Sea el siguiente programa

```
boolean test_me(int x, int y) {
1:   boolean result = false;
2:   int z = 2 * y;
3:   if (z == x) { // c1
4:     if (x > y + 10) {// c2
5:       result = true;
6:     }
7:   }
8:   return result;
}
```

Usando el operador de mutación AOR (modifica una ~~comparación~~ expresión aritmética reemplazandola por +, -, *, /, %, **, x, y) exhibir dos mutantes tales que:

1. Un mutante con un test que lo mata fuertemente (indicando la línea que cambió).
2. Un mutante equivalente tal que no exista test que lo detecte fuertemente (indicando la línea que cambió). En caso de no existir, justificar. → NO Existe

1) linea 4 → $x > y - 10$ test Case ASSERT (test_me(2,1), False)

2) $z = 2 * y$ $2 \cdot y == x$ $x > 5$

$2y > y + 10$ $y > 10$

$\forall x, y \quad y > 10, \; x = 2y \rightarrow$ true

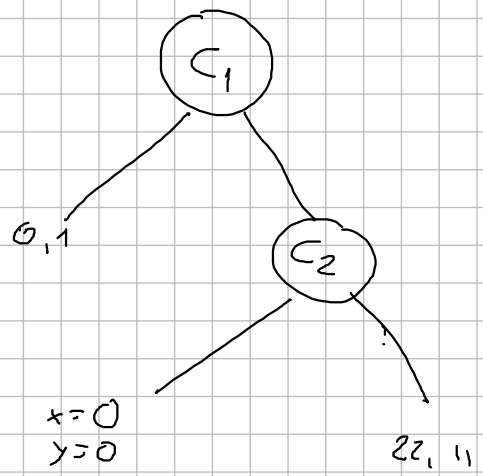
Ejercicio 2 editar

Sea el siguiente programa test_me del ejercicio #1.

1. Describir el árbol de cómputo del programa explorado durante la ejecución simbólica dinámica del programa
2. Completar la siguiente tabla con la ejecución simbólica dinámica del programa de forma manual, indicando para cada iteración:

- El input concreto utilizado
- La condición de ruta (ie path condition) que se produce de ejecutar el input concreto, asumiendo que el valor simbólico inicial es x = x0, y = y0.
- La fórmula lógica (no es necesario escribirla en SMTLib) que se envía al demostrador de teoremas de acuerdo al algoritmo de ejecución simbólica dinámica.
- El resultado posible que podría producir un demostración de teoremas (ej Z3).

Iteración	Input Concreto	Condición de Ruta	Fórmula enviada al demostrador	Resultado posible
1	x=0, y=0	$c_1 \wedge \neg c_2$	$c_1 \wedge c_2$	$x = 22 \; y = 11$
2	22, 11	$c_1 \wedge c_2$	$\neg c_1$	0, 1
...	0, 1	$\neg c_1$	END	...




```
boolean result = false;
int z = 2 * y;
if (z == x) { // c1
  if (x > y + 10) {// c2
    result = true;
  }
}
```

Sea el programa test_me del ejercicio #1 y el siguiente test suite:

```
class TestSuite(unittest.TestCase):
  def test_1(self):
    self.assertEqual(0, test_me(0, 0))

  def test_2(self):
    self.assertEqual(0, test_me(1, 1))

  def test_3(self):
    self.assertEqual(0, test_me(2, 2))
```

1. Completar la tabla con los valores de la distancia de braanch no normalizada para cada decision (tanto true como false) luego de ejecutar **todo** el test suite. El valor de K para la distancia de branch es 1.

	c1	c2
distance_true	0	11
distance_false	0	0

1. ¿Cuál es el cubrimiento de líneas?
2. ¿Cuál es el cubrimiento de branches?

```
boolean test_me(int x, int y) {
1:   boolean result = false;
2:   int z = 2 * y;
3:   if (z == x) { // c1
4:     if (x > y + 10) {// c2
5:       result = true;
6:     }
7:   }
8:   return result;
}
```

1) $C_1\_T = 0$ $C_1\_F = 1$ $C_2\_T = 11$ $C_2\_F = 0$

2) $C_1\_T = 1$ $C_1\_F = 0$ /

3) $C_1\_T = 2$ $C_1\_F = 0$

$CL = 5/6$ $CB = 3/4$

Sea el siguiente programa cgi_decode:

```
def cgi_decode(s):
    # 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 y 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
```

Asumiendo que tenemos un boosted greybox fuzzer con exponente a=4. Sea el siguiente conjunto inicial de inputs: "hola+mundo", "hello+world", "gracias+mundo", "%12", "%AA", "%BB", "mistring".

1. Indicar la energía de cada input en el conjunto inicial
2. ¿Cuál es la probabilidad que el fuzzer elija el input "mistring" para mutar?

$L = C_1 \neg C_2 \neg C_3$

$+ = C_1 C_2$

$\% = C_1 \neg C_2 C_3$

A) HOLA+MUNDO $= L_4 + L_5 \neg C_1$

B) Hello+WORLD $= L_5 + L_5 \neg C_1$

C) GRACIAS+MUNDO $= L_7 + L_5 \neg C_1$

D) $\%12 = \% C_4 C_5 \neg C_1$

E) $\%AA = \% C_4 C_5 \neg C_1$

F) $\%BB = \% C_4 C_5 \neg C_1$

G) MISTRING $= L_8 \neg C_1$

$s^2 = 9$

$A = B = C = G = 1/1^4 = 1$

$D = E = F = 1/3^4 = 1/81$

$P(G) = 1/(4 + 1/81) \approx 1/4$ 0,25