Ingeniería del Software II

Taller #4 – Algoritmos Genéticos

Ejercicio 1

Sea el siguiente programa `cgi_decode`:

```
def cgi_decode(s):
    """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."""

    # Mapping of hex digits to their integer values
    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): //c1
        c = s[i]
        if c == '+': //c2
            t += ' '
        elif c == '%': //c3
            digit_high, digit_low = s[i + 1], s[i + 2]
            i += 2
            if digit_high in hex_values and digit_low in hex_values: // c4 and c5
                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
```

Escribir un test suite que tenga 100 % de cubrimiento de líneas y de branches. Para escribir los tests utilize la librería de Python `unittest`, y para obtener su cobertura la librería `coverage`.

El Taller provee un archivo `requirements.txt` para instalar todas las dependencias necesarias en un ambiente virtual de Python. Para instalar estas dependencias, ejecute los siguientes comandos en la carpeta del taller:

```
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

Luego, puede ejecutar el test suite y obtener su cobertura desde una IDE, como *PyCharm*, o desde la terminal ejecutando los siguientes comandos:

```
./venv/bin/coverage run --branch -m unittest discover
./venv/bin/coverage html
```

Ejercicio 2

Sea la siguiente función `update_maps`

```
from typing import Dict
# Inicializar mappings globales
distances_true: Dict[int, int] = {}
distances_false: Dict[int, int] = {}

def update_maps(condition_num, d_true, d_false):
    global distances_true, distances_false

    if condition_num in distances_true.keys():
        distances_true[condition_num] = min(
            distances_true[condition_num], d_true)
    else:
        distances_true[condition_num] = d_true

    if condition_num in distances_false.keys():
        distances_false[condition_num] = min(
            distances_false[condition_num], d_false)
    else:
        distances_false[condition_num] = d_false
```

a. Escribir una función `evaluate_condition` que recibe los siguientes argumentos:

- condition_num: un entero que representa el identificador de la condición (i.e. branch)
- op: La operación de comparación. Las comparaciones puede ser "Eq" (==), "Ne" (!=), "Lt" (<), "Gt" (>), "Le" (<=), "Ge" (>=), "In")
- lhs: el valor de la expresión izquierda de la comparación
- rhs: el valor de la expresión derecha de la comparación

La operación retorna `True` o `False` de acuerdo a si la condición es verdadera o falsa, actualizando los mappings invocando a `update_maps`. Por ejemplo,

- `evaluate_condition(1, ''Eq'', 10, 20)` retorna False y actualiza el distances_true, distances_false para la condición 1
- `evaluate_condition(2, ''Eq'', 20, 20)` retorna True y actualiza el distances_true, distances_false para la condición 2
- `evaluate_condition(4, ''In'', 'a', ['b','c','d'])` retorna False y actualiza el distances_true, distances_false para la condición 4. Usar la función *ord*() dentro de `evaluate_condition()` sin modificar su signatura.

El valor de actualización para distances_true, distances_false, utilizando $K = 1$ para calcular la distancia de branch, es el que sigue:

Operación	distance_true	distance_false
20 == 10	10	0
20 == 20	0	1
20 != 10	0	10
20 != 20	1	0
$10 \leq 20$	0	11
$20 \leq 10$	10	0
$20 \leq 20$	0	1
$10 < 20$	0	10
$20 < 10$	11	0
$20 < 20$	1	0
10 In []	sys.maxsize	0
10 In [1,2,3]	7	0
10 In [10]	0	1
10 In [10,10]	0	1
13 in [11,12,18]	1	0

b. Escribir un test suite que tenga 100 % de cubrimiento de líneas y de branches usando unittest para `evaluate_condition`.

Ejercicio 3

Representaremos cada caso de test como un string, mientras que un test suite será representado con una lista de test cases.

Crear una función `cgi_decode_instrumented(test_case)` donde cada condición del programa original es reemplazada por la llamada correspondiente a `evaluate_condition` indicando el identificador de la condición. Usando los casos de test del ejercicio anterior como inspiración, escribir casos de test nuevos para comprobar que `distances_true` y `distances_false` son actualizados correctamente al ejecutar nuestro programa instrumentado. Ejemplo:

- Ejecutando `cgi_decode_instrumented(''Hello+Reader'')` retorna "Hello Reader"
- El mapping `distances_true` queda {`1: 0, 2: 0, 3: 35`}
- El mapping `distances_false` queda {`1: 0, 2: 0, 3: 0`}

Ejercicio 4

Se desea crear una función de fitness para guiar inputs que ejerciten todo el código del programa `cgi_decode` .

Crear una función `get_fitness_cgi_decode(test_suite)` que computa el valor de fitness para una lista de casos de tests usando la función `cgi_decode_instrumented(test_case)`. Dado que estamos usando *branch coverage*, el fitness va a estar dado por la suma de un valor determinado para cada objetivo (una rama verdadera o falsa en un *branch*) en el programa que estamos testeando. Para un objetivo en particular, si el test suite logra ejecutar el *branch*, entonces usamos como valor la distancia normalizada[1]. Sino, el valor que usamos es 1.

Tome como guía los siguientes ejemplos y escriba casos de tests para la nueva función:

- `get_fitness_cgi_decode(['' %AA''])` debe retornar ...
- `get_fitness_cgi_decode(['' %AU''])` debe retornar ...
- `get_fitness_cgi_decode(['' %UU''])` debe retornar ...
- `get_fitness_cgi_decode(['Hello+Reader''])` debe retornar ...
- `get_fitness_cgi_decode([''''])` debe retornar ...
- `get_fitness_cgi_decode(['' %''])` debe retornar ...
- `get_fitness_cgi_decode(['' %1''])` debe retornar ...
- `get_fitness_cgi_decode(['+''])` debe retornar ...
- `get_fitness_cgi_decode(['+%1''])` debe retornar ...
- `get_fitness_cgi_decode(['' %1+''])` debe retornar ...

Ejercicio 5

Crear una función `create_population(population_size)` que crea una lista de `size` individuos, y cada individuo es una lista de hasta 15 casos de tests, donde cada caso de test es un string de hasta 10 caracteres. Escriba tests para esta nueva función.

Ejercicio 6

Crear una función `evaluate_population(population)` que dado una lista de individuos, retorna un mapping de cada individuo a su valor de fitness usando la función `get_fitness_cgi_decode`. Tenga en cuenta la función `clear_maps` que limpia los diccionarios de distancias a *branches*. Escriba tests para la función implementada en este ejercicio.

[1] usando la función de normalización $x/x+1$

Ejercicio 7

Crear una función `selection(evaluated_population, tournament_size)` que dado una mapping de individuos a su valor de fitness y un tamaño de torneo como un entero positivo, retorna el ganador del torneo. Escriba tests para esta nueva función.

Ejercicio 8

Crear una función `crossover(parent1, parent2, seed=None)` que dado dos padres retorna el single-point cross-over no-uniforme offspring1 y offspring2. Escriba tests para esta nueva función.

Ejercicio 9

Crear una función `mutate(individual, seed=None)` que dado un individuo aplica una mutación de acuerdo a una probabilidad. La mutación puede (con igual probabilidad) agregar un nuevo caso de test aleatorio de hasta 10 caracteres, eliminar un caso de test, o modificar un caso de test existente. En caso de modificar un caso de test existente, puede (con igual probabilidad) quitar, agregar o modificar un caracter del test. Escriba tests para esta nueva función.

Ejercicio 10

Usando todas las funciones definidas anteriormente, completar la implementación de un algoritmo genético donde los individuos son listas de casos de tests, y cada caso de test es un string. Puede implementar, por ejemplo, el algoritmo genético standard sin elitismo.

```
def genetic_algorithm(seed=None):
    population_size = 100
    tournament_size = 5
    p_crossover = 0.70
    p_mutation = 0.20

    # Generar y evaluar la poblacion inicial
    generation = 0

    population = None # COMPLETAR
    evaluated_population = None # COMPLETAR

    # Imprimir el mejor valor de fitness encontrado
    best_individual = None # COMPLETAR
    fitness_best_individual = None # COMPLETAR

    # Continuar mientras la cantidad de generaciones es menor que 1000
    # y no haya ningun individuo que cubra todos los objetivos

    while True: # COMPLETAR

        # Producir una nueva poblacion en base a la anterior.
        # Usar selection, crossover y mutation.
        new_population = None # COMPLETAR

        # Una vez creada, reemplazar la poblacion anterior con la nueva
        generation += 1
        population = new_population

        # Evaluar la nueva poblacion e imprimir el mejor valor de fitness
        evaluated_population = None # COMPLETAR
        best_individual = None # COMPLETAR
        fitness_best_individual = None # COMPLETAR

    # retornar el mejor individuo de la ultima generacion
    return best_individual
```

Crear 10 tests distintos para la nueva función que informen (utilizando *asserts* en el test):

- La semilla utilizada para el test.
- La cantidad de generaciones que realiza el algoritmo.
- La *fitness* del mejor individuo al final del algoritmo.
- El mejor *branch coverage* logrado al final del algoritmo por el mejor individuo.

Formato de Entrega

El taller debe ser entregado en el campus de la materia. La entrega debe incluir un archivo `entrega.zip` con el código implementado. Este debe estar detalladamente documentado. Además, debe incluir en la documentación una descripción de la resolución de cada ejercicio, incluyendo una breve discusión de las decisiones de diseño más importantes tomadas para resolver el taller.

El archivo `entrega.zip` debe contener también el reporte de coverage generado por `coverage.py` para todos los tests sobre la implementación final.