Tecnología Digital 1: Introducción a la Programación - TP1

Autores: Kuschnir, Lucian
Ledesma, Benjamín
Mana, Gianni

Función cantidad_de_0:

i) En kdominante-testing.py. Primero antes de escribir cualquier caso, pensamos en todos los casos posibles, con esto llegamos a la conclusión de distintos casos. Estos incluyen: casos de solo unos o ceros , casos mezclando ambos, solo str, otros donde mezclamos str(int) con int y caso de str vacío. A partir de la línea 15 utilizamos la función assertNotEqual para aplicar todas las funciones vistas en clase que puedan ser utilidad, donde seguimos los mismos principios que a los puntos anteriores, donde pusimos los mismos ejemplos pero con un distinto return para demostrar que nuestros primeros casos eran correctos.

ii) e)

Terminación de ciclo: El ciclo termina cuando la condición de la línea 10 se vuelve falsa. (i < len(s)) cosa que es inevitable ya que, len(s) es un valor fijo ≥ 0, i comienza valiendo 0 y aumenta en 1 su valor por cada iteración. Por lo que eventualmente i va a alcanzar el valor de len(s) dando como resultado la falsedad de la condición y la finalización del ciclo.

Predicado invariante: Luego de k iteraciones: cantcero vale la cantidad de ocurrencias de 0 en el string s hasta la posición i-1 inclusive, i vale k

Correctitud:

En general para cualquier valor del parámetro s,
Luego de las líneas 8 y 9, cantcero vale 0; i vale 0.
Llegamos al while de las líneas 10-13. Para k = 0,1,2,...len(s):

> Luego de k iteraciones: cantcero vale la cantidad de ocurrencias de 0 en el string s hasta la posición i-1 inclusive, i vale k

Como ya vimos, en algún momento la condición es False y el ciclo termina.
Esto ocurre cuando i vale len(s), equivalentemente cuando k vale len(s).
Entonces en la línea 14 cantcero vale la cantidad de ‘0’ que aparecen en s hasta la posición len(s) sin incluir.
Dicho de otra manera, cantcero vale la cantidad de ‘0’ que aparecen en s, que es exactamente el Devuelve de la especificación.

Función cantidad_de_1:

i) En kdominante-testing.py. Primero antes de escribir cualquier caso, pensamos en todos los casos posibles, con esto llegamos a la conclusión de distintos casos. Estos incluyen, casos de solo unos o ceros , casos mezclando ambos, solo str, otros donde mezclamos str(int) con int y luego a partir de la línea 25 utilizamos la función assertNotEqual para aplicar todas las funciones vistas en clase que puedan ser utilidad, donde seguimos los

mismos principios que a los puntos anteriores, donde pusimos los mismos ejemplos pero con un distinto return para demostrar que nuestros primeros casos eran correctos.

ii) e)

Terminación de ciclo: El ciclo termina cuando la condición de la línea 24 se vuelve falsa. (i < len(s)) cosa que es inevitable ya que, len(s) es un valor fijo ≥ 0, i comienza valiendo 0 y aumenta en 1 su valor por cada iteración. Por lo que eventualmente i va a alcanzar el valor de len(s) dando como resultado la falsedad de la condición y la finalización del ciclo.

Predicado invariante: Luego de k iteraciones: cantuno vale la cantidad de ocurrencias de 1 en el string s hasta la posición i-1 inclusive, i vale k

Correctitud:
En general para cualquier valor del parámetro s,
Luego de las líneas 22 y 23 , cantuno vale 0; i vale 0.
Llegamos al while de las líneas 24-27 . Para k = 0,1,2,...len(s):

> Luego de k iteraciones: cantuno vale la cantidad de ocurrencias de 1 en el string s hasta la posición i-1 inclusive, i vale k

Como ya vimos, en algún momento la condición es False y el ciclo termina.
Esto ocurre cuando i vale len(s), equivalentemente cuando k vale len(s).
Entonces en la línea 29 cantuno vale la cantidad de '1' que aparecen en s hasta la posición len(s) sin incluir.
Dicho de otra manera, cantuno vale la cantidad de '1' que aparecen en s, que es exactamente el Devuelve de la especificación.

Función es_kdominante:

i) En kdominante-testing.py: A diferencia de las demas funciones, al retornar un bool utilizamos las funciones "assertTrue" y "assertFalse", como aclaración si pueden ser utilizadas assertEqual y su contraparte pero intentamos utilizar todas las funciones dadas en clase. Proporcionamos casos donde: existan sólo unos y solo ceros y otro donde existen ambos en su conversión binaria. Luego pasamos al assertFalse para dar contra resultados de los recién dados, utilizamos los mismos ejemplos para demostrar como justificación.

ii) No incluye ciclos directamente, pero utiliza otras funciones ya definidas y justificadas en este archivo.

Función cero_dominantes_entre:

i) En kdominante-testing.py. Consideramos todos los casos posibles para testear, estos son: sin argumentos (es decir, empieza donde termina, ejemplo 1,1), entre ellos no hay ninguno más que el mismo 1 por lo que debería retornar una lista vacía al no tener 0 dominantes. Un caso más cotidiano que sería entre 1 y 17 incluido, para demostrar que no tiene límites la lista que retorna todos los 0 dominantes. Otro ejemplo con el mismo objetivo de recién pero utilizando 10 hasta 32. Este ejemplo es para demostrar más ejemplos utilizando un bit de más, el número 32 posee un bit más que el 17 (también es 0 dominante) por lo que mostramos que independientemente la cantidad de bits nuestra

función, corre bien. Luego utilizamos el assertNotEqual para comprobar dos casos, el primero donde tenemos un solo elemento que es 0 (antes usamos el 1, pero con equal), por lo que considera el caso de 0 hasta 0. Donde n es igual a 0 bin = 0 por lo que debería retornar [0]. Y por último elegimos un caso donde tampoco tenemos 0 dominantes entre 1 y 1 incluido, donde 1 es uno dominante, independientemente lo que pongamos en la lista nunca retornará un error.

ii) e)
Terminación de ciclo: El ciclo termina cuando la condición de la línea 55 se vuelve falsa. (i <= m) cosa que es inevitable ya que, m es un valor fijo N, i comienza valiendo n y aumenta en 1 su valor por cada iteración. Por lo que eventualmente i va a superar el valor de m dando como resultado la falsedad de la condición y la finalización del ciclo.

Predicado invariante: Luego de k iteraciones: vr es una lista que contiene todos los números cero dominantes entre n e i inclusive, i vale k+n

Correctitud:
En general para cualquier valor del parámetro n y m,
Luego de las líneas 53 y 54 , vr vale []; i vale n.
Llegamos al while de las líneas 55-58 . Para (i <=m):

> Luego de k iteraciones: vr es una lista que contiene todos los números cero dominantes entre n e i inclusive, i vale k+n

Como ya vimos, en algún momento la condición es False y el ciclo termina.
Esto ocurre cuando i>m, equivalentemente cuando i es más grande que m.
Entonces en la línea 59 vr vale la lista de números 0-dominantes entre n y m.
Dicho de otra manera, vr vale la lista de números 0-dominantes entre n y m, que es exactamente el Devuelve de la especificación.

Funcion suma_cero_dominantes_menores:

i) En kdominante-testing.py. Consideramos los casos: donde retorna un solo valor por lo que la suma no existiría y simplemente retorna la variable 4, donde también el siguiente dominante sería 8 pero al no estar incluido en el ciclo while, no lo considera. Segundo caso donde tenemos dos valores 0 dominantes menores a 10 por lo que se ejecutaría una suma, esta sería 4 + 8 = 12. Luego para demostrar que sin importar la cantidad de 0 dominantes existan, nuestro código funciona correctamente, lo ejecutamos con los estrictamente menores a 20, son 5 números que su suma es igual a 63. Y luego un caso extra para demostrar otro return de un solo valor que es igual a 0, por lo tanto la suma de cero dominantes estrictamente menores a 3 es 0, donde 0 es el único dominante. A partir de este punto pasamos a los assertNotEqual, donde ejecutamos los mismos valores que los casos anteriores pero cambiando lo de que devolvería para demostrar que nuestros casos anteriores son correctos.

ii) e)
Terminación del ciclo: El ciclo termina cuando la condición de la línea 68 se vuelve falsa.

(i < n) cosa que es inevitable ya que, n es un valor fijo ≥ 0, i comienza valiendo 0 y aumenta en 1 su valor por cada iteración. Por lo que eventualmente i va a alcanzar el valor de n dando como resultado la falsedad de la condición y la finalización del ciclo.

Predicado invariante: Luego de k iteraciones: vr vale la sumatoria de los valores que son cero dominantes en su forma binaria hasta i inclusive; i vale k.

Correctitud:
En general para cualquier valor del parámetro n,
Luego de las líneas 66 y 67 , vr vale 0; i vale 0.
Llegamos al while de las líneas 68-71 . Para (i <n):

> Luego de k iteraciones: vr vale la sumatoria de los valores que son cero dominantes en su forma binaria hasta i inclusive; i vale k.

Como ya vimos, en algún momento la condición es False y el ciclo termina.
Esto ocurre cuando i=n, equivalentemente cuando i vale n.
Entonces en la línea 72 vr vale la suma de todos los números 0-dominantes estrictamente menores que n.
Dicho de otra manera, vr vale la lista de números 0-dominantes entre n y m, que es exactamente el Devuelve de la especificación.