Proiect testare software

Licenta

Cerinta problema

Ajunsă în ultimul an de facultate, Mihaela se hotărăşte să îşi termine lucrarea de licenţă intitulată„Polinomul minimal al unei rădăcini de ordin O într-o extindere algebrică de grad G". Fiindcă în lucrarea ei a obţinut rezultate legate de existenţa marţienilor, aceasta doreşte să se întâlnească cu îndrumătorul ei, Decanul.

Din păcate, timpul nu e de partea ei. Mihaela are la dispoziţie o mulţime de intervale S de dimensiune N la care poate veni în facultate.Decanul este o persoană foarte ocupată, precum Mihaela, are la îndemână o mulţime de intervale T de dimensiune M în care este la biroul său.

Mihaela are nevoie de exact K minute pentru a explica ideile prezente în lucrarea ei indrumătorului. Pentru că eroina lucrează la conjectura P = NP vă roagă să găsiţi un interval de timp de fix K secunde astfel încât ea să fie la facultate, iar profesorul ei îndrumător prezent în birou.

În caz că există mai multe soluţii de forma [X, Y], ea doreşte să afişaţi intervalul cu X minim. Dacă nu există o soluţie atunci afişaţi -1 pe prima linie.

Date de intrare (!Trimit ca argumente functiei main aceste date, pastrand structura lor)

Fişierul de intrare licenta.in conţine pe prima linie numărul natural K specificat în enunţ. Pe a doua linie este un singur număr natural N reprezentând dimensiunea mulţimei S. Următoarele N linii conţin câte 2 numere naturale separate prin câte un spaţiu: XS[i], YS[i] specificând intervalele de minute în care Mihaela este la facultate. Intervalele sunt disjuncte douăcâte două.

Următoarea linie conţine numărul natural M. Următoarele M linii conţin câte 2 numere naturale separate prin câte un spaţiu: XT[j], YT[j] specificând intervalele de minute în care profesorul este la biroul său. Intervalele sunt disjuncte două câte două.

Date de iesire (!Printez datele)

Prima linie conţine 2 numere naturale separate prin câte un spaţiu X Y, reprezentând intervalul de timp în care Mihaela se întâlneşte cu Decanul.

Restrictii (!Am ales niste restrictii care imi vor permite sa realizez testele)

$1 \leq K < 20$

$1 \leq N \leq 4$

$0 \leq XS[i] < YS[i] < 20$ oricare $1 \leq i \leq N$

$1 \leq M \leq 4$

$0 \leq XT[j] < YT[j] < 20$ oricare $1 \leq j \leq M$

Rezolvare problema

```
package pachet1;

import java.io.InvalidClassException;
import java.util.Arrays;
import java.util.Comparator;


/* Definesc clasa Interval ca o pereche numere */
class Interval {
    int x, y;

    public Interval(int x, int y){
        this.x = x;
        this.y = y;
    }
}

/* Definesc clasa SortareIntervale pentru sortarea unui array de
perechi dupa primul element */
class SortareIntervale {
    static void sorteaza(Interval a[]) {
        Arrays.sort(a, new Comparator<>() {
            @Override public int compare(Interval p1, Interval p2)
            {
                return p1.x - p2.x;
            }
        });
    }

   static int max(int a, int b) {
      if(a > b) {
         return a;
      } else {
         return b;
      }
   }
```

```
   static int min(int a, int b) {
      if(a <= b) {
         return a;
      } else {
         return b;
      }
   }
}

/*! ------------------------------------------------------------------
--------
 * Definesc clasa principala MyClass
 * @param N - numarul de intervale in care Mihaela poate venii la
facultate
 * @param M - numarul de intervale in care Decanul este la birou
 * @param K - numarul de minute de care are nevoie pentru a explica
lucrarea
 * @param Mihaela - multimea de intervale in care Mihaela poate venii
 * @param Decanul - multimea de intervale in care Decanul e la birou
 * ------------------------------------------------------------------
----- */
class Licenta {
   public int N, M, K;

    public Interval[] Mihaela;
    public Interval[] Decanul;

   // restrictii - voi alege niste restrictii cu care imi va fi mai
usor sa lucrez
   static final int MAX = 20;
    static final int MAX2 = 4;
   static final int MIN = 1;
    static final int MIN2 = 0;

   // Metoda care se ocupa de citirea si verificarea datelor
    public void Citire(String[] arg) {
        int pos = 0;

        this.K = Integer.parseInt(arg[pos]);
        pos++;

        if (this.K < MIN || this.K >= MAX) {
            throw new IllegalArgumentException("K is out of bounds");
        }

        this.N = Integer.parseInt(arg[pos]);
        pos++;

        if (this.N < MIN || this.N > MAX2) {
            throw new IllegalArgumentException("N is out of bounds");
        }
```

```
        this.Mihaela = new Interval[N];

        for (int i = 0; i < N; i++) {
            this.Mihaela[i] = new Interval(Integer.parseInt(arg[pos]),
Integer.parseInt(arg[pos+1]));

            if (this.Mihaela[i].x < MIN2 || this.Mihaela[i].x >= MAX)
                throw new IllegalArgumentException("Mihaela - x is out
of bounds");

            if (this.Mihaela[i].y < MIN2 || this.Mihaela[i].y >= MAX)
                throw new IllegalArgumentException("Mihaela - y is out
of bounds");

            if (this.Mihaela[i].x >= this.Mihaela[i].y) {
                throw new IllegalArgumentException("Mihaela - x should
be less than y");
         }

            pos+=2;
        }

        this.M = Integer.parseInt(arg[pos]);
        pos++;

        if (this.M < MIN || this.M > MAX2) {
            throw new IllegalArgumentException("M is out of bounds");
        }

        this.Decanul = new Interval[M];

        for (int i = 0; i < M; i++) {
            this.Decanul[i] = new Interval(Integer.parseInt(arg[pos]),
Integer.parseInt(arg[pos+1]));

            if (this.Decanul[i].x < MIN2 || this.Decanul[i].x >= MAX)
                throw new IllegalArgumentException("Decanul - x is out
of bounds");

         if (this.Decanul[i].y < MIN2 || this.Decanul[i].y >= MAX)
                throw new IllegalArgumentException("Decanul - y is out
of bounds");

            if (this.Decanul[i].x >= this.Decanul[i].y) {
                throw new IllegalArgumentException("Decanul - x should
be less than y");

         }

            pos+=2;
```

```
        }
    }

   // metoda se va ocupa de gasirea intervalului, iar in cazul in
   // care nu gaseste, va returna un interval (-1, -1)
    public Interval GasireInterval() {
        SortareIntervale.sorteaza(this.Mihaela);
        SortareIntervale.sorteaza(this.Decanul);

        int L = 0, R = 0;

        while (L < this.N && R < this.M) {
            int xmax = SortareIntervale.max(this.Mihaela[L].x,
this.Decanul[R].x);
            int ymin = SortareIntervale.min(this.Mihaela[L].y,
this.Decanul[R].y);

            if (ymin - xmax >= this.K) {
                return (new Interval(xmax, xmax + this.K));
            }

            if (this.Mihaela[L].y < this.Decanul[R].y) {
                L++;
         } else if (this.Mihaela[L].y > this.Decanul[R].y) {
                R++;
         } else {
                L++;
                R++;
            }
        }

        return (new Interval(-1, -1));
    }
}

public class MyClass {
    public static void main(String[] arg) {
        Licenta licenta = new Licenta();

        licenta.Citire(arg);

        Interval iesire = licenta.GasireInterval();

        if(iesire.x == -1) {
            System.out.println("Nu exista solutie");
        } else {
            System.out.println("Interval: " + iesire.x + " " +
iesire.y);
        }
    }
}
```

1. Sa se genereze date de test folosind metode functionale
   a) Partitionare in clase de echivalenta

   Exista 5 intrari:

   - Un intreg pozitiv K
   - Un intreg pozitiv N
   - N perechi de intregi pozitivi
   - Un intreg pozitiv M
   - M perechi de intregi pozitivi
   - K trebuie sa fie intre 1 si 19, deci se disting 3 clase de echivalenta:
     K_1 = 1..19
     K_2 = {K | K < 1}
     K_3 = {K | K > 19}
   - N trebuie sa fie intre 1 si 4, deci se disting 3 clase de echivalenta
     N_1 = 1..4
     N_2 = {N | N < 1}
     N_3 = {N | N > 4}
   - Urmeaza N perechi de numere, fiecare numar din fiecare pereche genereaza alte 3 clase de achivalenta. Avand perechea (XS[i], YS[i] oricare $1 \leq i \leq N$), avem pentru XS[i]:
     X_1i = 0..19
     X_2i = {XS[i] | XS[i] < 0}
     X_3i = {XS[i] | XS[i] > 19}
     Acelasi lucru se repeta pentru Y si mai apoi pentru restul perechilor. In total vor fi generate 6 * N clase de echivalenta.
   - Pentru M si urmatoarele M perechi, se vor genera clase dupa aceeasi regula si vom mai avea 3 clase de echivalenta pentru M si 6 * M pentru perechi

   Domeniul de iesiri consta in doua raspunsuri:

   - Un interval, cel in care se vor intalnii Mihaela cu Decanul
   - Un respuns care arata ca nu a fost gasit

     Acestea sunt folosite pentru a imparti domeniul de intrare in 2 clase: una pentru cazul in care pot gasii un interval de timp in care sa se intalneasca si una in care nu.

     I_1 = {exista interval}

     I_2 = {nu exista interval}

     Clasele de echivalenta pentru intregul program (glabale) se pot obtine ca o combinatie a claselor individuale Din cauza numarului claselor obtinute prin combinarea celor

individuale e atat de mare, nu ar aduce un plus notarea lor, dar le voi evidentia prin testele realizate.

```
@org.junit.Test(expected = IllegalArgumentException.class)
public void equivalencePartitioning1() {
    tester.main(new String[]{"0", null, null, null, null});
    tester.main(new String[]{"25", null, null, null, null});
    tester.main(new String[]{"4", "0", null, null, null, null});
    tester.main(new String[]{"4", "5", null, null, null, null});
    tester.main(new String[]{"4", "1", "-1", "10", null, null,
null});
    tester.main(new String[]{"4", "1", "1", "20", null, null,
null});
    tester.main(new String[]{"4", "1", "1", "10", "0", null, null});
    tester.main(new String[]{"4", "1", "1", "10", "5", null, null});
    tester.main(new String[]{"4", "1", "1", "10", "1", "-1", "4"});
    tester.main(new String[]{"4", "1", "1", "10", "1", "2", "20"});
}

@Test
public void equivalencePartitioning2() {
    tester.main(new String[]{"4", "1", "1", "10", "1", "2", "4"});
    tester.main(new String[]{"4", "1", "1", "10", "1", "5", "10"});

    assertEquals("Nu exista solutie\nInterval: 5 9\n",
systemOutRule.getLogWithNormalizedLineSeparator());
}
```

b) Analiza valorilor de frontier

Odata ce au fost identificate clasele, valorile de frontiera sunt usor de identificat:

- Valorile 0, 1, 19, 20 pentru K
- Valorile 0, 1, 4, 5 pentru N si M
- Valorile -1, 0, 19, 20 pentru XS[i] si YS[i], XT[j] si YT[j], 1 <= i <= N, 1 <= j <= M

```
@org.junit.Test(expected = IllegalArgumentException.class)
public void boundaryValueAnalysis1() {
    tester.main(new String[]{"0", null, null, null, null});
    tester.main(new String[]{"20", null, null, null, null});

    tester.main(new String[]{"1", "0", null, null, null, null});
    tester.main(new String[]{"1", "5", null, null, null, null});

    tester.main(new String[]{"1", "1", "-1", "10", null, null,
null});
    tester.main(new String[]{"1", "1", "1", "20", null, null,
null});
    tester.main(new String[]{"1", "1", "0", "19", "0", null, null});
    tester.main(new String[]{"1", "1", "0", "19", "5", null, null});
    tester.main(new String[]{"1", "1", "0", "19", "1", "-1", "4"});
```

```
    tester.main(new String[]{"1", "1", "0", "19", "1", "2", "20"});

    tester.main(new String[]{"1", "4", "-1", "10", null, null,
null});
    tester.main(new String[]{"1", "4", "1", "20", null, null,
null});
    tester.main(new String[]{"1", "4", "0", "1", "2", "5", "5", "8",
"8", "19", "0", null, null});
    tester.main(new String[]{"1", "4", "0", "1", "2", "5", "5", "8",
"8", "19", "5", null, null});
    tester.main(new String[]{"1", "4", "0", "1", "2", "5", "5", "8",
"8", "19", "1", "-1", "4"});
    tester.main(new String[]{"1", "4", "0", "1", "2", "5", "5", "8",
"8", "19", "1", "2", "20"});
    tester.main(new String[]{"1", "4", "0", "1", "2", "5", "5", "8",
"8", "19", "1", "1", "4"});

    tester.main(new String[]{"19", "0", null, null, null, null});
    tester.main(new String[]{"19", "5", null, null, null, null});
    //consider ca am atins un numar semnificativ de teste de
frontiera, iar continuarea lor ar fi pe langa scopul proiectului
}

@Test
public void boundaryValueAnalysis2() {
    tester.main(new String[]{"1", "1", "0", "19", "1", "1", "4"});
    tester.main(new String[]{"1", "4", "0", "1", "2", "5", "5", "8",
"8", "19", "1", "1", "4"});

    assertEquals("Interval: 1 2\nInterval: 2 3\n",
systemOutRule.getLogWithNormalizedLineSeparator());
}
```

c) Partitionarea in categorii

Această metodă se bazează pe cele două anterioare. Ea caută să genereze date de test care "acoperă" funcționalitatea sistemului și maximizeaza posibilitatea de găsire a erorilor.

- Descompune specificatia in unitati: avem mai multe functii utilitare pe care le apelam In functia principal
- Identifica parametrii: cei prezentati la categoriile de mai sus
- Partitioneaza fiecare categorie in alternative: vom avea optiunile vazute la categoriile anterioare
- Sa vedem niste cazuri de testare prin testele realizate:

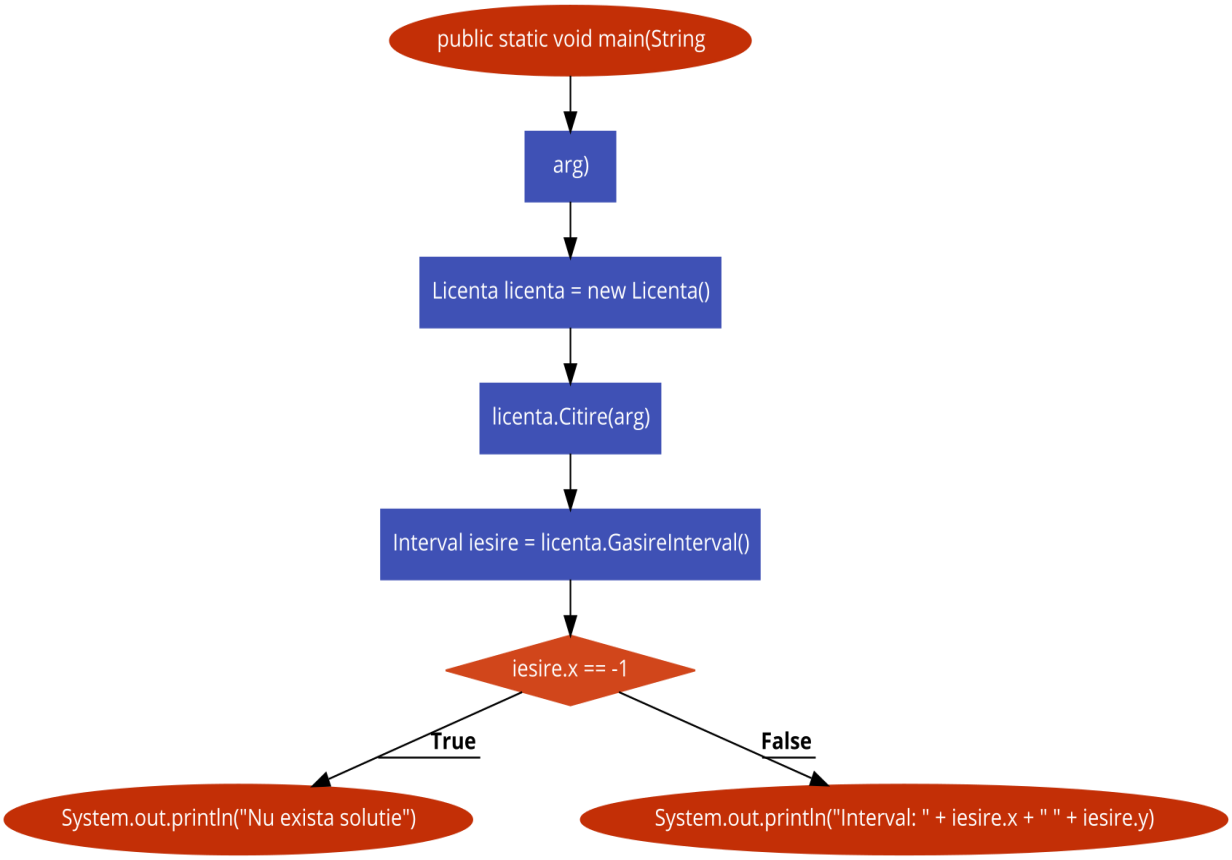
```
@org.junit.Test(expected = IllegalArgumentException.class)
public void categoryPartitioning1() {
    tester.main(new String[]{"0", null, null, null, null});
```

```
    tester.main(new String[]{"20", null, null, null, null});

    tester.main(new String[]{"1", "0", null, null, null, null});
    tester.main(new String[]{"1", "5", null, null, null, null});

    tester.main(new String[]{"1", "1", "-1", "10", null, null,
null});
    tester.main(new String[]{"1", "1", "1", "20", null, null,
null});
    tester.main(new String[]{"1", "1", "0", "19", "0", null, null});
    tester.main(new String[]{"1", "1", "0", "19", "5", null, null});
    tester.main(new String[]{"1", "1", "0", "19", "1", "-1", "4"});
    tester.main(new String[]{"1", "1", "0", "19", "1", "2", "20"});
    tester.main(new String[]{"1", "1", "0", "19", "1", "1", "4"});

    tester.main(new String[]{"1", "2", "-1", "10", null, null,
null});
    tester.main(new String[]{"1", "2", "1", "20", null, null,
null});
    tester.main(new String[]{"1", "2", "0", "8", "10", "19", "0",
null, null});
    tester.main(new String[]{"1", "2", "0", "8", "10", "19", "5",
null, null});
    tester.main(new String[]{"1", "2", "0", "8", "10", "19", "1", "-
1", "4"});
    tester.main(new String[]{"1", "2", "0", "8", "10", "19", "1",
"2", "20"});
    tester.main(new String[]{"1", "2", "0", "8", "10", "19", "1",
"1", "4"});

    tester.main(new String[]{"1", "4", "-1", "10", null, null,
null});
    tester.main(new String[]{"1", "4", "1", "20", null, null,
null});
    tester.main(new String[]{"1", "4", "0", "1", "2", "5", "5", "8",
"8", "19", "0", null, null});
    tester.main(new String[]{"1", "4", "0", "1", "2", "5", "5", "8",
"8", "19", "5", null, null});
    tester.main(new String[]{"1", "4", "0", "1", "2", "5", "5", "8",
"8", "19", "1", "-1", "4"});
    tester.main(new String[]{"1", "4", "0", "1", "2", "5", "5", "8",
"8", "19", "1", "2", "20"});
    tester.main(new String[]{"1", "4", "0", "1", "2", "5", "5", "8",
"8", "19", "1", "1", "4"});

    tester.main(new String[]{"8", "0", null, null, null, null});
    tester.main(new String[]{"8", "5", null, null, null, null});
    tester.main(new String[]{"15", "0", null, null, null, null});
    tester.main(new String[]{"15", "5", null, null, null, null});
    tester.main(new String[]{"19", "0", null, null, null, null});
    tester.main(new String[]{"19", "5", null, null, null, null});
```

```
    //consider ca am atins un numar semnificativ de categorii, iar
continuarea lor ar fi pe langa scopul proiectului
}

@Test
public void categoryPartitioning2() {
    tester.main(new String[]{"1", "1", "0", "19", "1", "1", "4"});
    tester.main(new String[]{"1", "2", "0", "8", "10", "19", "1",
"1", "4"});
    tester.main(new String[]{"1", "4", "0", "1", "2", "5", "5", "8",
"8", "19", "1", "1", "4"});

    assertEquals("Interval: 1 2\nInterval: 1 2\nInterval: 2 3\n",
systemOutRule.getLogWithNormalizedLineSeparator());
}
```

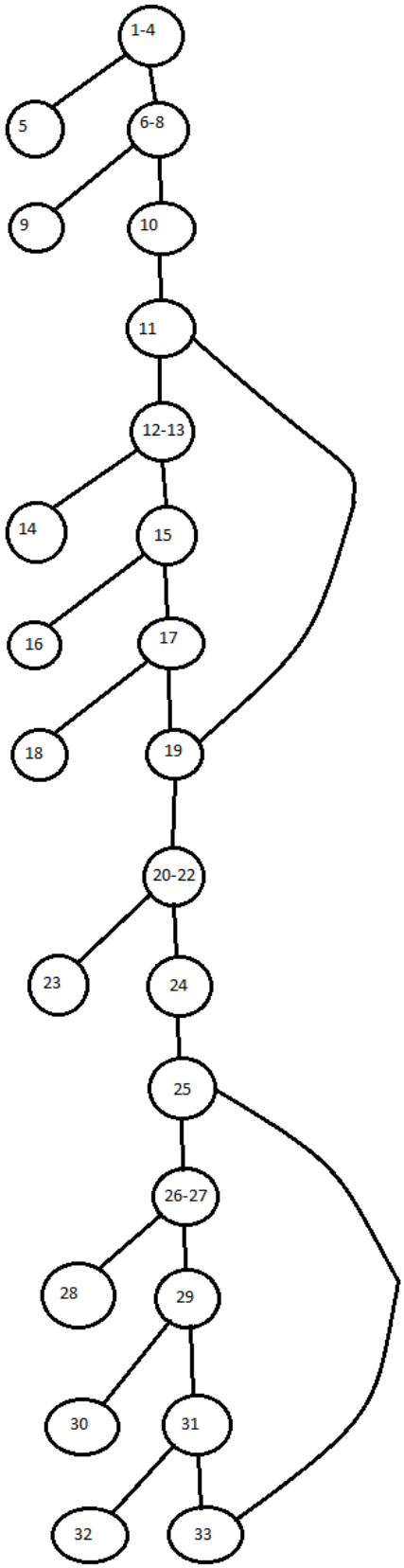
2. Sa se transforme metoda intr-un graf orientat si, pe baza acestuia, sa se genereze date de test care realizeaza acoperirea la nivel de:
   a) Instructiune
   b) Decizie
   c) Conditie

Funtia principal pe care o apelam este functia “main”, iar grafiul asociat acesteia este:

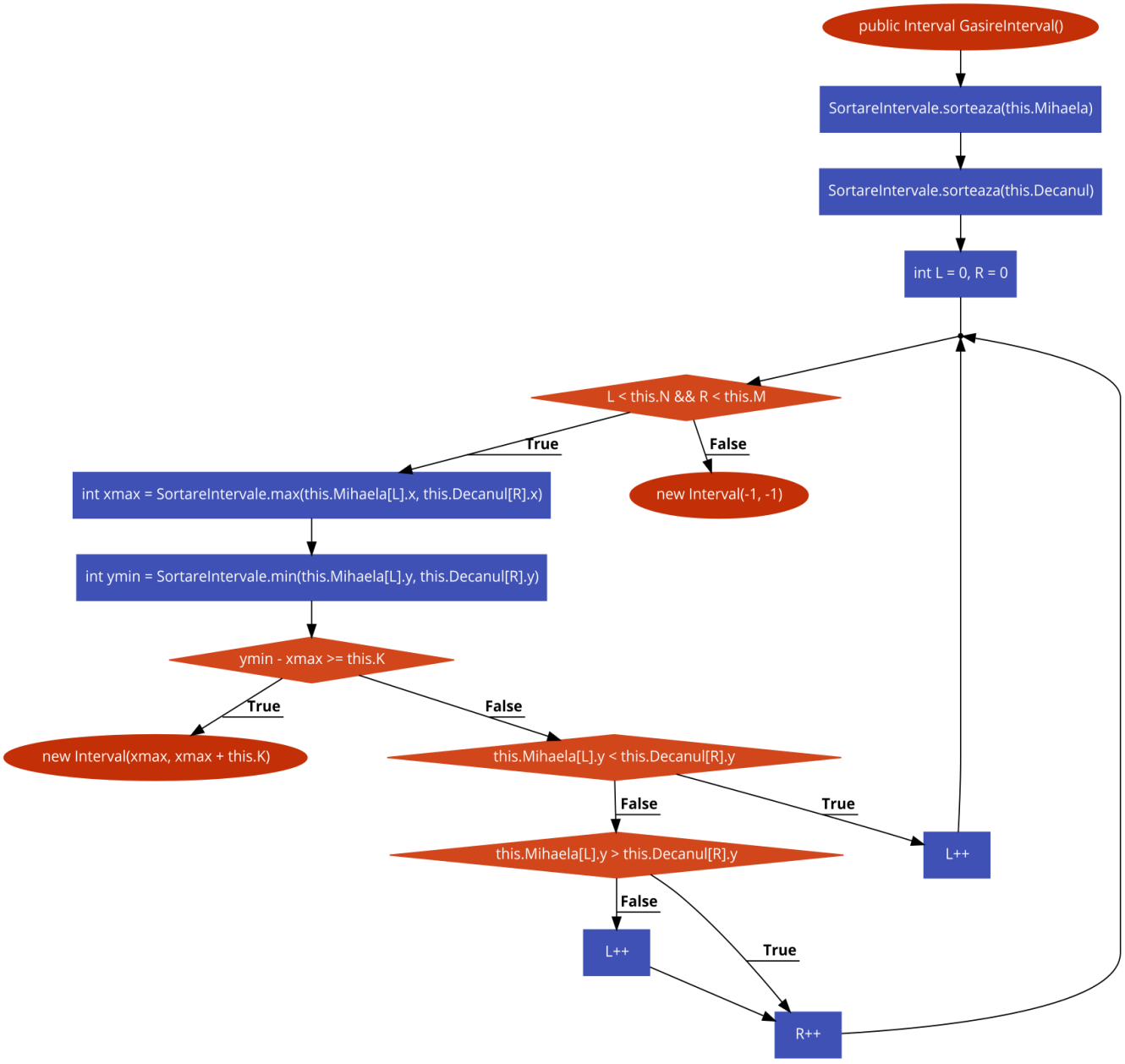


Aceasta functie apeleaza la randul iei alte doua functii importante:

- functia de citire, o functie cu multe ramificatii


1-4
5
6-8
9
10
11
12-13
14
15
16
17
18
19
20-22
23
24
25
26-27
28
29
30
31
32
33

- functia de gasire a solutiei, numita “GasireInterval”



Pe baza lor, voi scrie teste care realizeaza acoperirea la nivel de:

a. instructiune - pentru a obtine o acoperire la nivel de instructiune, trebuie sa ne concentram asupra acelor instructiuni care sunt controlate de conditii (acestea corespund ramificatilor din graf)

```
@org.junit.Test(expected = IllegalArgumentException.class)
public void statementCoverage(){
    tester.main(new String[]{"4", "2", "1", "10", "11", "13", "2",
"2", "4", "5", "12"});
    tester.main(new String[]{"4", "2", "2", "4", "5", "12", "2", "1",
"10", "11", "13"});
    tester.main(new String[]{"4", "2", "1", "4", "11", "13", "2", "2",
"4", "5", "12"});

    tester.main(new String[]{"0", null, null, null, null});
    tester.main(new String[]{"1", "0", null, null, null, null});
    tester.main(new String[]{"1", "1", "-1", "10", null, null, null});
    tester.main(new String[]{"1", "1", "1", "20", null, null, null});
    tester.main(new String[]{"1", "1", "6", "4", null, null, null});

    tester.main(new String[]{"1", "1", "0", "19", "0", null, null});
    tester.main(new String[]{"1", "1", "0", "19", "1", "-1", "4"});
    tester.main(new String[]{"1", "1", "0", "19", "1", "2", "20"});
    tester.main(new String[]{"1", "1", "0", "19", "1", "5", "3"});
}
```

b. decizie - este o extindere natural a metodei precedente si genereaza date de test care testeaza cazurile cand fiecare decizie este adevarata sau falsa

```
@org.junit.Test(expected = IllegalArgumentException.class)
public void branchCoverage() {
    tester.main(new String[]{"4", "2", "1", "10", "11", "13", "2",
"2", "4", "5", "12"});
    tester.main(new String[]{"4", "2", "2", "4", "5", "12", "2", "1",
"10", "11", "13"});
    tester.main(new String[]{"4", "2", "1", "4", "11", "13", "2", "2",
"4", "5", "12"});

    tester.main(new String[]{"0", null, null, null, null});
    tester.main(new String[]{"1", "0", null, null, null, null});
    tester.main(new String[]{"1", "1", "-1", "10", null, null, null});
    tester.main(new String[]{"1", "1", "1", "20", null, null, null});
    tester.main(new String[]{"1", "1", "6", "4", null, null, null});

    tester.main(new String[]{"1", "1", "0", "19", "0", null, null});
    tester.main(new String[]{"1", "1", "0", "19", "1", "-1", "4"});
    tester.main(new String[]{"1", "1", "0", "19", "1", "2", "20"});
    tester.main(new String[]{"1", "1", "0", "19", "1", "5", "3"});
```

```
}
```

c. conditie – genereaza date de test astfel incat fiecare conditie individuala dintr-o decizie sa ia atat valoarea adevarat cat si valoarea falst (daca acest lucru este posibil)

```
@org.junit.Test(expected = IllegalArgumentException.class)
public void conditionCoverage() {
    tester.main(new String[]{"4", "2", "1", "10", "11", "13", "2",
"2", "4", "5", "12"});
    tester.main(new String[]{"4", "2", "2", "4", "5", "12", "2", "1",
"10", "11", "13"});
    tester.main(new String[]{"4", "2", "1", "4", "11", "13", "2", "2",
"4", "5", "12"});

    tester.main(new String[]{"0", null, null, null, null});
    tester.main(new String[]{"20", null, null, null, null});

    tester.main(new String[]{"1", "0", null, null, null, null});
    tester.main(new String[]{"1", "5", null, null, null, null});

    tester.main(new String[]{"1", "1", "-1", "10", null, null, null});
    tester.main(new String[]{"1", "1", "20", "10", null, null, null});
    tester.main(new String[]{"1", "1", "1", "-1", null, null, null});
    tester.main(new String[]{"1", "1", "1", "20", null, null, null});
    tester.main(new String[]{"1", "1", "0", "19", "0", null, null});
    tester.main(new String[]{"1", "1", "0", "19", "5", null, null});
    tester.main(new String[]{"1", "1", "0", "19", "1", "-1", "4"});
    tester.main(new String[]{"1", "1", "0", "19", "1", "2", "20"});
    tester.main(new String[]{"1", "1", "0", "19", "1", "20", "4"});
    tester.main(new String[]{"1", "1", "0", "19", "1", "-1", "20"});
    tester.main(new String[]{"1", "1", "0", "19", "1", "5", "3"});
}
```

3. Complexitatea problemei folosind formula McCabe
   Dat fiind un graf complet conectat G cu e arce și n noduri, atunci numărul de circuite linear independente este dat de: V(G) = e –n + 1

   Alte formula ar fi:

   i.V(G) = e-n+2p, unde
   e = numărul de muchii ale graficului.
   n = numărul de noduri ale graficului.
   p = numărul de componente conectate.

Formulă alternativă, unde fiecare punct de ieșire este conectat înapoi la punctul de intrare(adică graful e complet conectat):
ii.V(G)=e−n+p.
Pentru un singur program (sau subrutină sau metodă), P este întotdeauna egal cu 1. Deci, o formulă mai simplă pentru osingurăsubrutină este:
iii.V(G) = e−n+2

Complexitatea ciclomatică poate fi, totuși, aplicată mai multor astfel de programe sau subprograme în același timp (de exemplu, la toate metodele dintr-o clasă), iar în aceste cazuri P va fi egal cu numărul de programe în cauză, ca fiecare subprogram. va apărea ca un subset deconectat al graficului.

Vom calcula complexitatea ciclomatica pentru cele trei metode reprezentate de grafuri, deci avem:

P = 3

E = 49

N = 47

V(G) = 8

```
@org.junit.Test(expected = IllegalArgumentException.class)
public void circuitsCoverage() {
    tester.main(new String[]{"4", "2", "1", "10", "11", "13", "2",
"2", "4", "5", "12"});
    tester.main(new String[]{"4", "2", "2", "4", "5", "12", "2", "1",
"10", "11", "13"});
    tester.main(new String[]{"4", "2", "1", "4", "11", "13", "2", "2",
"4", "5", "12"});

    tester.main(new String[]{"0", null, null, null, null});
    tester.main(new String[]{"1", "0", null, null, null, null});
    tester.main(new String[]{"1", "1", "-1", "10", null, null, null});
    tester.main(new String[]{"1", "1", "1", "20", null, null, null});
    tester.main(new String[]{"1", "1", "6", "4", null, null, null});

    tester.main(new String[]{"1", "1", "0", "19", "0", null, null});
    tester.main(new String[]{"1", "1", "0", "19", "1", "-1", "4"});
    tester.main(new String[]{"1", "1", "0", "19", "1", "2", "20"});
    tester.main(new String[]{"1", "1", "0", "19", "1", "5", "3"});
}
```

4. Expresiile regulate formata din nodurile grafurilor:
   Pentru functia principal, vom avea expresia regulate
   1.2.3.4.5.6.(7+8)

   Pentru functia de citire vom avea expresia regulate
   1.(2+3.(4+5.6.(7.(8+9.(10+11.(12+13.6)*.14.(15+16.17.(18.(19+20.(21+22.(23+24.
   17.(18.(19+20.(21+22.(23+24.17)*))))))))))))

   Pentru functia de gasire a intervalului am expresia regulate
   1.2.3.4.(5.(6.8.9.10.(11+12.(13.(15+16)+14).4)*) +7)

   Operatorul star(Kleene): R* = R de zero sau mai multe ori la rând.
   Inlocuim in expresii operatorul Kleene:

   1.(2+3.(4+5.6.(7.(8+9.(10+11.(12+13.6+null).14.(15+16.17.(18.(19+20.(21+22.(23+24.
   17.(18.(19+20.(21+22.(23+24.17+null))))))))))))

   1.2.3.4.(5.(6.8.9.10.(11+12.(13.(15+16)+14).4+null)) +7)

   Numărul de căi se obține înlocuind în expresia regulate fiecare nod cu 1(inclusive pt. null), iar operația de concatenare devine înmulțire. Pentru expresile de mai sus avem:

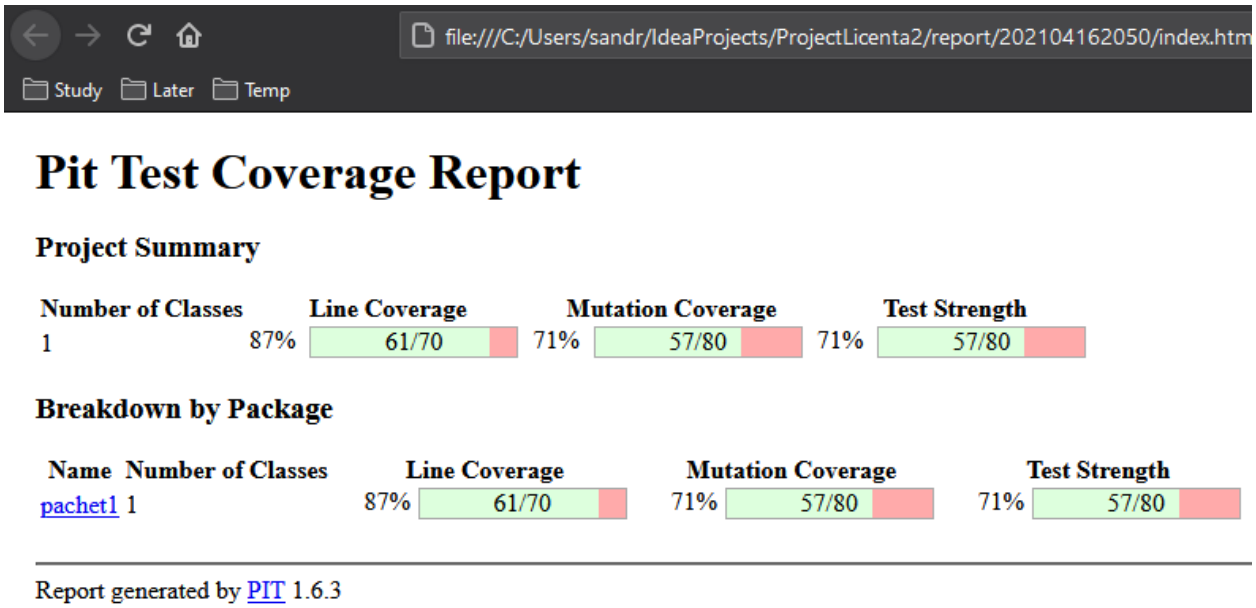
   1.1.1.1.1.1.(1+1) = 2 cai

   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+1.1+1))))))))))))) = 32 de cai

   1.1.1.1.(1.(1.1.1.1.(1+1.(1.(1+1)+1).1+1)) +1) = 6 cai

   In total avem 384 de cai.

5. Am utilizat generatorul de mutani Pitest
6. Dupa rularea seturilor de teste de la punctele prcedente, am obtinut urmatorul raport:



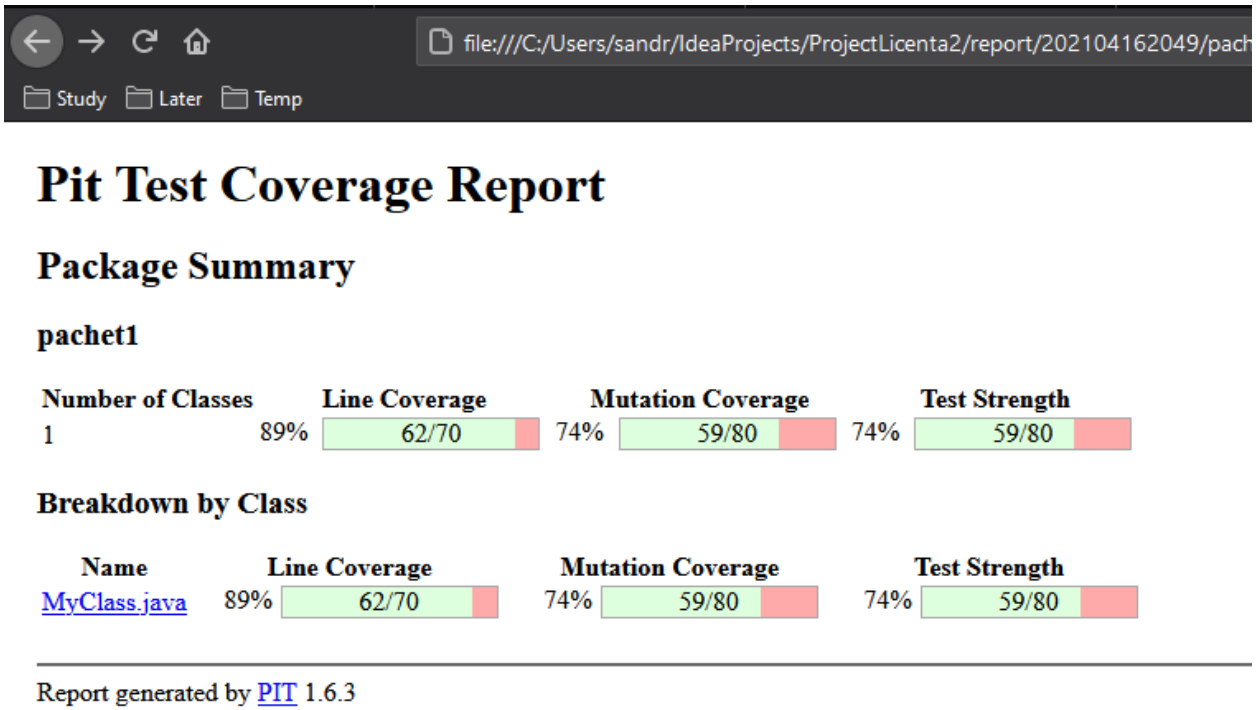
Au fost detectate 57 din 80 de mutatii, in procente insemnand 71%

Vom creea niste teste suplimentare pentru a omori 2 dintre mutantii ramasi in viata:

```
@Test
public void killSort() {
    tester.main(new String[]{"4", "2", "11", "15", "1", "10", "2", "5", "15", "2", "4"}); //PIT
returneaza % fara si % cu acest test
}

@Test
public void killMin() {
    tester.main(new String[]{"4", "2", "11", "15", "1", "10", "2", "5", "15", "2", "10"}); //PIT
returneaza % fara si % cu acest test
}

@org.junit.Test(expected = IllegalArgumentException.class)
public void kill3() {
    tester.main(new String[]{"4", "2", "11", "15", "1", "10", "2", "5", "15", "10", "2"}); //PIT
returneaza % fara si % cu acest test
}

@Test
public void kill4() {
    tester.main(new String[]{"4", "2", "1", "10", "11", "13", "2", "2", "5", "5", "12"}); //PIT
returneaza % fara si % cu acest test
}
```

```
@Test
public void kill5() {
   tester.main(new String[]{"4", "1", "1", "10", "4", "1", "3", "5", "6", "8", "9", "10", "11"});
//PIT returneaza 71% fara si 74% cu acest test
}
```

Dupa rularea si a acestor teste, mai sunt omorati 2 mutanti, iar procentul de descoperire a mutantilor este de 74%



Acesta este si link-ul catre proiect: Github