Ingeniería del Software II

Taller #2 – Algoritmos Genéticos

Deadline: Lunes 19 de septiembre de 2022 a las 23:59 hs
Fecha de Reentrega: Lunes 3 de octubre de 2022 a las 23:59 hs

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 usando unittest

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 es con $K = 1$ y 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

Crear una función `cgi_decode_instrumented(s)` donde cada condición es reemplazada por la llamada correspondiente a `evaluate_condition` indicando el identificador de la condición. Usando los casos de test del ejercicio comprobar que `distances_true` y `distances_false` son actualizados correctamente. 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 el código que decodifica los valores hexadecimales de `cgi_decode`. Para ello, el input debe hacer verdadera la condición 1, falsa la condición 2, verdadera la condición 3, verdadera la condición 4 y verdadera la condición 5.

Crear una función `get_fitness_cgi_decode(s)` que computa el valor de fitness para un input usando la función `cgi_decode_instrumented(s)`. Esta función suma por cada objetivo (i.e. si se desea que la condición sea verdadera o falsa), si fue alcanzado, su distancia normalizada[1], y si no fue alcanzado el valor 1.

Por ejemplo:

- `get_fitness_cgi_decode(''%AA'')` debe retornar 0.0
- `get_fitness_cgi_decode(''%AU'')` debe retornar 0.9230769230769231
- `get_fitness_cgi_decode(''%UU'')` debe retornar 1.9230769230769231
- `get_fitness_cgi_decode(''Hello+Reader'')` debe retornar 2.9722222222222223
- `get_fitness_cgi_decode('''')` debe retornar 4.5
- `get_fitness_cgi_decode(''%'')` debe retornar 2.0
- `get_fitness_cgi_decode(''%1'')` debe retornar 2.0
- `get_fitness_cgi_decode(''+'')` debe retornar 3.5
- `get_fitness_cgi_decode(''+%1'')` debe retornar 2.0
- `get_fitness_cgi_decode(''%1+'')` debe retornar 0.8333333333333334

Ejercicio 5

Crear una función `create_population(population_size)` que crea una lista de `size` individuos, y cada individuo es un string de hasta 10 caracteres.

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`.

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.

Ejercicio 8

Crear una función `crossover(parent1, parent2)` que dado dos padres retorna el single-point cross-over no-uniforme offspring1 y offspring2. Siendo ambos offsprings cadenas de hasta 10 caracteres.

Ejercicio 9

Crear una función `mutate(individual)` que dado un individuo aplica una mutación de acuerdo a una probabilidad. La mutación puede reducir o aumentar la cantidad de caracteres del individuo siempre que el individuo mutado no tenga mas de 10 caracteres.

Ejercicio 10

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

Usando todas las funciones definidas anteriormente, completar la implementación de un algoritmo genético donde los individuos son strings de hasta 10 caracteres. Puede implementar, por ejemplo, el algoritmo genético standard sin elitismo.

```
def genetic_algorithm():
    population_size=100
    tournament_size=5
    p_crossover=0.70
    p_mutation=0.20

    # Generar y evaluar la poblaci'on inicial
    generation = 0
    // ...
    population = ...
    evaluated_population = ...

    # Imprimir el mejor valor de fitness encontrado
    // ...
    best_individual = ...
    fitness_best_individual = ...

    # Continuar mientras la cantidad de generaciones es menor que 1000
    # o no se haya encontrado un m\'inimo global
    while ... :

        # Producir una nueva poblaci'on con el mismo tama~no que
        # la generaci'on anterior, usar selection, crossover y mutation
        new_population = []
        // ...

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

        # Evaluar la nueva poblaci'on e imprimir el mejor valor de fitness
        evaluated_population = ...
        best_individual = ...
        fitness_best_individual = ...

    # retornar el mejor individuo de la ultima generaci'on
    return best_individual
```

Completar la siguiente tabla usando 10 semillas aleatorias distintas

	Semilla	#generaciones	fitness_best_individual	% branch coverage
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

Formato de Entrega

El taller debe ser entregado en el campus de la materia durante la fecha de entrega indicada en el documento.

La entrega debe incluir el siguiente material

1. Un archivo `src.zip` con el código implementado. Este debe estar detalladamente documentado.
2. Un archivo `readme.txt` con instrucciones sobre como obtener los resultados del taller.
3. Un archivo `report.pdf` con la descripción de la resolución de todos los ejercicios, incluyendo una breve discusión de las más importantes decisiones de diseño tomadas para resolver el taller.