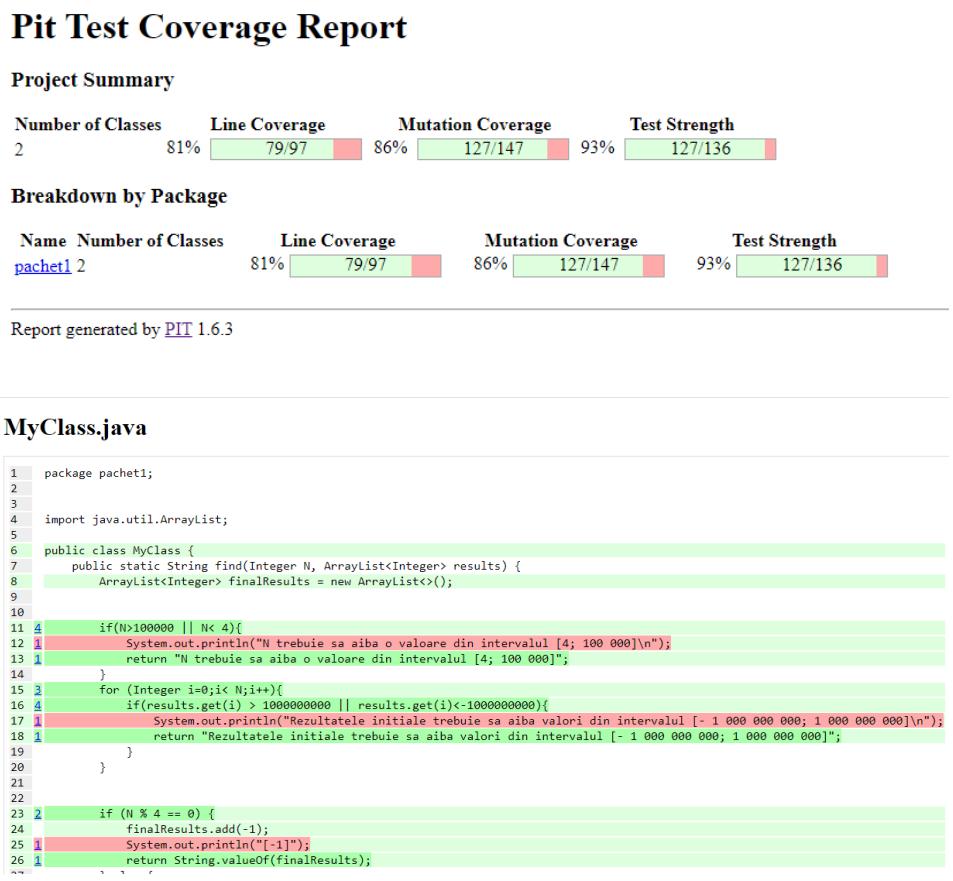
5. Acoperirea la nivel de cale

Dupa utilizarea tehnicii lui Paige si Holthouse, se obtine umatoarea expresie regulata pentru graful prezentat la punctul 3:

1.(2-4 + 5. ((6.5*+6.7-9) + 10.11(12-14 + 15.16(17-18.19.(20.(21-22.23.(24-25.26.(27-28.29(30-31.32+ 32)*+29.(30-31+32)*)+26.(27-28.29(30-31.32+ 32)*+29.(30-31+32)*)) + 23.(24-25.26.(27-28.29.(30-31.32 +32) +29.(30-31.32 +32)) +26))*+33.34(34.35-36* + 37.38(38.39-40* +41(41.42-43* + 45-46.108-111))) +47.(48-49.50(51.(52-53+54-55).56* + 57.58(59-60.58* + 61.62.(63.(64-64 + 66-67).68.62* + 69.70.(71-72.70* + 73.(74-75.73*+ 76-77.108-111)))) + 78.79-80.81(82.(83-84.85.(86-87.88.(89-90.91.(92-93.94+94) +91.(92-93.94+94)) +88.(89-90.91.(92-93.94 +94) + 91.(92-93+94))) + 85.(86-87.88. (89-90.91.(92-93.94 +94) + 91.(92-93+94))) + 88.(89-90.91.(92-93.94 +94) + 91.(92-93+94)))))* +95.96.(97-98.96* + 99.100.(101-102.100* + 103.(104-105.103*+106-107.108-111)))))))

6. Generatori de mutanti

Pentru generarea mutantilor s-a folosit generatorul Pitest. Initial, testele acopera 127 din 136 de mutanti. O mare parte din mutantii ramasi in viata sunt un produs al secventei de cod System.out.println().



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
2	81%	79/97	86%	127/147	93%	127/136

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
pachet1	2	81%	79/97	86%	127/147	93%	127/136

Report generated by PIT 1.6.3

MyClass.java

```
1  package pachet1;
2
3
4  import java.util.ArrayList;
5
6  public class MyClass {
7      public static String find(Integer N, ArrayList<Integer> results) {
8          ArrayList<Integer> finalResults = new ArrayList<>();
9
10
11 4        if(N>100000 || N< 4){
12 1            System.out.println("N trebuie sa aiba o valoare din intervalul [4; 100 000]\n");
13 1            return "N trebuie sa aiba o valoare din intervalul [4; 100 000]";
14             }
15 3        for (Integer i=0;i< N;i++){
16 4            if(results.get(i) > 1000000000 || results.get(i)<-1000000000){
17 1                System.out.println("Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]\n");
18 1                return "Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]";
19             }
20         }
21
22
23 2        if (N % 4 == 0) {
24             finalResults.add(-1);
25 1            System.out.println("[-1]");
26 1            return String.valueOf(finalResults);
```

Adaugand un nou test, prin care se doreste omorarea mutantilor generati de System.out.println(), se observa ca, de data aceasta, doar 3 mutanti supravietuiesc fata de 9 mutanti ramasi in viata inaintea adaugarii testelor.

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
2	81%	79/97	90%	133/147	98%	133/136

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
pachet1	2	81%	79/97	90%	133/147	98%	133/136

Report generated by PIT 1.6.3

MyClass.java

```
1  package pachet1;
2
3
4  import java.util.ArrayList;
5
6  public class MyClass {
7      public static String find(Integer N, ArrayList<Integer> results) {
8          ArrayList<Integer> finalResults = new ArrayList<>();
9
10
11 4        if(N>100000 || N< 4){
12 1            System.out.println("N trebuie sa aiba o valoare din intervalul [4; 100 000]");
13 1            return "N trebuie sa aiba o valoare din intervalul [4; 100 000]";
14             }
15 3        for (Integer i=0;i< N;i++){
16 4            if(results.get(i) > 1000000000 || results.get(i)<-1000000000){
17 1                System.out.println("Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]");
18 1                return "Rezultatele initiale trebuie sa aiba valori din intervalul [- 1 000 000 000; 1 000 000 000]";
19             }
20         }
21
22
23 2        if (N % 4 == 0) {
24             finalResults.add(-1);
25 1            System.out.println("[-1]");
26 1            return String.valueOf(finalResults);
```

```
@Test
public void killMutants() {
    ArrayList<Integer> arrayList1 = new ArrayList<>();
    ArrayList<Integer> arrayList2 = new ArrayList<>();
    ArrayList<Integer> arrayList3 = new ArrayList<>();
    ArrayList<Integer> arrayList4 = new ArrayList<>();
    ArrayList<Integer> arrayList5 = new ArrayList<>();

    arrayList1.add(-1000000001);
    arrayList1.add(-1000000001);
    arrayList1.add(-1000000001);
    arrayList1.add(-1000000001);

    arrayList2.add(1);
    arrayList2.add(3);
```

```
    arrayList2.add(1);
    arrayList2.add(3);

    arrayList3.add(30);
    arrayList3.add(3);
    arrayList3.add(15);
    arrayList3.add(14);
    arrayList3.add(10);

    arrayList4.add(-3);
    arrayList4.add(9);
    arrayList4.add(4);
    arrayList4.add(5);
    arrayList4.add(9);
    arrayList4.add(10);

    arrayList5.add(18);
    arrayList5.add(8);
    arrayList5.add(6);
    arrayList5.add(5);
    arrayList5.add(3);
    arrayList5.add(14);
    arrayList5.add(12);


    assertEquals("N trebuie sa aiba o valoare din intervalul
[4; 100 000]",tester.find(3,null));
    assertEquals("Rezultatele initiale trebuie sa aiba valori
din intervalul [- 1 000 000 000; 1 000 000
000]",tester.find(4,arrayList1));
    assertEquals("[-1]",tester.find(4,arrayList2));
    assertEquals("[7, 12, -4, 3,
18]",tester.find(5,arrayList3));
    assertEquals("[7, -4, 2, 8, 3,
1]",tester.find(6,arrayList4));
    assertEquals("[7, 8, 1, -2, 4, 5,
10]",tester.find(7,arrayList5));
```

```
    assertEquals("N trebuie sa aiba o valoare din intervalul
[4; 100 000]\n"+
                    "Rezultatele initiale trebuie sa aiba
valori din intervalul [- 1 000 000 000; 1 000 000 000]\n"+
                    "[-1]\n"+
                    "final results are:[7, 12, -4, 3, 18]\n"+
                    "final results are:[7, -4, 2, 8, 3,
1]\n"+
                    "final results are:[7, 8, 1, -2, 4, 5,
10]\n",

systemOutRule.getLogWithNormalizedLineSeparator());

}
```