

**UNIVERSIDAD NACIONAL DE LA MATANZA**

Departamento de Ingeniería e Investigaciones Tecnológicas

***PROGRAMACION AVANZADA***

**Jefe de Cátedra: Verónica Aubin**

**Jefe de TP: Leonardo Blautzik**

**Docentes: Lucas Videla, Lucas Ponce de León**

**TP 1 - PRUEBA DE SOFTWARE“Búsqueda numérica”**

**ENTREGA 04/09/2015**

**GRUPO Nº 8**

**INTEGRANTES**

* BLANCO, JUAN IGNACIO DNI. 35272529
* DIGIACOMO, GASTON DNI. 36075262
* MIRANDA, CRISTIAN NAHUEL DNI. 35349257
* MARTIN, GONZALO JAVIER DNI. 36170285

# OBJETIVO

Recibimos el enunciado de un problema y distintos programas ejecutables que, supuestamente, resuelven el problema.

Debemos diseñar e implementar distintos casos de prueba que permitan testear en profundidad el funcionamiento del software entregado, que sean lo suficientemente ingeniosos y atómicos para poder decir de la manera más certera posible, el o los errores cometidos en la implementación de cada ejecutable.

Además elaboraremos un informe detallado de los resultados obtenidos, indicando los errores observados.

# ESTRUCTURA DEL DIRECTORIO

## TESTING

Inicialmente en la carpeta raíz verán los siguientes archivos:

* **Informe.docx🡪** documentación del TP.
* **Enunciado.pdf 🡪** enunciado del TP, brindado por la cátedra.
* **PlanillaDeMetricas.xlsx 🡪**Planilla de métricas de tiempos para dicho TP.
* **ResultadoPruebas.xlsx 🡪**Tabla de comparación de resultados de cada caso de prueba, con cada ejecutable.

Luego ya que los ejecutables tomaban como datos de entrada el archivo que se encontrara en el mismo directorio que estos, con el nombre de “estrada.in”, es que definimos la siguiente estructura para las pruebas:

Dentro de la carpeta ***LotesDePrueba*** encontraran una carpeta para cada caso de prueba, es decir, ***Caso01; Caso02;…; CasoNN***.

A su vez dentro de cada carpeta de los casos estarán los 9 ejecutables, con su “entrada.in” correspondiente. Y cada vez que ejecuten algún .exe se ira generando el archivo de salida “salida.out” donde deberán ir viendo los resultados 1 por 1.

* LotesDePrueba
  + CasoNN
    - entrada.in
    - A.exe
    - B.exe
    - C.exe
    - D.exe
    - E.exe
    - F.exe
    - G.exe
    - H.exe
    - I.exe

## ENTREGABLE

Dado que la entrega será vía email y la estructura anterior es demasiado pesada en el archivo .rar lo haremos de la siguiente manera:

* Carpeta IN 🡪 con todos los archivos de entrada.in para cada caso de prueba.
* Carpeta OUT 🡪 con todos los archivos salida.out para cada caso de prueba y cada .exe.
  + Carpeta CasoNN
    - Archivo salidaEJECUTABLE.out
* Archivo Informe.docx
* Archivo PlanillaDeMetricas.xlsx
* ResultadoPruebas.xlsx

# LOTE DE PRUEBAS DE ERROR

## 00 – Enunciado

Primer ejemplo del enunciado, N de longitud 8 y M de longitud 4.

|  |  |
| --- | --- |
| Entrada | Salida |
| 8 | SI 2 |
| 16345678 | 2 3 |
| 4 |  |
| 3456 |  |

**Resultado para cada ejecutable:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I |
| OK | OK | OK | OK | SI  2 2 3 | OK | NO | OK | SI 2  3 2 |

## 01 – Único digito para M

La cantidad de N dígitos será igual a 10, mientras que la cantidad de dígitos para M será igual a 1

|  |  |
| --- | --- |
| Entrada | Salida |
| 10 | SI 2 |
| 1628394954 | 6 8 |
| 1 |  |
| 9 |  |

**Resultado para cada ejecutable:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I |
| OK | OK | OK | OK | SI  2 6 8 | OK | OK | OK | OK |

## 02 – Único digito duplicado para M

La cantidad de N dígitos será igual a 10, mientras que la cantidad de dígitos para M será igual a 2, siendo iguales estos últimos 2.

|  |  |
| --- | --- |
| Entrada | Salida |
| 10 | NO |
| 1628394954 |  |
| 2 |  |
| 99 |  |

**Resultado para cada ejecutable:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I |
| OK | OK | OK | OK | OK | OK | OK | OK | OK |

## 03 – Combinación N igual a Combinación M

Tanto la cantidad como la combinación de dígitos es la misma, para la expresión N y M.

|  |  |
| --- | --- |
| Entrada | Salida |
| 10 | SI 1 |
| 1628394954 | 1 |
| 10 |  |
| 1628394954 |  |

**Resultado para cada ejecutable:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I |
| OK | OK | OK | NO | SI  1 1 | OK | OK | OK | QUEDA COLGADA LA APLICACIÓN Y GENERA LA SALIDA EN BLANCO |

## 04 – Misma longitud, combinación invertida

La cantidad de dígitos de N y M serán iguales, pero la combinación de M será N a la inversa.

|  |  |
| --- | --- |
| Entrada | Salida |
| 10 | SI 1 |
| 1628394954 | 1 |
| 10 |  |
| 4594938261 |  |

**Resultado para cada ejecutable:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I |
| OK | NO | OK | NO | SI  1 1 | OK | OK | OK | QUEDA COLGADA LA APLICACIÓN Y GENERA LA SALIDA EN BLANCO |

## 05 – Combinación M y N en base 2 (binarios)

Tanto N y M serán una combinación de 1 y 0, diferentes longitudes entre ellos.

|  |  |
| --- | --- |
| Entrada | Salida |
| 5 | SI 4 |
| 10101 | 1 2 3 4 |
| 2 |  |
| 10 |  |

**Resultado para cada ejecutable:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I |
| OK | OK | OK | SI 3  1 2 3 | SI  4 1 2 3 4 | OK | SI 2  1 3 | OK | OK |

## 06 – Combinación M capicúa

La cantidad de dígitos de N es igual a 5, mientras que la de M es igual a 3, siendo este último un número capicúa.

|  |  |
| --- | --- |
| Entrada | Salida |
| 6 | SI 3 |
| 929299 | 1 3 4 |
| 3 |  |
| 929 |  |

**Resultado para cada ejecutable:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I |
| OK | OK | OK | OK | OK | OK | OK | OK | OK |

## 07 –Fatiga con N de Longitud 20

La cantidad de dígitos N es igual a 20, mientras que la de M es igual a 2.

|  |  |
| --- | --- |
| Entrada | Salida |
| 20 | NO |
| 12345678901234567890 |  |
| 2 |  |
| 25 |  |

**Resultado para cada ejecutable:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I |
| OK | OK | OK | OK | OK | SI 2  3 13 | OK | OK | OK |

## 08 – Fatiga con N de Longitud 50

La cantidad de dígitos N es igual a 50, mientras que la de M es igual a 2.

|  |  |
| --- | --- |
| Entrada | Salida |
| 20 | NO |
| 12345678901234567890123456789012345678901478963214 |  |
| 2 |  |
| 25 |  |

**Resultado para cada ejecutable:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I |
| OK | OK | OK | OK | OK | SI 4  3 13 23 33 | OK | OK | OK |

## 09 - Fatiga con N de Longitud 100

La cantidad de dígitos N es igual a 100, mientras que la de M es igual a 2.

|  |  |
| --- | --- |
| Entrada | Salida |
| **“./IN/Caso09.in”** | NO |

**Resultado para cada ejecutable:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I |
| OK | OK | OK | OK | OK | SI 8  3 13 23 33 53 63 73 83 | OK | OK | OK |

## 10 - Fatiga con N de Longitud 250 con éxito

La cantidad de dígitos N es igual a 250, mientras que la de M es igual a 2.

|  |  |
| --- | --- |
| Entrada | Salida |
| **“./IN/Caso10.in”** | SI 1 |
|  | 249 |

**Resultado para cada ejecutable:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I |
| OK | OK | OK | NO | OK | ERROR (\*) | OK | OK | OK |

(\*)SI 21

3 13 23 33 53 63 73 83 103 113 123 133 153 163 173 183 203 213 223 233 249

## 11 - Fatiga con N de Longitud 250 sin éxito

La cantidad de dígitos N es igual a 250, mientras que la de M es igual a 2.

|  |  |
| --- | --- |
| Entrada | Salida |
| **“./IN/Caso11.in”** | NO |

**Resultado para cada ejecutable:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I |
| OK | OK | OK | OK | OK | ERROR (\*) | OK | OK | OK |

(\*)SI 20

3 13 23 33 53 63 73 83 103 113 123 133 153 163 173 183 203 213 223 233

## 12 - Fatiga con N y M de Longitud 250 sin éxito

La cantidad de dígitos N y M es igual a 250, el máximo de dígitos permitido.

|  |  |
| --- | --- |
| Entrada | Salida |
| **“./IN/Caso12.in”** | NO |

**Resultado para cada ejecutable:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I |
| OK | OK | OK | OK | OK | OK | OK | OK | QUEDA COLGADA LA APLICACIÓN Y GENERA LA SALIDA EN BLANCO |

## 13 – N contiene 24 combinaciones posibles de 4 digitos de M

La cantidad de digitos de M es igual a 4, y esta compuesta por numeros distintos. Mientras que N tiene longitud 120, y dentro de ella se encuentran las 24 combinaciones posibles que puede tomar M.

|  |
| --- |
| Entrada |
| **“./IN/13\_M4\_NTodasCombinatoriaPosiblesDe4.in”** |
|  |

|  |
| --- |
| Salida |
| SI 24 |
| 1 6 11 16 21 26 31 36 41 46 51 56 61 66 71 76 81 86 91 96 101 106 111 116 |

**Resultado para cada ejecutable:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I |
| OK | ERROR | ERROR | OK | ERROR | ERROR | ERROR | OK | ERROR |

(\*) Dado que son varios los errores y las salidas algo largas ver los archivos de salida correspondientes a cada ejectuable. Los mismos los podra ver en “LoteDePruebas\OUT\13\_M4\_NTodasCombinatoriaPosiblesDe4”.

# CONCLUSIONES SOBRE LOS ERRORES

|  |  |
| --- | --- |
| **EJECUTABLE** | **ERROR DETECTADO** |
| **A** | No se detectaron errores. Por lo que decimos que el programa funciona correctamente según el enunciado. |
| **B** | Detectamos que cuando M tiene una logitud mayor o igual a 4, el programa solo encuentra ocurrencias comparando M contra N sin evaluar posibles combinaciones de M, es decir, tal cual recibe la combinatoria M la va comparando contra N.  Por ejemplo si mi M fuese “1234” solo encontrara las ocurrencias “1234”. |
| **C** | Detectamos que cuando M tiene una logitud mayor o igual a 4, el programa solo encuentra ocurrencias comparando M contra N variando a M entre lo que se recibe y este mismo en orden inverso.  Por ejemplo si mi M fuese “1234” solo encontrara las ocurrencias “1234” y “4321”. |
| **D** | Detectamos que no recorre correctamente el numero N, ya que el algoritmo se interrumpe antes de llegar a los últimos dígitos.  Por último, este programa se pincha en los casos de fatiga más extremos, entonces estimamos que el algoritmo falla a causa de su poca eficiencia. |
| **E** | Detectamos que este programa lo unico que resuelve mal es la escritura del archivo de salida cuando debe imprimir ocurrencias encontradas. Donde en vez de escribir:  “SI [numeroDeOcurrencias]  posicion1 posicion2 … posicionN”  escribe:  “SI  [numeroDeOcurrencias] posicion1 posicion2 … posicionN” |
| **F** | Detectamos que el programa suma los digitos que componen a M para realizar las comparaciones contra N. Esto puede encontrar las ocurrencias requeridas, pero ademas puede encontrar otras ocurrencias que no corresponden por dar el mismo resultado en la suma.  Por ejemplo si mi M fuese “1234” la suma es 10, lo que encontraria tambien ocurrencias para un M igual a “0235” ya que aquí tambien la suma es 10. |
| **G** | Detectamos que el programa realiza las comparaciones cortando a N en bloques de logitudes iguales a M y va comparando estos bloques contra las combinaciones posibles de M.  Podemos verlo claramente en el caso de prueba de los numeros binarios, donde M es igual a “10” y N “10101”. Entonces parte a N en los siguientes bloques |10|10|1| entonces solo encuentra las ocurrencias de los bloques 10. No por ejemplo las ocurrencias del 01 que comienzan en la posicion 2 y 4. Tambien se puede apreciar en el caso 13 pero es demasiado extenso para citarlo aquí. |
| **H** | No se detectaron errores. Por lo que decimos que el programa funciona correctamente según el enunciado. |
| **I** | Detectamos que el programa realiza las comparaciones de la siguiente forma: toma una combinacion posible de M y busca todas sus ocurrencias, a medida que las encuentra las va guardando. Luego toma otra combinacion de M y busca las ocurrencias, a medida que las encuentra las va guardando. Y asi sucesivamente con cada combinacion posible de M. Ahora bien, cuando va a grabar el archivo no se toma el trabajo de ordenar estas ocurrencias, por lo que en la salida podemos tener las posiciones desordenadas. |

# CONCLUSIONES GENERALES

Tener listo un buen lote de pruebas antes de la construcción del algoritmo, es fundamental, ya que nos hace ahorrar muchísimo tiempo entre el fin de la codificación y la fase de pruebas.

Este lote, no debe incluir cualquier caso de prueba, sino que debe tener aquellos que fuercen al algoritmo a cometer un error. Esta pruebas deben ser lo mas atómicas posibles, es decir, deben apuntar puntualmente a un error, ya que pruebas más genéricas, provocarían fallas, pero sería mucho más difícil de detectar su causa.

Se puede pensar un mejor lote de pruebas mientras más se conozca sobre el problema, y se piense una correcta solución. Al tener una idea de cómo será el algoritmo para la resolución del problema, se pueden diseñar pruebas mas especificas que intenten romperlos.