Code Coverage

IIC2143

Prof. Juan Pablo Sandoval

Coverage

Code Coverage o cobertura de código es una métrica que nos permite determinar con el grado de completitud se ha probando las diversas partes de un programa.

- ***Statement Coverage.*** *Statements ejecutados por las pruebas respecto al total*
- ***Branch Coverage.*** *Branches ejecutados por las pruebas respecto al total.*
- ***Function Coverage.*** *Funciones ejecutadas por las pruebas respecto al total.*

Contenido

- *Function and Test Example*
- *Trace Function*
- *Simple Coverage Visualization*
- *La clase Coverage*
- *Diferencia de Cobertura*

Contenido

- ***Function and Test Example***
- *Trace Function*
- *Simple Coverage Visualization*
- *La clase Coverage*
- *Diferencia de Cobertura*

Analizando
Ejecuciones de Código

```python
def cgi_decode(s: str) -> str:
    """Decode the CGI-encoded string `s`:
       * replace '+' by ' '
       * replace "%xx" by the character with hex number xx.
       Return the decoded string.  Raise `ValueError` for invalid inputs."""
    hex_values = {
        '0': 0, '1': 1, '2': 2, '3': 3, '4': 4,
        '5': 5, '6': 6, '7': 7, '8': 8, '9': 9,
        'a': 10, 'b': 11, 'c': 12, 'd': 13, 'e': 14, 'f': 15,
        'A': 10, 'B': 11, 'C': 12, 'D': 13, 'E': 14, 'F': 15,
    }

    t = ""
    i = 0
    while i < len(s):
        c = s[i]
        if c == '+':
            t += ' '
        elif c == '%':
            digit_high, digit_low = s[i + 1], s[i + 2]
            i += 2
            if digit_high in hex_values and digit_low in hex_values:
                v = hex_values[digit_high] * 16 + hex_values[digit_low]
                t += chr(v)
            else:
                raise ValueError("Invalid encoding")
        else:
            t += c
        i += 1
    return t
```

Ejemplo de uso “cgi_decode”

```
cgi_decode("Hello+World")
```

```
Hello World
```

Output

Test Básicos

```
assert cgi_decode('+') == ' '
assert cgi_decode('%20') == ' '
assert cgi_decode('abc') == 'abc'

# Prueba que la función lanze error con el siguiente input
try:
    cgi_decode('%?a')
    assert False
except ValueError:
    pass
```

Prueba Correcto reemplazo de +

Prueba Correcto reemplazo de %xx

Prueba de no reemplazo

Prueba reconomiento de entradas ilegales

Test Básicos

```
assert cgi_decode('+') == ' '
assert cgi_decode('%20') == ' '
assert cgi_decode('abc') == 'abc'

# Prueba que la función lanze error con el siguiente input
try:
    cgi_decode('%?a')
    assert False
except ValueError:
    pass
```

*Cual seria el **Coverage** de estas pruebas?*

Contenido

- *Function and Test Example*
- ***Trace Function***
- *Simple Coverage Visualization*
- *La clase Coverage*
- *Diferencia de Cobertura*

Trace Function

```
import sys
sys.settrace(trace_function)
def trace_function(frame, event, arg):
    ...
```

frame → *el frame actual de ejecución (objeto frame de Python, con info sobre variables locales, globales, nombre del archivo, etc.).*

event → *cadena que indica qué pasó:*

- *"call" → se llamó a una función.*
- *"line" → se va a ejecutar una nueva línea.*
- *"return" → la función actual va a retornar.*
- *"exception" → se lanzó una excepción.*

arg → *depende del evento:*

- *En "call" → es None.*
- *En "return" → es el valor retornado.*
- *En "exception" → es una tupla (exc_type, exc_value, traceback).*

Ejemplo Básico

```python
import sys

def trace_func(frame, event, arg):
    if event == "call":
        print(f"Llamando a {frame.f_code.co_name}")
    elif event == "line":
        print(f"Ejecutando línea {frame.f_lineno}")
    elif event == "return":
        print(f"Retornando valor: {arg}")
    return trace_func  # seguir trazando

def saludar(nombre):
    print(f"Hola, {nombre}")
    return "listo"

sys.settrace(trace_func)
saludar("Juan")
sys.settrace(None)  # desactivar
```

Guardando el numero de lineas que se ejecutaron

```
from types import FrameType, TracebackType
coverage = []

def traceit(frame, event, arg):
    """Trace program execution. To be passed to sys.settrace()."""
    if event == 'line':
        global coverage
        function_name = frame.f_code.co_name
        lineno = frame.f_lineno
        coverage.append(lineno)
    return traceit
```

Guarda la linea en una variable global

Activando/Desactivando la Función

```
import sys

def cgi_decode_traced(s):
    global coverage
    coverage = []
    # Turn on
    sys.settrace(traceit)
    cgi_decode(s)
    # Turn off
    sys.settrace(None)

cgi_decode_traced("a+b")
print(coverage)
```

```
[8, 9, 8, 9, 8, 9, 8, 9,
8, 9, 8, 10, 8, 10, 8, 10,
8, 10, 8, 10, 8, 11, 8,
11, 8, 11, 8, 11, 8, 11,
8, 11, 8, 12, 8, 12, 8,
15, 16, 17, 18, 19, 21,
30, 31, 17, 18, 19, 20,
31, 17, 18, 19, 21, 30,
31, 17, 32]
```

Output

Contenido

- *Function and Test Example*
- *Trace Function*
- ***Simple Coverage Visualization***
- *La clase Coverage*
- *Diferencia de Cobertura*

Obteniendo el Código Fuente

Función a Analizar

```
import inspect
cgi_decode_code = inspect.getsource(cgi_decode)
lines = cgi_decode_code.splitlines()
for i, line in enumerate(lines, start=1):
    print(f"{i:3}: {line}")
```

Imprime linea por linea

```
  1: def cgi_decode(s: str) -> str:
  2:     """Decode the CGI-encoded string `s`:
  3:        * replace '+' by ' '
  4:        * replace "%xx" by the character with hex number xx.
  5:        Return the decoded string.  Raise `ValueError` for invalid inputs."""
  6:
  7:     # Mapping of hex digits to their integer values
  8:     hex_values = {
  9:         '0': 0, '1': 1, '2': 2, '3': 3, '4': 4,
 10:         '5': 5, '6': 6, '7': 7, '8': 8, '9': 9,
 11:         'a': 10, 'b': 11, 'c': 12, 'd': 13, 'e': 14, 'f': 15,
 12:         'A': 10, 'B': 11, 'C': 12, 'D': 13, 'E': 14, 'F': 15,
 13:     }
 14:
 15:     t = ""
 16:     i = 0
 17:     while i < len(s):
 18:         c = s[i]
 19:         if c == '+':
 20:             t += ' '
 21:         elif c == '%':
 22:             digit_high, digit_low = s[i + 1], s[i + 2]
 23:             i += 2
 24:             if digit_high in hex_values and digit_low in hex_values:
 25:                 v = hex_values[digit_high] * 16 + hex_values[digit_low]
 26:                 t += chr(v)
 27:             else:
 28:                 raise ValueError("Invalid encoding")
 29:         else:
 30:             t += c
 31:         i += 1
 32:     return t
```

Colecta las lineas ejecutadas

Ejecutar la Función

Elimina duplicados

Imprime linea por linea, si la linea fue ejecutada imprime un **#** *al inicio*

```
import sys

def cgi_decode_traced(s):
    global coverage
    coverage = []
    sys.settrace(traceit) # Turn on
    cgi_decode(s)
    sys.settrace(None)    # Turn off

cgi_decode_traced("a+b")

covered_lines = set(coverage)

for lineno in range(1, len(cgi_decode_lines)):
    if lineno not in covered_lines:
        print("# ", end="")
    else:
        print("  ", end="")
    print("%2d  " % lineno, end="")
    print(cgi_decode_lines[lineno])
    print()
```

Output

```
#  1  def cgi_decode(s: str) -> str:
#  2      """Decode the CGI-encoded string `s`:
#  3         * replace '+' by ' '
#  4         * replace "%xx" by the character with hex number xx.
#  5         Return the decoded string.  Raise `ValueError` for invalid inputs."""
#  6
#  7      # Mapping of hex digits to their integer values
   8      hex_values = {
   9          '0': 0, '1': 1, '2': 2, '3': 3, '4': 4,
  10          '5': 5, '6': 6, '7': 7, '8': 8, '9': 9,
  11          'a': 10, 'b': 11, 'c': 12, 'd': 13, 'e': 14, 'f': 15,
  12          'A': 10, 'B': 11, 'C': 12, 'D': 13, 'E': 14, 'F': 15,
# 13      }
# 14
  15      t = ""
  16      i = 0
  17      while i < len(s):
  18          c = s[i]
  19          if c == '+':
  20              t += ' '
  21          elif c == '%':
# 22              digit_high, digit_low = s[i + 1], s[i + 2]
# 23              i += 2
# 24              if digit_high in hex_values and digit_low in hex_values:
# 25                  v = hex_values[digit_high] * 16 + hex_values[digit_low]
# 26                  t += chr(v)
# 27              else:
# 28                  raise ValueError("Invalid encoding")
# 29          else:
  30              t += c
  31          i += 1
  32      return t
```

Contenido

- *Function and Test Example*
- *Trace Function*
- *Simple Coverage Visualization*
- ***La clase Coverage***
- *Diferencia de Cobertura*

Diseño

```
with Coverage() as cov:
    function_to_be_traced()
c = cov.coverage()
```

*Queremos una clase **Coverage** que me devuelva el coverage de una función ejecutada (**function_to_be_traced**)*

Coverage

Ver el código en el Jupyter book en Canvas.

Ejemplo de Uso

```
with Coverage() as cov:
    cgi_decode("a+b")

print(cov.coverage())
```

Devuelve las lineas ejecutadas y la función en la que se encuentra esa linea. Elimina duplicados.

```
{('cgi_decode', 18), ('cgi_decode', 31),
('cgi_decode', 8), ('cgi_decode', 11),
('cgi_decode', 17), ('cgi_decode', 30),
('cgi_decode', 20), ('cgi_decode', 10),
('cgi_decode', 19), ('cgi_decode', 32),
('cgi_decode', 16), ('cgi_decode', 12),
('cgi_decode', 9), ('cgi_decode', 15),
('cgi_decode', 21)}
```

Output

Contenido

- *Function and Test Example*
- *Trace Function*
- *Simple Coverage Visualization*
- *La clase Coverage*
- ***Diferencia de Cobertura***

Comparación de Cobertura

```
with Coverage() as cov_plus:
    cgi_decode("a+b")
with Coverage() as cov_standard:
    cgi_decode("abc")

cov_plus.coverage() -
    cov_standard.coverage()
```

El primer test cubre la linea 20, el segundo test no.

```
{('cgi_decode', 20)
```

Output

Si probamos todos los casos

```
with Coverage() as cov_max:
    cgi_decode('+')
    cgi_decode('%20')
    cgi_decode('abc')
    try:
        cgi_decode('%?a')
    except Exception:
        pass

cov_max.coverage() - cov_plus.coverage()
```

*Podemos ver **lineas que faltan testear**, simplemente restamos el coverage máximo (**cov_max)** menos el coverage actual (**cov_plus**)*

```
{('cgi_decode', 22),
 ('cgi_decode', 23),
 ('cgi_decode', 24),
 ('cgi_decode', 25),
 ('cgi_decode', 26),
 ('cgi_decode', 28)}
```

Output

Simple Fuzzing

Bonus

Prof. Juan Pablo Sandoval

fuzzer function

- *La función **fuzzer()** retorna un string aleatorio.*

```
from fuzzingbook.Fuzzer import fuzzer
sample = fuzzer()
print(sample)
```

```
!7#%"*#0=)$;%6*;>638:*>80"=</
>(/*:-(2<4 !:5*6856&?""11<7+
%<%7,4.8,*+&,,$,."
```

Output

Calculando el coverage de un input random

```
sample = fuzzer()
print(sample)
with Coverage() as cov_fuzz:
    try:
        cgi_decode(sample)
    except:
        pass
cov_fuzz.coverage()
```

```
{('cgi_decode', 8),
 ('cgi_decode', 9),
 ('cgi_decode', 10),
 ('cgi_decode', 11),
 ('cgi_decode', 12),
 ('cgi_decode', 15),
 ('cgi_decode', 16),
 ('cgi_decode', 17),
 ('cgi_decode', 18),
 ('cgi_decode', 19),
 ('cgi_decode', 21),
 ('cgi_decode', 22),
 ('cgi_decode', 23),
 ('cgi_decode', 24),
 ('cgi_decode', 28),
 ('cgi_decode', 30),
 ('cgi_decode', 31)}
```

Output

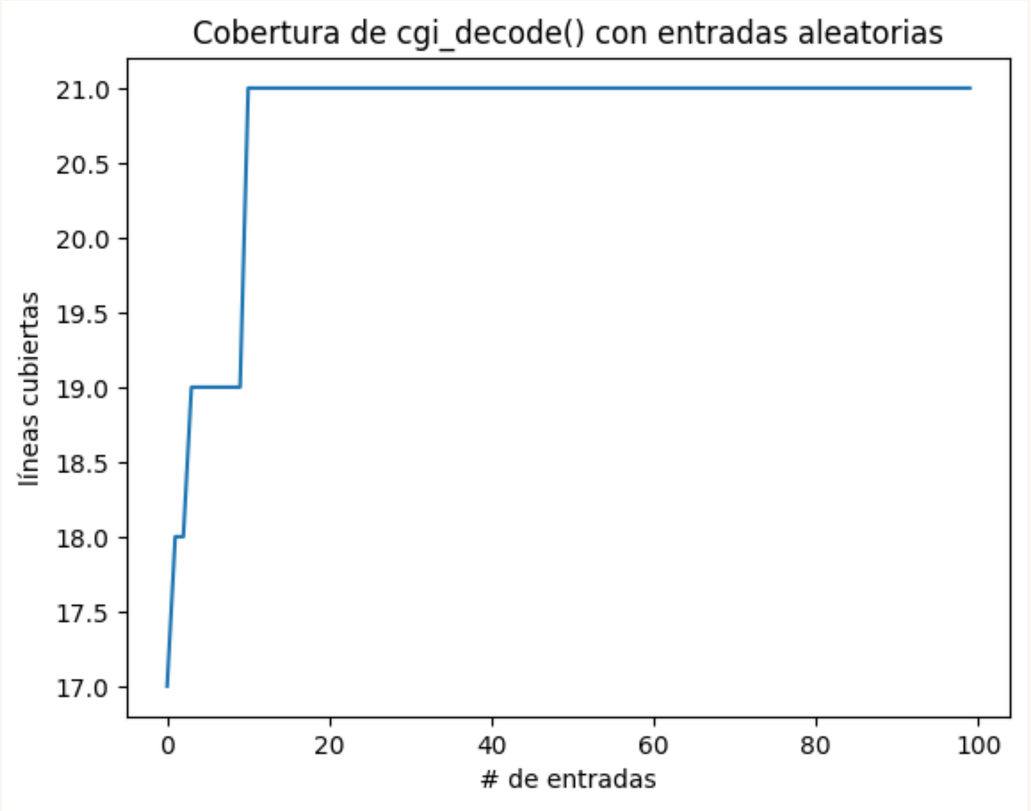
Coverage Acumulado de 100 inputs Random

```
trials = 100
def population_coverage(population: List[str], function: Callable) \
        -> Tuple[Set[Location], List[int]]:
    cumulative_coverage: List[int] = []
    all_coverage: Set[Location] = set()

    for s in population:
        with Coverage() as cov:
            try:
                function(s)
            except:
                pass
        all_coverage |= cov.coverage()
        cumulative_coverage.append(len(all_coverage))

    return all_coverage, cumulative_coverage
```

*Acumula el **coverage**, hace un **join** de dos **Sets***

Los inputs random eventualmente cubren lineas antes no cubiertas

Después del input 15 (mas o menos), los siguientes inputs random no logran cubrir ninguna linea adicional.



Ejecutamos el experimento 100 veces, ya que los inputs random pueden variar

Después de probar 30 (mas o menos) inputs random, llegamos al coverage máximo. Al menos para esta función en particular.