ORICE DIN TESTARE FUNCTIONALA, STRUCTURALA, BAZATA PE MUTANTI

1. Se consideră următorul program C++, care implementează algoritmul de sortare QuickSort:

```
// program QS
void partitionare(int x[], int m, int n, int &j){
   // m<n,
//partitioneaza x[m..n]={x[m],…x[n]}; la sfarsit,
// (m <= j <= n) && (x[m..j-1] <= x[j] <= x[j+1..n])
   int pivot, t;
   j=m;pivot=x[m]; //*1
   for(int i=m+1; i<=n; i++){//*2
      if (x[i]<pivot){ // *3
         j=j+1;
         t=x[i];x[i]=x[j];x[j]=t;
      }
   }
   t=x[m];x[m]=x[j];x[j]=t;
}
```

```
void quickSort(int x[],int m, int n){
//sorteaza crescator x[m..n]={x[m],…x[n]}
   int j;
   if(m<n){
      partitionare(x,m,n,j);
      quickSort(x,m,j-1); //*4
      quickSort(x,j+1,n); //*5
   }
}
int main(){
   //eșantion de test 1 (test case)
    int v[]={9,8,7,6,5}; quickSort(v,0,4);
   //rezultat v[]={5,6,7,8,9}
   return 0;
```

Se consideră mutanții:

a. QS_1, obținut din QS înlocuind în //*1: "pivot=x[m];" prin "pivot=x[n];" (mutant de ordin1)
b. QS_2, obținut din QS înlocuind în //*2: "int i=m+1;" prin "int i=m;"
c. QS_3, obținut din QS înlocuind în //*3: "<" prin "<="
d. QS_4, obținut din QS înlocuind în //*4: " quickSort(x,j-1,n);" prin " quickSort(x,j,n);"
e. QS_5, obținut din QS înlocuind în //*5: " quickSort(x,j+1,n);" prin " quickSort(x,j,n);"
f. QS_6, obținut din QS înlocuind în //*6: " m < n" prin "m <= n"
g. QS_46, obținut din QS cu modificarile de la QS4 si QS6, simultan (mutant de ordin 2)
h. QS_56, obținut din QS cu modificarile de la QS5 si QS6, simultan

1.1. Pentru fiecare din mutanții QS_k pe care îi considerați neechivalenți cu QS, găsiți un eșantion v_k[]={…} (test case) concludent (QS și mutantul dau rezultate diferite, adică sunt distinși prin v_k).
2.2. Pentru testul T={v_k}, alcătuit din eșantioanele găsite,calculați scorul de adecvare al testului, S=D/(M-E), unde D este numărul mutanților distinși prin testul T, M este numărul total iar E este numărul celor pe care îi considerați echivalenți.

1. Se consideră următoarele instrucțiuni Java:

1int x, y, z;

```
2x=m; z=n;
3if ( x>z) {y=1; System.out.println("y=" +y); y=1/(x-z-1); }
        4       else {y=2; System.out.println ("y="+y);}
5if (x>z+1)     {y=3; System.out.println ("y="+y);}
                6 else {y=4; System.out.println ("y="+y);}
```

a. Determinaţi un test T1= {(x=m1, z=n1), (x=m2,z=n2)}, cu două eşantioane, care să acopere toate arcele;
b. Determinaţi un test T2= {(x=m, z=n) } cu un singur eşantion care să semnaleze eroare (divide by zero);
c. Poate fi semnalată eroarea prin teste de tipul T1? Justificaţi răspunsul.
d. Pot fi acoperite toate arcele printr-un test cu un singur eşantion? Justificaţi.

3. Fie următorul program:

```
int[] x = new int[dim]; //dim>=1
int i= 0; int j= dim-1; int pivot= x[ (i+j) / 2 ];
while (i <= j) {
while (x[i] < pivot) i= i+1;
while (x[j] > pivot) j= j-1;
if(i <= j) {int t= x[i];x[i]= x[j];x[j]=t;i=i+1;j=j-1;}
}
```

Acest program se termină.
Se consideră mutantul obţinut prin înlocuirea condiţiei if(i <= j)prin if(i < j).
Găsiţi un test T={<x,dim>} care să „ucidă” mutantul.

4. Se considera urmatoarea problema:
“Fie n un numar natural dat cu cel mult 5 cifre si c o cifra data. Determinati numarul de aparitii ale cifrei c printre cifrele numarului n".
1. exemplificati partitionarea de echivalenta pentru problema enuntata;
2. scrieti un program C++/JAVA care sa rezolve problema;
3. transformati programul de mai sus intr-un graf neorientat si determinati pentru acesta numarul circuitelor liniar independente.