Paraschiv Alexandru

Grupa 2

Testarea Sistemelor Software

Functia de testat: se cere un text, un cuvant, un numar n de aparitii si un cuvant de continuare cu valori “y” sau “n”. Se va folosi libraria pytest pentru a realiza testarile functionale si structurale.

Test 1) Testare functionala (metoda grafului cauza-efect)

Cauze:

C1: n<=0

C2: n>nr de cuvinte din text

C3: n nu este integer

C4: lungime cuvant = 0

C5: lungime cuvant > lungime text

C6: cuvantul de continuare nu are valori “y” sau “n”

C7: n>0

C8: n<=nr de cuvinte din text

C9: n este integer

C10: lungime cuvant >0

C11: lungime cuvant <= lungime text

C12: cuvantul de continuare are valoarea “y”
C13: cuvantul de continuare are valoarea “n”

C14: cuvantul se gaseste de n ori in text

C15: cuvantul nu se gaseste de n ori in text

C16: cuvantul nu se gaseste in text

Efecte:

Ef 1: se afiseaza “Please enter a valid number of occurrences”

Ef 2: se afiseaza “Please enter a valid word”

Ef 3: se afiseaza “Please enter a valid text and word”

Ef 4: se afiseaza “Please enter a valid continue word”

Ef 5: se afiseaza “The word is present in the text with n occurrences”

Ef 6: se afiseaza “The word is NOT present in the text with n occurrences”

Ef 7: se afiseaza “The word is NOT present in the text”

Ef 8: se afiseaza “Enter new info”

Ef 9: se afiseaza “Exiting”

Testari efectuate:

	1	2	3	4	5	6	7	8	9	10	11
C1	1	0	0	0	0	0	0	0	0	1	0
C2	0	1	0	0	1	0	0	0	0	0	0
C3	0	0	1	1	0	1	0	0	0	0	1
C4	0	0	0	1	0	1	0	0	0	1	0
C5	1	0	0	0	1	0	0	0	0	0	1
C6	0	0	0	0	0	1	0	0	0	1	1
C7	0	1	0	0	1	0	1	1	1	0	0
C8	0	0	0	0	0	0	1	1	1	0	0

C9	1	1	0	0	1	0	1	1	1	1	0
C10	1	1	1	0	1	0	1	1	1	0	1
C11	0	1	1	1	0	1	1	1	1	1	0
C12	1	0	0	1	0	0	0	1	0	0	0
C13	0	1	1	0	1	0	1	0	1	0	0
C14	0	0	0	0	0	0	0	0	1	0	0
C15	0	0	0	0	0	0	1	0	0	0	0
C16	0	0	0	0	0	0	0	1	0	0	0
Ef 1	1	1	1	1	1	1	0	0	0	1	1
Ef 2	0	0	0	1	0	1	0	0	0	1	0
Ef 3	1	0	0	0	1	0	0	0	0	0	1
Ef 4	0	0	0	0	0	1	0	0	0	1	1
Ef 5	0	0	0	0	0	0	0	0	1	0	0
Ef 6	0	0	0	0	0	0	1	0	0	0	0
Ef 7	0	0	0	0	0	0	0	1	0	0	0
Ef 8	1	0	0	1	0	0	0	1	0	0	0
Ef 9	0	1	1	0	1	0	1	0	1	0	0

Cand o conditia (sau efect) are valoarea 1 inseamna ca se aplica in exemplu, daca are valoarea 0 atunci nu se aplica

Rezultate testare:

```
============================= test session starts ==============================
collecting ... collected 11 items

test.py::test_function_cause_and_effect[test-cattt--1-y-expecteds0] PASSED [  9%]
test.py::test_function_cause_and_effect[cat cat-cat-3-n-expecteds1] PASSED [ 18%]
test.py::test_function_cause_and_effect[cat cat-cat-d-n-expecteds2] PASSED [ 27%]
test.py::test_function_cause_and_effect[cat cat--d-y-expecteds3] PASSED  [ 36%]
test.py::test_function_cause_and_effect[cat cat-catttttttttttt-3-n-expecteds4] PASSED [ 45%]
test.py::test_function_cause_and_effect[cat cat--d-ns-expecteds5] PASSED [ 54%]
test.py::test_function_cause_and_effect[cat Cat dog Dog-cat-3-n-expecteds6] PASSED [ 63%]
test.py::test_function_cause_and_effect[cat Cat dog Dog-hat-3-y-expecteds7] PASSED [ 72%]
test.py::test_function_cause_and_effect[cat Cat dog Dog-cat-2-n-expecteds8] PASSED [ 81%]
test.py::test_function_cause_and_effect[cat Cat dog Dog---3-ns-expecteds9] PASSED [ 90%]
test.py::test_function_cause_and_effect[cat-catt-d-ns-expecteds10] PASSED [100%]

============================== 11 passed in 0.09s ==============================
```

Test 2) Testare Structurala

Se numeroteaza instructiunile din program:

```
   class Function:
       def word_has_n_occurrences_in_text(self,string,word,n,y):
1.         a = string.split()
2.         valid=True
3.         r=''
4.         if type(n)!=int or n<=0 or n>len(a) :
5.             print("Please enter a valid number of occurrences")
6.             valid=False
7.             r+='1'
8.         if len(word)==0:
9.             print("Please enter a valid word")
10.            valid = False
11.            r += '2'
12.        if len(word)>len(string):
13.            print("Please enter a valid text and word")
14.            valid = False
15.            r += '3'
16.        if y!='y' and y!='n':
17.            print("Please enter a valid continue word")
18.            y='y'
19.            valid = False
20.            r += '4'
21.        if valid:
22.            count=0
23.            for i in a:
24.                if i.lower()==word.lower():
25.                    count+=1
26.            if count==n:
27.                print(f"The word is present in the text with {n} occurrences\n")
28.                r += '5'
29.            elif count!=n and count>0:
30.                print(f"The word is NOT present in the text with {n} occurrences\n")
31.                r += '6'
32.            elif count==0:
33.                print(f"The word is NOT present in the text\n")
34.                r += '7'
35.        if y=='y':
36.            print("Enter new info")
37.            r += '8'
38.        elif y=='n':
```

```
39.             print("Exiting")
40.             r += '9'
41.         r=int(r)
42.         return r
```

Se creaza graful de flux de control

Se genereaza testele de acoperire la nivel de:

a) Instructiune – se acopera fiecare instructiune din program cel putin o data
b) Decizie – se acopera fiecare ramura (while,for,if,else) cel putin o data
c) Conditie – se acopera fiecare conditie cel putin o data
d) Circuit – se acopera fiecare cale din graf cel putin o data

Rezultate testare:

```
============================= test session starts ==============================
collecting ... collected 24 items

test.py::test_statement_coverage[test---1-ds-expecteds0] PASSED          [  4%]
test.py::test_statement_coverage[cat cat-catttttttt-1-n-expecteds1] PASSED [  8%]
test.py::test_statement_coverage[cat Cat dog Dog-cat-3-n-expecteds2] PASSED [ 12%]
test.py::test_statement_coverage[cat Cat dog Dog-hat-3-y-expecteds3] PASSED [ 16%]
test.py::test_statement_coverage[cat Cat dog Dog-cat-2-n-expecteds4] PASSED [ 20%]
test.py::test_branch_coverage[test---1-ds-expecteds0] PASSED             [ 25%]
test.py::test_branch_coverage[cat cat-catttttttt-1-n-expecteds1] PASSED  [ 29%]
test.py::test_branch_coverage[cat Cat dog Dog-cat-3-n-expecteds2] PASSED [ 33%]
test.py::test_branch_coverage[cat Cat dog Dog-hat-3-y-expecteds3] PASSED [ 37%]
test.py::test_branch_coverage[cat Cat dog Dog-cat-2-n-expecteds4] PASSED [ 41%]
test.py::test_condition_coverage[test---1-ds-expecteds0] PASSED          [ 45%]
test.py::test_condition_coverage[cat cat-catttttttt-1-n-expecteds1] PASSED [ 50%]
test.py::test_condition_coverage[cat Cat dog Dog-cat-3-n-expecteds2] PASSED [ 54%]
test.py::test_condition_coverage[cat Cat dog Dog-hat-3-y-expecteds3] PASSED [ 58%]
test.py::test_condition_coverage[test--d-ds-expecteds4] PASSED           [ 62%]
test.py::test_condition_coverage[test--2-ds-expecteds5] PASSED           [ 66%]
test.py::test_condition_coverage[cat Cat dog Dog-cat-2-n-expecteds6] PASSED [ 70%]
test.py::test_circuit_coverage[test---1-ds-expecteds0] PASSED            [ 75%]
test.py::test_circuit_coverage[cat cat-catttttttt-1-n-expecteds1] PASSED [ 79%]
test.py::test_circuit_coverage[cat Cat dog Dog-cat-3-n-expecteds2] PASSED [ 83%]
test.py::test_circuit_coverage[cat Cat dog Dog-hat-3-y-expecteds3] PASSED [ 87%]
test.py::test_circuit_coverage[test--d-ds-expecteds4] PASSED             [ 91%]
test.py::test_circuit_coverage[test--2-ds-expecteds5] PASSED             [ 95%]
test.py::test_circuit_coverage[cat Cat dog Dog-cat-2-n-expecteds6] PASSED [100%]

============================== 24 passed in 0.04s ==============================
```

Test 3) Testare mutatnti

Se vor folosi librariile unittest si cosmic-ray pentru a genera testari pentru mutanti:

Se va folosi un script nou pentru testarea mutantilor (cu un input care acopera toate ramurile/conditiile din functie):

```
import unittest
from function_for_test import Function


clasa_test = Function()
```

```python
class Tests(unittest.TestCase):
    def test_func(self):
        self.assertEqual(clasa_test.word_has_n_occurrences_in_text("test", "", -1, 'ds'),
1248)
        self.assertEqual(clasa_test.word_has_n_occurrences_in_text("cat cat",
"catttttttt", 1, 'n'), 39)
        self.assertEqual(clasa_test.word_has_n_occurrences_in_text("cat Cat dog Dog",
"cat", 3, 'n'), 69)
        self.assertEqual(clasa_test.word_has_n_occurrences_in_text("cat Cat dog Dog",
"hat", 3, 'y'), 78)
        self.assertEqual(clasa_test.word_has_n_occurrences_in_text("cat Cat dog Dog",
"cat", 2, 'n'), 59)
```

Dupa se creaza (din terminal) un nou fisier config (toml):

```
(.venv) PS C:\Users\alex\PycharmProjects\pythonProject\proiect\pythonProject1> cosmic-ray new-config mutants.toml
[?] Top-level module path: function_for_test.py
[?] Test execution timeout (seconds): 10
[?] Test command: python -m unittest test_function_for_test.py
-- MENU: Distributor --
  (0) http
  (1) local
[?] Enter menu selection: 1
```

Se incepe sesiunea de testare:

```
(.venv) PS C:\Users\alex\PycharmProjects\pythonProject\proiect\pythonProject1> cosmic-ray init mutants.toml mutants.sqlite
```

Aceasta comanda creaza fisierul sqlite, un fisier care va stoca toti mutantii

Se incepe testarea:

```
(.venv) PS C:\Users\alex\PycharmProjects\pythonProject\proiect\pythonProject1> cosmic-ray exec mutants.toml mutants.sqlite
```

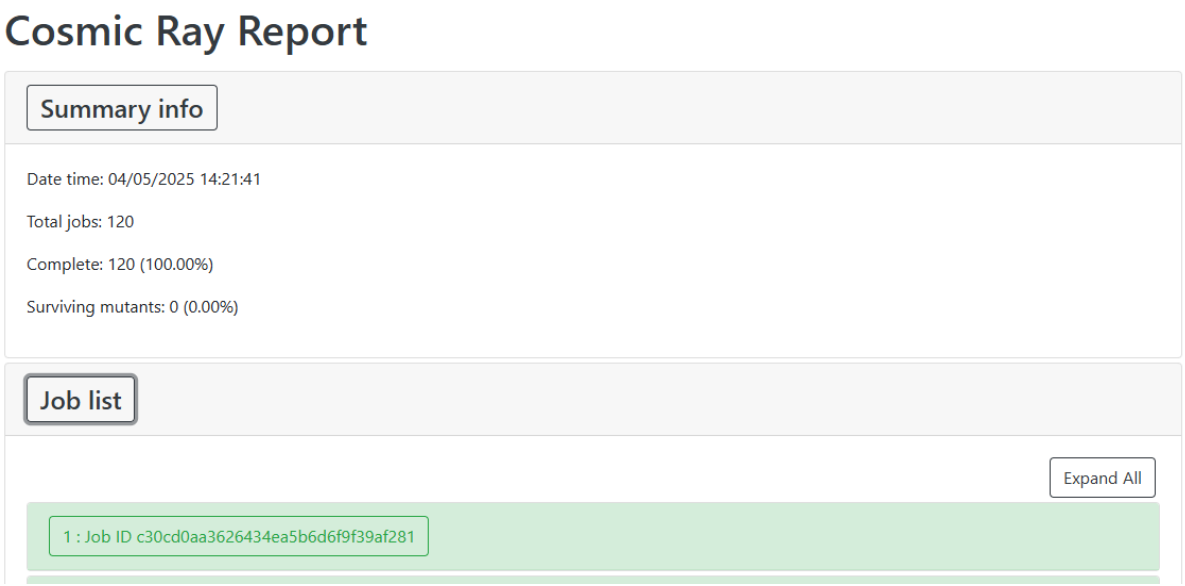
Aceasta comanda va crea toti mutantii posibili si incearca sa-i rezolve

Se genereaza un raport:

```
(.venv) PS C:\Users\alex\PycharmProjects\pythonProject\proiect\pythonProject1> cr-html mutants.sqlite > report_mutanti.html
```

Aceasta comanda va crea un raport html cu toti mutantii.

Rezultat:



Toti mutantii au fost rezolvati