**UNIVERSITATEA DIN BUCUREŞTI**

**FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ**

**Testarea Sistemelor Software**

Proiect individual

Coordonator: lect dr. Preduț Sorina Student: Ceaușescu Ciprian Mihai

Grupa 453

**Specificațiile problemei:**

Să se implementeze un program care primește de la tastatură un număr n, 1<n<50, urmat de n numere naturale și o valoare count. Să se returneze TRUE dacă numărul de perechi aflate pe poziții consecutive din șirul dat care sunt prime între ele este egal cu count, FALSE în caz contrar.

Input:

* n = numărul de elemente al vectorului
* n numere naturale
* count = numărul de perechi pe care îl dorim în șir

Output:

* TRUE, dacă numărul de perechi prime între ele din șir este egal cu count
* FALSE, altfel

**Cerința 1:**

* 1. **Partiționare în clase de echivalență**

Domeniul de intrări:

* 1<n<50 lungimea vectorului
* Un vector v de n numere naturale
* Count

Pentru fiecare intrare obținem următoarele clase de echivalență:

* Pentru n:

1. N\_1 = {1, 2, …, 50}
2. N\_2 = {n|n<1}
3. N\_3 = {n|n>50}

* Pentru v:

1. V\_1 = {v|v are numai valori pozitive}
2. V\_2 = {v|v are cel puțin o valoare negativă}

* Pentru count:

1. Count\_1 = {count|0<=count<n}
2. Count\_2 = {count|count<0}
3. Count\_3 = {count|count>=n}

Domeniul de ieșiri constă din două răspunsuri:

* TRUE, dacă numărul de perechi prime între ele din șir este egal cu count
* FALSE, altfel

Domeniul de intrare este împărțit astfel în două clase de echivalență:

* I\_11(v) = { count |numărul de perechi prime între ele din sir este egal cu count}
* I\_12(v) = { count |numărul de perechi prime între ele din șir nu este egal cu count }

Astfel, obținem următoarele clase de echivalență globale:

* C\_1 = {(n,v,count)|n in N\_1, v in V\_1, count in Count\_1, count in I\_11(v)}
* C\_2 = {(n,v,count)|n in N\_1, v in V\_1, count in Count\_1, count in I\_12(v)}
* C\_3 = {(n,v,count)|n in N\_1, v in V\_1, count in Count\_2}
* C\_4 = {(n,v,count)|n in N\_1, v in V\_1, count in Count\_3}
* C\_5 = {(n,v,count)|n in N\_1, v in V\_2}
* C\_6 = {(n,v,count)|n in N\_2}
* C\_7 = {(n,v,count)|n in N\_3}

În total avem 7 clase de echivalență. Alegem următoarele date de test:

T1 = (5,{1,2,3,4,5},4) => TRUE

T2 = (5,{1,2,3,4,5},3) => FALSE

T3 = (5,{1,2,3,4,5},-1) => Conditii nesatisfacute

T4 = (5,{1,2,3,4,5},5) => Conditii nesatisfacute

T5 = (5,{1,-1,2,3,4},1) => Conditii nesatisfacute

T6 = (0,\_,\_) => Conditii nesatisfacute

T7 = (51,\_,\_) => Conditii nesatisfacute

* 1. **Analiza valorilor de frontieră**

Valorile de frontieră sunt:

* Dim vectorului: n=0, 1, 50, 51
* Count să se găsească între 0 și 49

Clasele de dpdv al valorilor de frontiera sunt:

* N\_1 = {(n,v,count) | n=0}
* N\_2 = {(n,v,count) | n=1}
* N\_3 = {(n,v,count) | n=50}
* N\_4 = {(n,v,count) | n=51}
* Count\_1 = {(n,v,count) | count=0 }
* Count\_2 = {(n,v,count) | count=49}

Combinand clasele dpdv al valorilor de frontiera obținem:

C\_1 = {(n,v,count) | n=0, count = 0}

C\_2 = {(n,v,count) | n=0, count = 49}

C\_3 = {(n,v,count) | n=1, count = 0}

C\_4 = {(n,v,count) | n=1, count = 49}

C\_5 = {(n,v,count) | n=50, count = 0}

C\_6 = {(n,v,count) | n=50, count = 49}

C\_7 = {(n,v,count) | n=51, count = 0}

C\_8 = {(n,v,count) | n=51, count = 49}

Alegem următoarele date de test:

T1 = (0,[],0) => Conditii nesatisfacute

T2 = (0,[],49) => Conditii nesatisfacute

T3 = (1,[1],0) => True

T4 = (1,[1],49) => Conditii nesatisfacute

T5 = (50,[3,3,….,3],0) => True

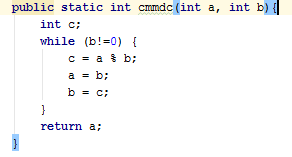
T6 = (50,[3,3,….,3],49) => False

T7 = (51,[3,3,….,3],0) => Conditii nesatisfacute

T8 = (51,[3,3,….,3],49) => Conditii nesatisfacute

* 1. **Partiționarea în categorii**

Avem următoarea partiționare în unități a problemei:



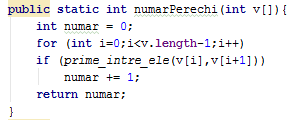
**Categorii: două numere => returneaza cmmdc a doua numere.**

Date de intrare 4,4 => 4 / 2,7 => 1



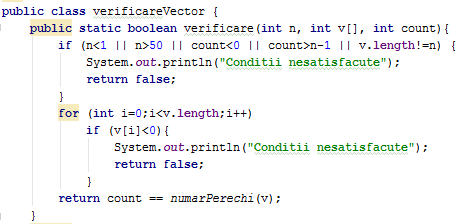
**Categorii: două numere prime între ele sau nu => returnează true sau false dacă cmmdc dintre x,y este 1, i.e sunt prime între ele.**

Date de intrare 4,4 => False / 2,7 => True



**Categorii: un vector => pentru un vector dat ca parametru returnează numărul de perechi prime între ele aflate pe poz consecutive**

Date de intrare [1,1,1] => 2 / [3,3,3] => 0



**Categorii:**

* **n: dacă se află în intervalul valid 1, 50 sau nu**
* **vectorul v, daca conține doar elem pozitive sau nu**
* **count daca este între 0 și n-1**

**Alternative:**

* **n<0, n=0, n=1, n=2, …, n=49, n=50, n>50**
* **v are doar numere naturale, sau cel puțin o val negativă**
* **count între 0 și n-1, count < 0, count>=n**

Datele de testare rezultate sunt:

T1 = (-1, [], 0) => Conditii nesatisfacute

T2 = (0, [], 0) => Conditii nesatisfacute

T3 = (1, [2], -1) => Conditii nesatisfacute

T4 = (1, [2], 0) => True

T5 = (1, [2], 2) => Conditii nesatisfacute

T6 = (1, [-1], 1) => Conditii nesatisfacute

T7 = (2, [1,2], -1) => Conditii nesatisfacute

T8 = (2, [2,2], 0) => True

T9 = (2, [2,2], 3) => Conditii nesatisfacute

T10 = (2, [-1,1], 1) => Conditii nesatisfacute

T11 = (50, [2,2,…,2], -1) => Conditii nesatisfacute

T12 = (50, [2,2,…,2], 0) => True

T13 = (50, [2,2,…,2], 51) => Conditii nesatisfacute

T14 = (50, [-2,-2,…,-2], 1) => Conditii nesatisfacute

T15 = (51,[],49) => Conditii nesatisfacute

**Cerința 2:**

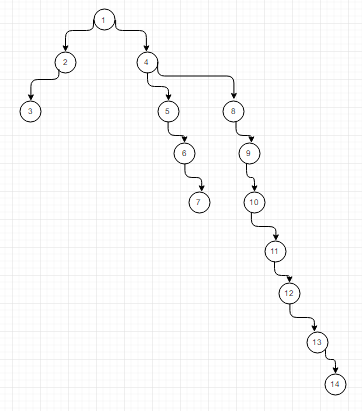
Programul în Java:

**public static boolean** verificare(**int** n, **int** v[], **int** count){  
 1 **if** (n<1 || n>50 || count<0 || count>n-1 || v.**length**!=n) {  
 2 System.***out***.println(**"Conditii nesatisfacute"**);  
 3 **return false**;  
 }  
 4 **for** (**int** i=0;i<v.**length**;i++)  
 5 **if** (v[i]<0){  
 6 System.***out***.println(**"Conditii nesatisfacute"**);  
 7 **return false**;  
 }  
 8 **return** count == *numarPerechi*(v);

**public static int** numarPerechi(**int** v[]){  
 **int** numar = 0;  
 9 **for** (**int** i=0;i<v.**length**-1;i++)  
 10 **if** (*prime\_intre\_ele*(v[i],v[i+1]))  
 numar += 1;  
 14 **return** numar;  
}

**public static boolean** prime\_intre\_ele(**int** a, **int** b){  
 11 **return** *cmmdc*(a, b) == 1;  
}

**public static int** cmmdc(**int** a, **int** b){  
 **int** c;  
 12 **while** (b!=0) {  
 c = a % b;  
 a = b;  
 b = c;  
 }  
 13 **return** a;  
}



Statement Coverage: Pentru a realiza acoperirea la nivel de instrucțiune dăm următoarele teste programului nostru:

T1 = (0,[],0) => Conditii nesatisfacute

T2 = (1,[-4],0) => Conditii nesatisfacute

T3 = (1,[2],0) => True

Branch coverage: Instrucțiuni care duc la ramuri în program

1. **if (n < 1 || n > 50 || count<0 || count>n-1 || v.length!=n)**
2. **if (v[i]<0)**

Pentru a testa acoperirea la nivelul ramurilor de instrucțiuni vom avea următoarele teste:

T1 = (0,[],0) => Conditii nesatisfacute

T2 = (1,[],-1) => Conditii nesatisfacute

T3 = (1,[1],-1) => Conditii nesatisfacute

T4 = (1,[1],0) => True

T5 = (1,[1],1) => Conditii nesatisfacute

T6 = (2,[1],0) => Conditii nesatisfacute

Condition coverage: Deciziile din programul nostru Java sunt următoarele:

1. **if (n < 1 || n > 50 || count<0 || count>n-1 || v.length!=n)**

Acestea duc la următoarele condiții individuale:

n<1

n>50

count<0

count>=n

len(v)!=n

1. **for (int i=0;i<v.length;i++)**

**if (v[i]<0):**

v[i] < 0

Pentru a acoperi toate cazurile de mai sus vom crea o nouă suită de teste unitare:

T1 = (0, [], 0) => Conditii nesatisfacute

T2 = (51, [1,1,…,1],0) => Conditii nesatisfacute

T3 = (1,[5],-1) => Conditii nesatisfacute

T4 = (1,[5],1) => Conditii nesatisfacute

T5 = (2,[5],0) => Conditii nesatisfacute

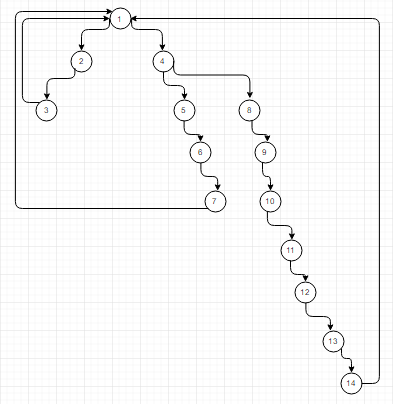
T6 = (1,[-1],0) => Conditii nesatisfacute

T7 = (1,[1],0) => True

**Cerința 3:**

Dat fiind un graf complet conectat G cu e arce si n noduri, atunci numărul de circuite linear independente este dat de: V(G) = e – n + 1

Pentru ca graful nostru prezentat mai sus sa devină unul complet conectat vom adăuga muchii de la nodul 3 către 1, de la nodul 7 către 1 și în final de la nodul 8 către nodul 1.



În concluzie V(G) = e – n + 1 = 16 – 14 + 1 = 3

Circuite independente:

* 1 -> 2 -> 3 -> 1
* 1 -> 4 -> 5 -> 6 -> 7 -> 1
* 1 -> 4 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14-> 1

**Cerința 4:**

Pentru graful pe care l-am construit mai sus avem următoarea expresie regulată:

**1.(2.3 + 4.(5.6.7 + 8.9.10.11.12.13.14))**

Numărul de căi: 3, iar căile posibile sunt următoarele:

**1.2.3**

**1.4.5.6.7**

**1.4.8.9.10.11.12.13.14**

Datele de test pentru acest căi posibile sunt următoarele:

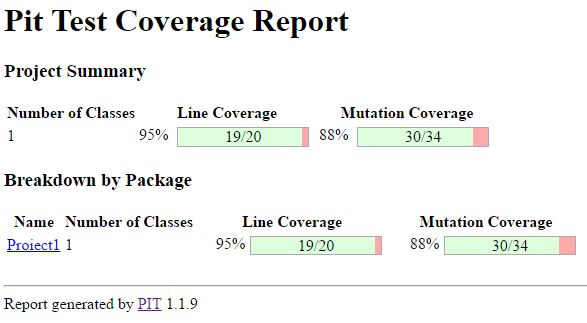
T1 = (0, [], 0) => Conditii nesatisfacute

T2 = (1, [-1], 0) => Conditii nesatisfacute

T3 = (1, [2], 0) => True

**Cerințele 5, 6, 7:**

Pentru generarea mutanților am folosit Pitest = care este un soft pentru generare de mutanți asupra codului scris în Java. Acest soft suportă modulul de testare unitară, generând astfel rapoarte asupra mutaților creați și asupra comportamentului acestora.

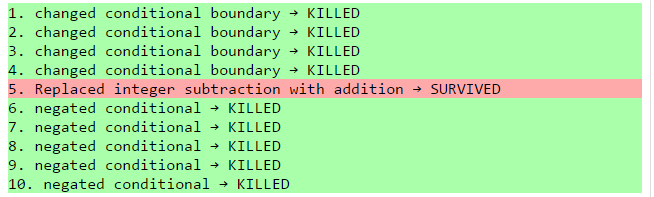


Testele pe care le-am creat programului nostru omoară 30 din cei 34 de mutanți creați de către Pitest.

Mutanții care rămân în viață sunt următorii:

1. Asupra condiției de mai jos s-au creat 10 mutanți dintre care doar unul a supraviețuit, cel care a pus condiția ca: count>n+1.





Acest mutant nu poate fi omorât pentru ca niciodată count nu poate fi mai mare decât numărul de elemente ale șirului, n. Daca ar fi mai mare, numărul de perechi prime între ele returnat de program, nu ar fi unul corect, astfel încât testele nu pot trece.

1. Asupra codului de mai jos s-a creat un mutant:





Acesta nu poate fi omorât, dar nici nu influențează programul nostru. Mesajul „Conditii nesatisfacute” poate fi sau nu afișat.

1. Asupra codului de mai jos s-au creat doi mutanți:





Pentru ca acesta să fie omorât s-a creat o suită de test suplimentară

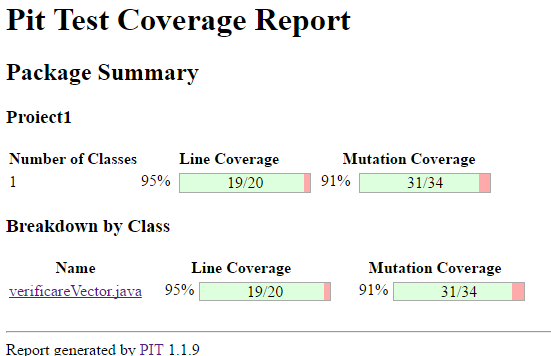
cu numele de testeSuplimentare(), în care se verifica cazul în care elementele sunt egale cu 0.

**public void** killChangedConditionalBoundary(){

*assertTrue*(*verificare*(2, **new int**[]{0,0}, 0));  
 }

1. Cel de-al patrulea mutant creat este similar cu cel de la pasul 2, care, de asemenea, nu poate fi omorât.

Noile rezultate ne arată ca s-a omorât mutantul creat la pasul 3.



Mutanții creați la punctele 2 și 4 pot fi omorâți comentând codul în care se afișează mesajele Conditii nesatisfacute. Un nou rezultat dat de Pitest ar fi următorul:

