departamento de ingeniería e investigaciones tecnológicas UNLAM

2017

Prueba de Caja Negra

TRABAJO PRÁCTICO N 1

Maximiliano Pérez Coto, Pablo López, Fabián Norberto Escobar, Dimas Aguirre

Programación Avanzada

ÍNDICE

[ÍNDICE 2](#_Toc480295309)

[INTRODUCCIÓN 3](#_Toc480295310)

[EL PROBLEMA 4](#_Toc480295311)

[DESARROLLO 6](#_Toc480295312)

[INFORME DE FALLAS 6](#_Toc480295313)

[CONCLUSIONES DE LAS FALLAS 11](#_Toc480295314)

INTRODUCCIÓN

Las pruebas de caja negra son pruebas funcionales dedicadas a mirar en el exterior de lo que se prueba, en este caso, software. También son conocidas como pruebas de caja opaca, pruebas de entrada/salida, pruebas inducidas por datos. Dichas pruebas se centran principalmente en lo que se quiere para determinar si se atiende a las especificaciones, para solventar problemas a nivel de datos externos.

Estas actividades se limitan a que el tester pruebe con datos de entrada el comportamiento del software y estudie las salidas producidas, sin preocuparse de lo que ocurre en su interior, ya que de eso se tratan las cajas negras: elementos que son estudiados desde el punto de vista de las entradas que recibe y las salidas que produce, desconociendo su funcionamiento interno. Lo que nos interesa de las cajas negras es la forma de interactuar con el medio que las rodea, entendiendo qué es lo que hacen pero sin darle importancia a cómo lo hacen. Por lo tanto, deben estar muy bien definidas sus entradas y salidas, lo que conforma la interfaz.



Suelen centrarse en módulos de interfaz de usuario, como ser pantallas, ficheros, canales de comunicación, pero suelen ser útiles en cualquier módulo ya que todos o la mayoría de los módulos tiene datos de entrada y de salida que pueden ser erróneos, y por lo tanto, es necesario comprobar y verificar. Como cualquier prueba, se apoyan y basan en la especificación de los requisitos y documentación funcional, requisitos que suelen ser más complejos que los requisitos internos, por lo que se realiza una cobertura de especificación para conseguir probar el mayor campo que sea posible.

Las pruebas funcionales se dedican a comprobar que los sistemas informáticos funcionan acorde a las especificaciones y requisitos del cliente, y es de gran ayuda para detectar los posibles defectos derivados de errores en la fase de programación. Los defectos encontrados y solventados durante la fase de pruebas supondrán un ahorro en tiempo y dinero y una reducción en los riesgos. En muchos casos, se hace necesario llevar a cabo pruebas de integración, ya que siempre pueden existir defectos que no se están teniendo en cuenta, y ellos acarrearán problemas ya sea en corto o largo plazo.

Si el sistema se encuentra formado por módulos que cumplan con las características de caja negra, será más sencillo de entender ya que permitirá dar una visión más clara del conjunto. El sistema también será más robusto y fácil de mantener, ya que, en caso de ocurrir un fallo, éste podrá ser aislado y abordado ágilmente.

EL PROBLEMA

El problema que vamos a analizar en este caso se llama **Número Escondido** y consiste en lo siguiente. Se debe encontrar una secuencia de dígitos dentro de otra. Es decir, dado una número de hasta N cifras y otro de hasta M cifras, se pide encontrar la posición dentro del primero, a partir de la cual comienza a leerse el segundo número buscado. Se considera como segundo número a encontrar a cualquier combinación de las M cifras del número indicado en segundo lugar. En caso de no encontrarlo, se informa simplemente NO. En caso de repetirse, se informa SI y se muestra cuantas repeticiones tiene y sus respectivas ubicaciones.

**Datos de entrada:**

Se recibe un archivo entrada.in donde se indica:

• Primera línea: La cantidad N de dígitos del primer número (1 <= N <= 250).

• Segunda línea: un número de N dígitos.

• Tercera línea: La cantidad M de dígitos del segundo número (1<=M<=N).

• Cuarta línea: un número de M dígitos.

**Datos de salida:**

El programa debe generar el archivo salida.out, en el directorio actual con:

*1. En caso de contener el número buscado*

• Primera línea: SI (sin acento) y la cantidad de veces separados por un blanco.

• Segunda línea: la/las posiciones donde comienza el número buscado, separados por un blanco.

*2. En caso de no encontrarlo, una línea que indique “NO”*

**Ejemplo:**

A continuación, veamos algunos ejemplos para entender mejor el problema.

Si el archivo de entrada fuese:

|  |
| --- |
| 8  16345678  4  3456 |

El archivo de salida debería ser:

|  |
| --- |
| SI 2  2 3 |

Si el archivo de entrada fuese:

|  |
| --- |
| 8  87654321  4  5462 |

El archivo de salida debería ser:

|  |
| --- |
| NO |

Si el archivo de entrada fuese:

|  |
| --- |
| 8  87654123  4  1234 |

El archivo de salida debería ser:

|  |
| --- |
| SI 1  5 |

Por último, si el archivo de entrada fuese:

|  |
| --- |
| 10  87654231234  3  234 |

El archivo se salida debería ser:

|  |
| --- |
| SI 2  5 9 |

Con los ejemplos anteriores, ha quedado más clara la estructura de los archivos de entrada y de salida, y cómo están relacionados.

DESARROLLO

Se cuenta con nueve archivos ejecutables (A.exe, B.exe, C.exe, D.exe, E.exe, F.exe, G.exe, H.exe e I.exe) que automatizan la resolución del problema, cada uno con una implementación diferente, la cual nos resulta desconocida y no nos interesa para realizar las pruebas, y con los casos de prueba comentados en la sección anterior.

Se cuenta también con los archivos de entrada y de salida, separados en carpetas por el nombre de caso, y con la documentación del lote de prueba.

Se procede a analizarlos de a uno por vez, ejecutándolo con el archivo de entrada de cada uno de los casos de prueba, donde se produce una salida que es comparada con el archivo de salida esperado. Si coinciden las salidas, entonces el ejecutable funciona correctamente con ese caso de prueba.

Una vez finalizadas todas las pruebas con un ejecutable, se pasa al siguiente. Se anotará cada falla en el informe de fallas por cada ejecutable y luego se redactarán las conclusiones de la prueba de esta prueba de caja negra.

Si durante el análisis de los ejecutables se considera que es necesario realizar nuevos casos de prueba para el lote, así se hará y se comentará de su existencia en el informe de fallas.

INFORME DE FALLAS

Ya realizadas las pruebas de todos los casos del lote con los nueve archivos ejecutables, se procede a detallar las fallas que acontecieron.

**Ejecutable A.exe:**

No se ha podido detectar casos en que este ejecutable resuelva mal el problema planteado, testeado con muchos de los casos con que se testearon los otros.

**Ejecutable B.exe:**

*Observaciones:* El programa no es capaz de reconocer números que no estén en la misma secuencia, comenzando desde cualquier posición. Para cualquier número ABCDEFG, encontrará cualquier número escondido tal que comience por cualquiera de sus dígitos, y lo recorra como una lista enlazada, por ejemplo, BCDE, FGAB, etc.

**Ejecutable C.exe:**

No se ha podido detectar casos en que este ejecutable resuelva mal el problema planteado.

**Ejecutable D.exe:**

**03\_AmbosTienenUnDígitoYSonIguales:**

*Observaciones:* El programa no es capaz de reconocer valores únicos a pesar de que sean iguales. Curiosamente **N = M = 1**, cuya resta da 0. Si se modifica la entrada dejándola incorrecta a propósito para que esta diga que **N = 2**, sin cambiar más nada, la salida resulta correcta, presuntamente dado a que **N – M = 1.**

**09\_ElNúmeroContenedorEstáCompletamenteFormadoPorAparicionesDelNúmeroEscondido:**

*Observaciones:* Se observa que en este caso el ejecutable no encuentra la última aparición del valor escondido. Esto podría deberse a que solo realiza la búsqueda hasta la posición **N – M**, lo cual sería igual a **12 – 3 = 9**. Por lo tanto, si la búsqueda solo se realizara hasta el dígito en la posición 9, las posiciones 10, 11 y 12 nunca serían leídas, obviando la última aparición del número escondido.

**12\_ElSegundoEstáIncluidoExactamenteIgualUnaVez:**

*Observaciones:* En este caso se da un comportamiento equivalente al anterior, el programa solo busca el número desde la posición **1** hasta la **5** (**8 – 3, es decir, N – M)**.

**13\_ElSegundoEstáIncluidoExactamenteIgualVariasVeces:**

*Observaciones:* Ídem anterior, solo que esta vez el valor escondido se encuentra repetido dentro del rango de búsqueda, el cual por supuesto es encontrado por el programa.

**19\_MismaCantidadDeDígitosMismosDígitos:**

*Observaciones:* Ídem primer caso planteado, solo que con números con mayor cantidad de dígitos.

**21\_TodosLosDígitosDeAmbosSonElMismoDígito:**

*Observaciones:* En este caso se observa un ligero cambio en el comportamiento del programa. Si bien la búsqueda se realizaría hasta la posición 3, se puede ver que el programa logra reconocer la aparición del número 777 incluso si este no está completamente incluido en el rango de búsqueda.

**Ejecutable E.exe:**

*Observaciones:* Presenta un error de formateo de salida. Fuera de ello, no se ha podido encontrar casos en que el problema esté mal resuelto.

**Ejecutable F.exe:**

**12\_ElSegundoEstáIncluidoExactamenteIgualUnaVez:**

*Observaciones:* El programa está tomando como válido un número completamente diferente al que tiene que buscar.

**20\_TodasLasAparicionesDelEscondidoSeSuperponen:**

*Observaciones:* Ídem anterior, se observa que en ambos casos las cifras del número que toma como válido SUMAN lo mismo que las cifras del número a buscar.

**Ejecutable G.exe:**

**06\_ConsiderandoCerosEnAmbos:**

*Observaciones:* Evidentemente el programa nunca llega a ver el número escondido de forma completa.

**09\_ElNúmeroContenedorEstáCompletamenteFormadoPorAparicionesDelNúmeroEscondido:**

*Observaciones:* El ejecutable se saltea valores internos, pero no se puede afirmar que sea incapaz de reconocer valores desordenados o superpuestos.

**10\_ElSegundoEstáIncluidoDeAmbasFormasUnaVez:**

*Observaciones:* El programa es incapaz de reconocer al número a buscar si este está desordenado. Sin embargo, puede ocurrir que este simplemente esté fuera de su rango de búsqueda, es decir, que nunca llegue a “verlo”.

**11\_ElSegundoEstáIncluidoDeAmbasFormasVariasVeces:**

*Observaciones:* Ocurre algo muy similar al caso anterior, solo que esta vez el valor está incluido una mayor cantidad de veces.

**12\_ElSegundoEstáIncluidoExactamenteIgualUnaVez:**

*Observaciones:* En este caso, incluso si el número a buscar se encuentra exactamente de la misma forma en el número contenedor, el programa no es capaz de encontrarlo. Esto nos lleva a pensar que en efecto, el problema no tiene que ver con que los números estén escritos de forma desordenada.

**13\_ElSegundoEstáIncluidoExactamenteIgualVariasVeces:**

*Observaciones:* Similar al caso anterior, solo que esta vez el número está incluido varias veces, nuevamente, escrito de la misma manera.

**14\_ElSegundoEstáIncluidoNoExactamenteIgualUnaVez:**

*Observaciones:* En este caso el programa no encuentra el número. Se puede destacar que el número tiene los dígitos desordenados, pero en base a observaciones anteriores esto no parece relevante.

**15\_ElSegundoEstáIncluidoNoExactamenteIgualVariasVeces:**

*Observaciones:* Finalmente el programa es capaz de encontrar un número desordenado, descartando casi por completo la incapacidad de este de identificarlos. Se considera que el problema está relacionado a un rango de búsqueda en específico.

**20\_TodasLasAparicionesDelEscondidoSeSuperponen:**

*Observaciones:* Nuevamente el programa se saltea los valores superpuestos. Como ya sabemos que no tiene imposibilitado reconocer valores escritos de forma desordenada, se asume que está definiendo un rango de búsqueda con la información que tiene disponible en la entrada, lo cual está provocando que el programa obvie determinados números que son válidos.

**21\_TodosLosDígitosDeAmbosSonElMismoDígito:**

*Observaciones:* El problema que tiene el programa para identificar valores superpuestos persiste. Su rango de búsqueda evidentemente hace que se saltee dichos valores. Curiosamente este rango puede estar dado por el número de dígitos **M (cantidad de dígitos del número escondido)**. Esta deducción verifica para todos los casos anteriores.

**Ejecutable H.exe:**

No se encontraron fallas en el ejecutable H.

**Ejecutable I.exe:**

**Casos varios de prueba:**

*Observaciones:* No se detectaron problemas, salvo con números de búsqueda de longitud >= 9

CONCLUSIONES DE LAS FALLAS

**Ejecutable B**

Considera escondidos válidos sólo los que se leen como lista enlazada, no así los que tienen dígitos desordenados.

**Ejecutable D**

Se llegó a la conclusión de que el ejecutable D realiza la búsqueda del número escondido tomando como rango de dígitos los que se encuentren hasta la posición dada por **N – M**, siendo **N** la cantidad de dígitos del número contenedor y **M** la cantidad de dígitos del número escondido. No tiene problemas para validar casos de superposición, dígitos en distinto orden o números que estén separados entre sí, siempre y cuando se encuentren dentro del rango de búsqueda que toma como válido, teniendo como condición que como mínimo un dígito del número a buscar debe estar dentro de dicho rango. Esto provoca que el programa sea incapaz de encontrar cualquier número que se encuentre **COMPLETAMENTE** fuera del rango de búsqueda.

**Ejecutable E**

Error de formato en la salida.

**Ejecutable F**

Evidentemente el programa está realizando una validación por **SUMA**, la cual consiste en tomar como válido cualquier valor cuyos dígitos sumen la misma cantidad que los dígitos del número a buscar. Es claro que esto se hizo con el objetivo de facilitar la identificación de valores con dígitos escritos de forma desordenada, pero a su vez provoca que valores completamente diferentes al buscado sean tomados como correctos.

**Ejecutable G**

Lo que está haciendo el ejecutable G es tomar la cantidad de dígitos **M** que tiene el valor escondido, y usar esa cantidad para recorrer el número con cantidad de dígitos **N**. En otras palabras, lo que hace es “partir” el número con **N** dígitos en partes de **M** dígitos (que no se superponen) y valida cada una de esas partes teniendo en cuenta que el número escondido puede tener los dígitos desordenados. Esto provoca que se saltee valores internos que son válidos, lo cual queda especialmente claro cuando se da un caso como **467467467** **(N=9)** siendo el número a buscar **467 (M=3)** ya nunca llega a tener en cuenta el valor **674 (el cual es válido)** debido a que no es un número que forme parte del recorrido de validación que el programa realiza, lo cual es erróneo.

**Ejecutable I**

El algoritmo empleado no permite utilizar palabras mayores a 9 dígitos en la búsqueda. Esto se debe a la cantidad de combinaciones posibles a encontrar que sobrecargan o hacen extremadamente largo el procesamiento de la búsqueda.