Testing de caja blanca

Guía 9

Repaso

- Test input (dato de prueba): Es un valor concreto de los parámetros de entrada que usamos para ejecutar un programa.
- Test case (caso de prueba): Es un pequeño programa que ejecuta el test input y chequea de forma automática si estos inputs cumplen el resultado esperado o no.
- Test Suite: Es un conjunto de test cases.

Guía 9. Ejercicio 5

```
problema signo (in x: \mathbb{R}, out result: \mathbb{Z}) {
      requiere: \{True\}
      asegura: \{(result = 0 \land x = 0) \lor (result = -1 \land x < 0) \lor (result = 1 \land x > 0)\}
def signo(x: float) -> int:
      result: int = 0
L1:
L2: if x < 0:
            result = -1
L3:
L4:
      elif x>0:
            result = 1
L5:
L6:
       return result
```

- 1. Describir el diagrama de control de flujo (control-flow graph) del programa signo.
- 2. Escribir un test suite que ejecute todas las líneas del programa signo.
- 3. ¿El test suite del punto anterior ejecuta todas las posibles decisiones ("branches") del programa?

SOLUCIÓN

```
def signo(x: float) -> int:
L1:    result: int = 0
L2:    if x<0:
L3:        result = -1
L4:    elif x>0:
L5:        result = 1
L6:    return result
```

Implementación

Control Flow-Graph

Test Suite

```
result: int = 0
def signo(x: float) -> int:
    result: int = 0
L1:
                                             if x<0:
                                       L2
L2: if x < 0:
                                        True
                                                   False
L3:
           result = -1
                                                             False
L4:
     elif x>0:
                                L3
                                     result = -1
                                               L4
                                                    elif x>0:
L5:
           result = 1
                                                   True
L6:
      return result
                                               L5
                                                   result = 1
                                          return result
          Implementación
                                         Control Flow-Graph
```

Test Suite

```
def signo(x: float) -> int:
L1:
       result: int = 0
                                                if x<0:
                                          L2
L2:
     if x < 0:
                                                                         ¿Los casos de test los
                                                      False
                                           True
L3:
            result = -1
                                                                         hacemos viendo el código
                                                                 False
                                                                         o la especificación?
L4:
       elif x>0:
                                  L3
                                       result = -1
                                                  L4
                                                        elif x>0:
L5:
            result = 1
                                                      True
L6:
        return result
                                                   L5
                                                       result = 1
                                             return result
           Implementación
                                            Control Flow-Graph
                                                                                 Test Suite
```

result: int = 0

```
problema signo (in x: \mathbb{R}) : \mathbb{Z} { requiere: \{True\} asegura: \{(result=0 \land x=0) \lor (result=-1 \land x<0) \lor (result=1 \land x>0)\} }
```

Test Suite

```
result: int = 0
def signo(x: float) -> int:
L1:
       result: int = 0
                                         L2
                                               if x<0:
L2:
     if x < 0:
                                                     False
                                          True
L3:
           result = -1
                                                                False
L4:
        elif x>0:
                                  L3
                                      result = -1
                                                 L4
                                                      elif x>0:
L5:
            result = 1
                                                     True
L6:
       return result
                                                 L5
                                                      result = 1
                                            return result
           Implementación
                                           Control Flow-Graph
```

```
problema signo (in x: \mathbb{R}) : \mathbb{Z} { requiere: \{True\} asegura: \{(result=0 \land x=0) \lor (result=-1 \land x<0) \lor (result=1 \land x>0)\} }
```

```
result: int = 0
                                                                               Test Case #1: valorPositivo
def signo(x: float) -> int:
                                                                                    entrada x = 1
        result: int = 0
L1:
                                                                                    salida esperada = 1
                                                   if x<0:
L2:
       if x < 0:
                                                          False
                                              True
                                                                              Test Case #2: valorNegativo
L3:
             result = -1
                                                                      False
                                                                                    entrada x = -3
L4:
         elif x>0:
                                     L3
                                          result = -1
                                                      L4
                                                            elif x>0:
                                                                                    salida esperada = -1
L5:
             result = 1
                                                          True
L6:
        return result
                                                      L5
                                                           result = 1
                                                return result
                                               Control Flow-Graph
                                                                                      Test Suite
            Implementación
```

```
problema signo (in x: \mathbb{R}) : \mathbb{Z} { requiere: \{True\} asegura: \{(result=0 \land x=0) \lor (result=-1 \land x<0) \lor (result=1 \land x>0)\} }
```

```
result: int = 0
                                                                              Test Case #1: valorPositivo
def signo(x: float) -> int:
                                                                                    entrada x = 1
        result: int = 0
L1:
                                                                                    salida esperada = 1
                                                   if x<0:
                                            L2
L2:
       if x < 0:
                                                          False
                                              True
                                                                              Test Case #2: valorNegativo
L3:
             result = -1
                                                                      False
                                                                                    entrada x = -3
L4:
         elif x>0:
                                     L3
                                          result = -1
                                                      L4
                                                           elif x>0:
                                                                                    salida esperada = -1
L5:
             result = 1
                                                          True
L6:
        return result
                                                      L5
                                                           result = 1
                                                return result
                                               Control Flow-Graph
                                                                                      Test Suite
            Implementación
```

```
problema signo (in x: \mathbb{R}) : \mathbb{Z} { requiere: \{True\} asegura: \{(result=0 \land x=0) \lor (result=-1 \land x<0) \lor (result=1 \land x>0)\} }
```

```
result: int = 0
                                                                              Test Case #1: valorPositivo
def signo(x: float) -> int:
                                                                                    entrada x = 1
        result: int = 0
L1:
                                                                                    salida esperada = 1
                                                   if x<0:
L2:
      if x < 0:
                                                          False
                                              True
                                                                              Test Case #2: valorNegativo
L3:
             result = -1
                                                                      False
                                                                                    entrada x = -3
L4:
         elif x>0:
                                     L3
                                          result = -1
                                                      L4
                                                            elif x>0:
                                                                                    salida esperada = -
L5:
             result = 1
                                                          True 🗸
L6:
        return result
                                                      L5
                                                           result = 1
                                                return result
                                               Control Flow-Graph
                                                                                      Test Suite
            Implementación
```

```
problema signo (in x: \mathbb{R}) : \mathbb{Z} { requiere: \{True\} asegura: \{(result=0 \land x=0) \lor (result=-1 \land x<0) \lor (result=1 \land x>0)\} }
```

```
result: int = 0
def signo(x: float) -> int:
        result: int = 0
L1:
                                                if x<0:
L2:
      if x < 0:
                                                                          ¿Cubrimos todas las
                                                       False
                                            True
L3:
            result = -1
                                                                          branches con estos dos
                                                                  False
                                                                          casos de test?
L4:
        elif x>0:
                                   L3
                                        result = -1
                                                   L4
                                                        elif x>0:
L5:
            result = 1
                                                       True
L6:
        return result
                                                   L5
                                                        result = 1
                                              return result
           Implementación
                                             Control Flow-Graph
                                                                                  Test Suite
```

```
problema signo (in x: \mathbb{R}) : \mathbb{Z} { requiere: \{True\} asegura: \{(result=0 \land x=0) \lor (result=-1 \land x<0) \lor (result=1 \land x>0)\} }
```

```
Test Case #1: valorPositivo
                                                result: int = 0
                                                                                    entrada x = 1
def signo(x: float) -> int:
                                                                                    salida esperada = 1
        result: int = 0
L1:
                                                   if x<0:
L2:
      if x < 0:
                                                          False
                                                                               Test Case #2: valorNegativo
                                               True
L3:
             result = -1
                                                                                    entrada x = -3
                                                                      False
L4:
         elif x>0:
                                                                                    salida esperada = -1
                                          result = -1
                                     L3
                                                      L4
                                                            elif x>0:
L5:
             result = 1
                                                          Test Case #3: valorCero
L6:
        return result
                                                      L5
                                                           result = 1
                                                                                    entrada x = 0
                                                                                    salida esperada = 0
                                                return result
                                               Control Flow-Graph
                                                                                      Test Suite
            Implementación
```

- 1. Escribir los diagramas de control de flujo (control-flow graph) para cantidadDePrimos y la función auxiliar esPrimo.
- 2. Escribir un test suite que cubra todos las líneas de programa del programa cantidadDePrimos y esPrimo.
- 3. En caso que el test suite del punto anterior no cubriera todo los branches del programa, extenderlo de modo que logre cubrirlos.

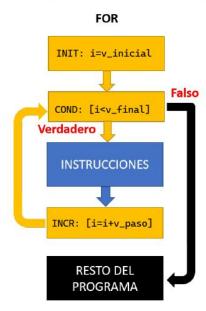
```
problema cantidadDePrimos (in n: \mathbb{Z}) : \mathbb{Z} { requiere: \{n \geq 0\} asegura: \{result = \sum_{i=2}^{n} (\text{if } esPrimo(i) \text{ then } 1 \text{ else } 0 \text{ fi})\}
```

```
def cantidadDePrimos(n: int) -> int:
                                                        #Funcion auxiliar
L1:
       result: int = 0
                                                        def esPrimo(x: int) -> bool:
L2, L3, L4: for i in range (2, n+1, 1):
                                                         result: bool = True
                                               L9:
L5:
       inc: bool = esPrimo(i)
                                               L10, L11, L12: for i in range (2, x, 1):
L6:
              if inc=True:
                                                                  if x \% i == 0:
                                               L13:
L7:
                 result += 1
                                               L14:
                                                                     result = False
L8:
           return result
                                               L15:
                                                               return result
```

Control Flow Graph [FOR]

 $si v_paso > 0$

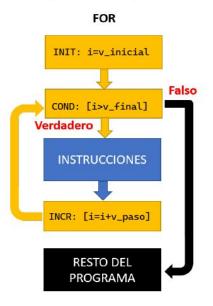
for i in range(v_inicial, v_final, v_paso)



Control Flow Graph [FOR]

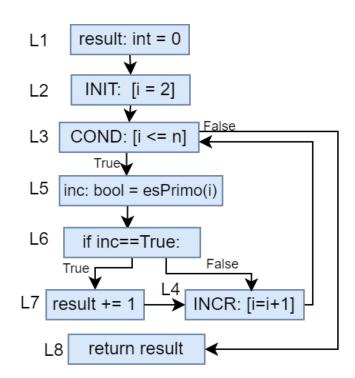
si v_paso < 0

for i in range(v_inicial, v_final, v_paso)



SOLUCIÓN

1. Escribir los CFG para cantidadDePrimos y la función auxiliar esPrimo.



cantidadDePrimos

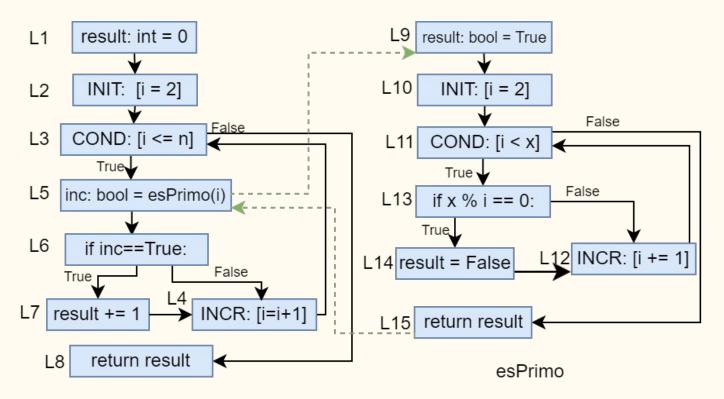
1. Escribir los CFG para cantidadDePrimos y la función auxiliar esPrimo.

```
L10
                                                           INIT: [i = 2]
                                                                          False
          #Funcion auxiliar
                                                     L11 COND: [i < x]
           def esPrimo(x: int) -> bool:
                                                           True 🗸
L9:
                  result: bool = True
                                                                        False
                                                     L13 if x \% i == 0:
                  for i in range (2, x, 1):
L10, L11, L12:
                      if x \% i == 0:
L13:
                                                         True
                         result = False
L14:
                                                                     L12 INCR: [i += 1]
                                                   L14 result = False
L15:
                  return result
                                                    L15 return result
```

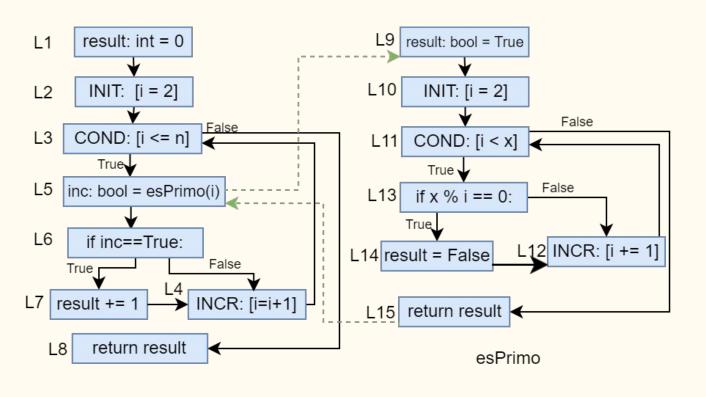
esPrimo

result: bool = True

1. Escribir los CFG para cantidadDePrimos y la función auxiliar esPrimo.



2. Escribir un test suite que cubra todos las líneas de programa del programa cantidadDePrimos y esPrimo.



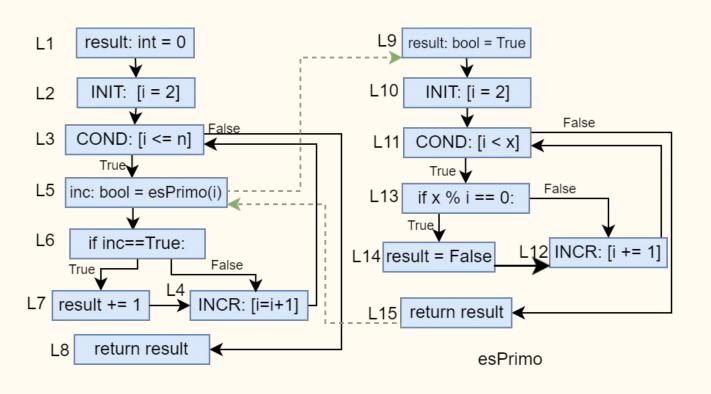
Test Case #1: primo entrada n = 3 salida esperada = 2

Test Case #2: noPrimo entrada n = 4 salida esperada = 2

Test Suite

cantidadDePrimos

2. Escribir un test suite que cubra todos las líneas de programa del programa cantidadDePrimos y esPrimo.



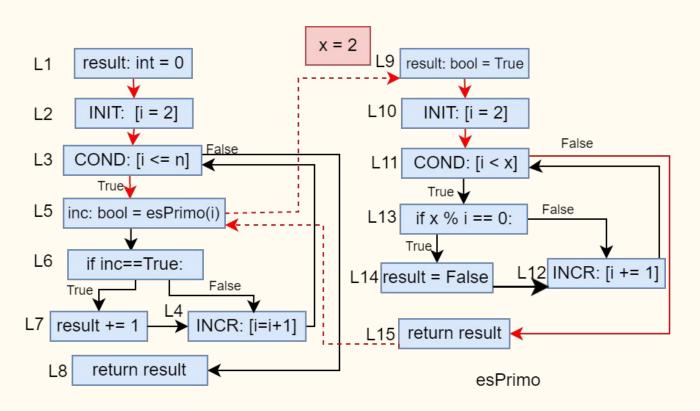
Test Case #1: primo entrada n = 3 salida esperada = 2

Test Case #2: noPrimo entrada n = 4 salida esperada = 2

Test Suite

cantidadDePrimos

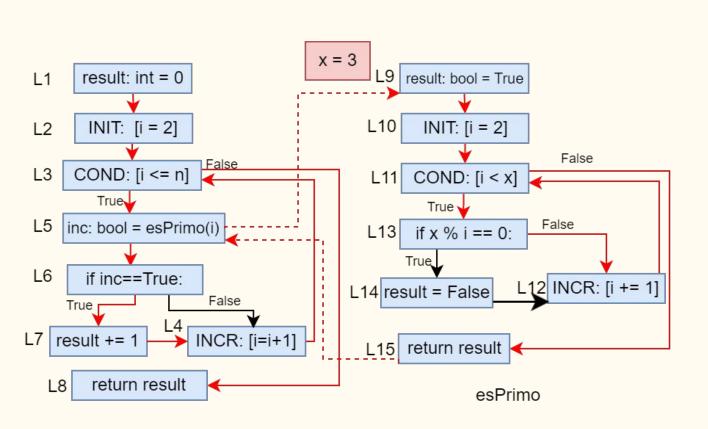
"detenemos" la "ejecución" cuando se ejecutó esPrimo(2)



Test Case #1: primo entrada n = 3 salida esperada = 2

Test Case #2: noPrimo entrada n = 4 salida esperada = 2

Test Suite

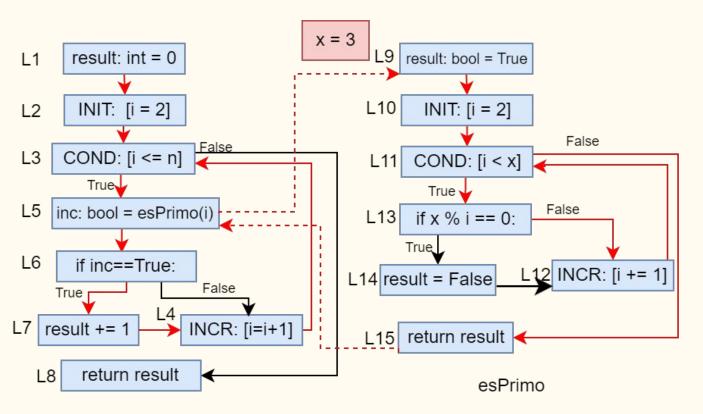


Test Case #2: noPrimo entrada n = 4 salida esperada = 2

Test Suite



"detenemos" la "ejecución" cuando se ejecutó esPrimo(3). Hasta acá es igual que antes. Ahora continuamos con x=4

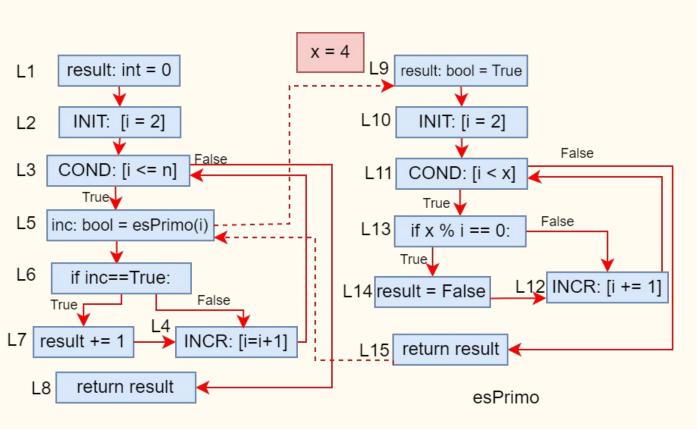


Test Case #1: primo entrada n = 3 salida esperada = 2

Test Case #2: noPrimo entrada n = 4 salida esperada = 2

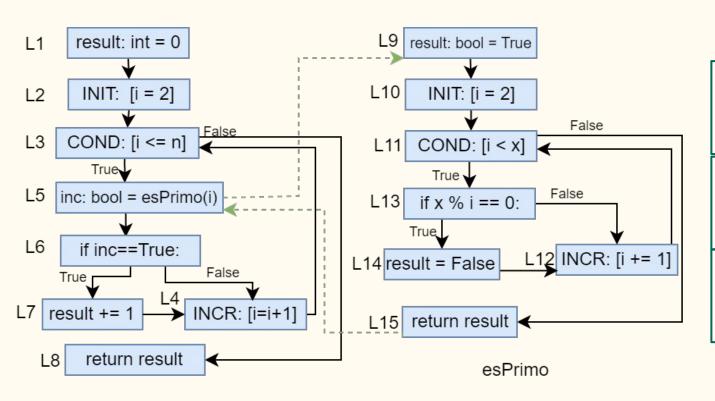
Test Suite

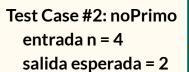
cantidadDePrimos



Test Case #2: noPrimo entrada n = 4 salida esperada = 2

Test Suite

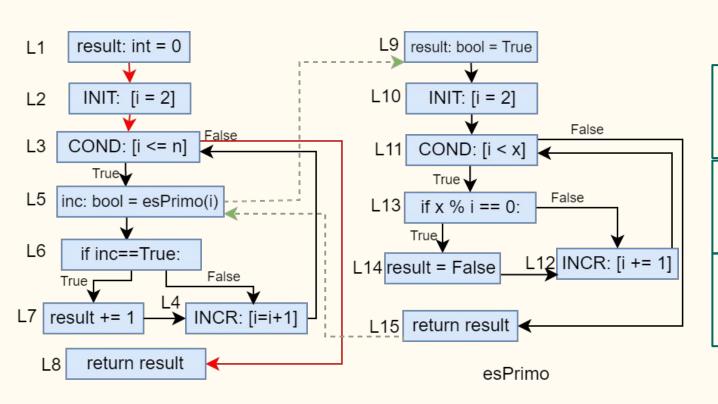




Test Case #2: noPrimo entrada n = 1 salida esperada = 0

Test Suite

cantidadDePrimos



Test Case #2: noPrimo entrada n = 4 salida esperada = 2

Test Case #2: noPrimo entrada n = 1 salida esperada = 0

Test Suite

cantidadDePrimos

3.	En caso que el test suite del punto anterior no cubriera todo los branches del
pro	ograma, extenderlo de modo que logre cubrirlos.

¡Cubrimos todas las líneas y branches con los casos de test anteriores!

Python Tutor

- Permite ver la ejecución de un programa escrito en Python (entre otros lenguajes) siguiendo el "paso a paso" de forma visual.
- Para ver el ejemplo anterior, ingresar <u>acá</u>.
 - Click en "Visualize Execution" y luego en "Next"

```
result += 1
          return result
     def esPrimo(x: int) -> bool:
          result: bool = True
 10
         for i in range(2, x, 1):
 11
              if x % i == 0:
 13
                  result = False
 14
         return result
 15
     print(cantidadDePrimos(4))
Visualize Execution
                  NEW: subscribe to our YouTube for
```

```
def cantidadDePrimos(n: int) -> int:
                                                                                                                                      result: int = 0
                                                                                                                                      for i in range(2,n+1,1):
                                                                                                                                                              inc: bool = esPrimo(i)
                                                                                                                                                              if inc==True:
                                                                                                                                                                                         result += 1
                                                                                                                                  return result
                                                                                     9 def esPrimo(x: int) -> bool:
                                                                                                                                    result: bool = True
                                                                                                                                    for i in range(2, x, 1):
                                                                                                                                                              if x % i == 0:
                                                                                                                                                                                       result = False
                                                                                                                                  return result
                                                                                15
                                                                                  16 print(cantidadDePrimos(4))
                                                                                                                                                                    Edit this code
- line that just executed
 - next line to execute
                                                                                                                                                                                                                                                                     Last >>
value de la companya del companya de la companya del companya de la companya de l
```

Bonus

Ejercicio 17. El programa que se transcribe más abajo pretende determinar la longitud del *fragmento* más largo en un texto. Los fragmentos son porciones del texto que no contienen punto y coma. Por ejemplo, en el siguiente texto el fragmento más largo es "Mercurio", de ocho letras:

"Mercurio; Venus; Tierra; Marte; Júpiter"

- 1. Escribir el CFG (control-flow graph) de la función.
- 2. Escribir un test que encuentre el defecto presente en el código. Es decir, escribir una entrada que cumple con el requiere pero que el resultado de ejecutar el código no cumple el asegura (Justificar la respuesta).
- 3. Agregar casos de test para cubrir todos los branches del programa.

Bonus - Especificación

```
 \begin{array}{l} {\rm problema\ calcularFragmentoMásLargo}\ (s:\ seq\langle {\rm Char}\rangle): \mathbb{Z}\ \{ \\ {\rm requiere:\ } \{True\} \\ {\rm asegura:\ } \{(\exists i,j:\mathbb{Z})({\rm esFragmento}(s,i,j) \wedge result \leq j-i) \wedge (\forall i,j:\mathbb{Z})({\rm esFragmento}(s,i,j) \longrightarrow result \geq j-i)\} \\ {\rm pred\ esFragmento\ } (s:\ seq\langle {\rm Char}\rangle,\ i,j:\mathbb{Z})\ \{ \\ {\rm 0} \leq i \leq j \leq |s| \wedge_L\ (\forall k:\mathbb{Z})(i \leq k < j \longrightarrow_L \neg {\rm esPuntoYComa}(s[k])) \\ {\rm pred\ esPuntoYComa\ } (c:\ {\rm Char})\ \{ \\ {\rm c=\ ';'} \\ {\rm } \} \end{array}
```

Bonus - Implementación

```
def calcular_fragmento_mas_largo(s: str) -> int:
        result: int = 0
        i: int = 0
        f: int = 0 \# Longitud del fragmento actual
        while i < len(s):
                if f > result:
                        result = f
                if s[i] == ';':
                        f = 0
                else:
                        f+=1
                i+=1
        return result
```