

Maratón de Programación

Día del Programador

David E. Narvaez
david.narvaez@computer.org

Fecha Límite: 12 de Septiembre de 2008

Resumen

Los siguientes problemas deben ser entregados antes del 12 de Septiembre de 2008 a las 2:00 pm a la dirección de correo que aparece en el encabezado. La calificación de los mismos empezará a esa hora y tomará en cuenta los siguientes aspectos (en orden de importancia): Ejecución Correcta, Tiempo de Ejecución y Estructura de Código. El número máximo de participantes por equipo es de 3 personas y deberán entregar los programas ejecutables y el código fuente en uno de los siguientes lenguajes: C, C++, Java, Perl, PHP o Python. El tiempo máximo de ejecución de cualquier programa en cualquier caso es de 5 segundos. Resolver todos los problemas no es requisito para entregar soluciones y los problemas no tienen que ser todos resueltos en el mismo lenguaje (siempre y cuando el lenguaje empleado sea admitido en la competencia). Para más detalles, escribir a la dirección de correo que aparece en el encabezado.

1. Problema 1

Típicamente, los lenguajes de programación utilizan uno o más símbolos de agrupación como el paréntesis y las llaves para agrupar el contenido lógico del código fuente. Desgraciadamente este mecanismo de organización del código puede terminar siendo complicado para expresiones realmente complejas que involucran muchos símbolos de agrupación, y los programadores terminamos siempre cometiendo errores de sintaxis al dejar uno o más símbolos de agrupación sin su respectiva pareja de cierre. Un ejemplo es el siguiente condicional escrito en PHP:

```
if (( feof( $archivo) && ($found==false ) ) || ! ( file_exists ( $archivo ) )
```

donde si se observa cuidadosamente, hace falta un paréntesis. Un texto así se conoce como “mal formado” mientras que un texto que no contiene errores de este tipo se conoce como “bien formado”.

La tarea de este problema consiste en leer un texto como parámetro del programa ejecutable y decir si el texto está correctamente formado o no imprimiendo “bien formado” o “mal formado” respectivamente.

Ejemplos:

```
>programa 'for(int i=0; i<(n%5); i++) { printf('\%d',i); }'  
bien formado  
>programa 'if((feof($archivo))||!(file_exists(\$archivo))'  
mal f  
ormado  
>programa 'xcoord = (float)(arrx[i]-(i%2))/(array[i]+1);'  
bien formado  
>programa 'a = b+1;'  
bien formado  
[user@host]# programa 'switch($arr[$i])'  
mal formado
```

Restricciones:

- Los caracteres que se considerarán símbolos de agrupación serán los siguientes:

(){}[]

- La cadena a analizar no tendrá más de 1000 caracteres y puede ser vacía.
- La cadena a analizar no necesariamente contendrá símbolos de agrupación, y en tal caso es bien formada.

2. Problema 2

Una estructura de datos muy utilizada hoy día en las ciencias computacionales es el *trie*, o árbol de prefijo (http://en.wikipedia.org/wiki/Prefix_tree) en donde los recorridos por sus nodos son hechos utilizando símbolos en una cadena los cuales indican el siguiente nodo a visitar. La Figura 1 ilustra un árbol de prefijo que contiene palabras como “in”, “tea” y “to”, el cual contiene 9 nodos. Cabe destacar que en un árbol de este tipo la llave del nodo raíz siempre es vacía.

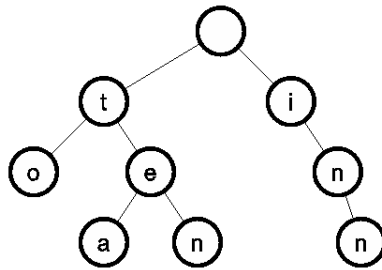


Figura 1: Ejemplo de un Trie

La tarea de este problema es leer de un archivo de texto, dado como argumento del programa, un conjunto de palabras y determinar cual es el numero minimo de nodos que se necesitan para representar este conjunto de palabras en un arbol de prefijo.

Ejemplo:

1. Si el contenido del archivo `dictionary9` es

```
to
tea
ten
in
inn
```

entonces

```
>programa dictionary9
9
```

Este es el árbol de la Figura 1

2. Si el contenido del archivo `dictionary15` es

all
abe
are
alloy
aloe
allot
ant

entonces

```
>programa dictionary15  
15
```

este árbol corresponde a la Figura 2.

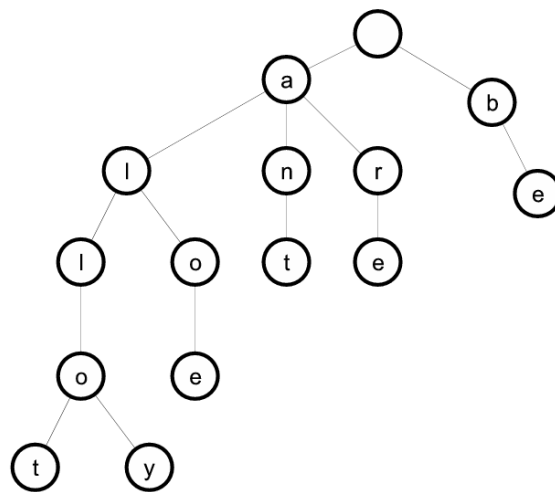


Figura 2: Árbol del ejemplo 2

3. Si el contenido del archivo `dictionary` es

ending
null
character
that
signals
the
end
of
string
is
automatically
appended
characters
from
input
sequence
stores

entonces

```
>programa dictionary
81
```

Restricciones:

- El conjunto de palabras será de máximo 100 palabras.
- El conjunto de letras utilizadas será únicamente letras minúsculas y dígitos.

3. Problema 3

Un sistema de protección de datos utilizado en arreglos de discos de almacenamiento es el sistema RAID en alguno de sus modos. En particular el RAID5 utiliza un sistema de “franjas de datos con revisión de paridad distribuida” en el cual una serie de bits es partida y almacenada a través de todos los discos del arreglo menos uno en el cual se almacena la información de paridad de los bits en el resto de los discos de manera que si alguno de los discos falla es posible leer la información completa de todos modos.

Este problema trata de emular el comportamiento de un controlador de un arreglo de discos que utiliza un sistema de protección de datos parecido al RAID5 y que funciona de la siguiente manera: Asumiendo que el arreglo de discos contienen n discos, en los cuales los datos se escriben en bloques de k bits, una cadena de l bits se escribe en los discos agregando al final de la cadena original suficientes ceros como para que la nueva longitud l' sea divisible por k ; luego los datos se escriben a partir de alguna posición (d, b) donde d es el número del disco y b es el número de la franja, escribiendo los primeros k bits de la cadena en esa posición, luego los siguientes k bits en la posición $(d + 1, b)$, luego otros k bits en la posición $(d + 2, b)$, y sucesivamente hasta llegar al último disco y escribir en la posición (n, b) para luego continuar escribiendo en la posición $(1, b + 1)$, luego la $(2, b + 1)$ y continuando de esta manera hasta que los l'/k bloques hayan sido escritos.

Sin embargo, para conseguir protección de datos, el sistema reserva un bloque en cada franja de bloques (bloques en la misma posición en diferentes discos) para almacenar la información de paridad. La información de paridad se almacena en un bloque de k bits de la siguiente manera: si la cantidad de 1's que hay en la primera posición de todos los bloques de la franja (sin incluir el bloque de paridad) es par, la primera posición del bloque de paridad es 0, y si la cantidad de 1's es impar, la primera posición del bloque de paridad es 1; luego se repite el procedimiento para el resto de las posiciones del bloque de paridad.

Para la primera franja de bloques se elige el bloque del primer disco para almacenar la información de paridad, luego para la segunda franja de bloques se elige el bloque del segundo disco para almacenar la información de paridad, y así sucesivamente se elige el i -ésimo bloque de la i -ésima franja de bloques para almacenar la paridad hasta que se llega a la n -ésima franja, donde se elige el bloque del n -ésimo disco y para la $n + 1$ -ésima franja se elige nuevamente el bloque del primer disco y se empieza de nuevo el proceso.

La Figura 3 muestra un ejemplo de cómo se escribiría la cadena

1011001110011111001110001001110000001001101100011100101111

en un arreglo de 5 discos donde el tamaño de bloque es de 4 bits (obsérvese que a la cadena se le agregan dos 0's para hacer que su longitud sea divisible por 4). La escritura se realiza a partir de la posición $(4, 5)$ (a partir del cuarto disco en la franja 5) y sigue de manera vertical a través de las siguientes franjas (los bloques de color azul). Los bloques de color rojo representan los bloques que son reservados para información de paridad y allí no se escribe el contenido de la cadena, sino que se calcula la información de paridad de la franja. Los bloques grises con información dentro son sólo para ejemplificar el cálculo de paridad de la franja donde se encuentran (no pertenecen a la cadena de bits que estamos escribiendo).

disco 1					1110	0110	1000	1001	1011										
disco 2					0101	0011	1101	1011	1100										
disco 3					0001	1001	1001	1111	1010										
disco 4																			
disco 5					0001	0011	0000	1100	0010										

Figura 3: Ejemplo de escritura de una cadena de bits en un arreglo de 5 discos desde la posición (4, 5)

La tarea de este problema consiste en leer cadenas de bits de un arreglo de discos que utilizan el sistema de protección de datos descrito en los párrafos anteriores en donde se sabe que uno de los discos está inutilizado. El problema leerá la información de un archivo de texto con el siguiente formato:

- (número de discos)
- (tamaño de bloque)
- (posición inicial):(cantidad de bits);(posicion inicial):(cantidad de bits);...;(posicion inicial):(cantidad de bits)
- (bits en los discos)
- (bits en los discos)
- (bits en los discos)

...
y devolverá la concatenación de todas las cadenas de bits que se pidan leer (tercera línea del archivo de entrada). Los bits en los discos serán dados en columnas: la primera columna contiene los bits del primer disco, la segunda columna los del segundo, etc. Los bits dañados (que corresponden a discos dañados) son marcados con 'x'. La cantidad de columnas en la sección de bits en los discos es siempre igual al número de discos dado en la primera línea del archivo.

Ejemplos:

1. Si el contenido del archivo `discos1` es

```
5
4
(2,1):22
1x110
0x111
1x001
1x011
1x000
0x011
0x011
0x001
1x010
0x001
0x010
0x011
1x111
1x000
0x011
0x001
```

entonces

```
>programa discos1
1101110011010111100000
```

2. Si el contenido del archivo `discos1` es

8

3

(4,3):10;(6,1):4;(7,5):25

x0110010

x0001011

x1001111

x1001110

x1111001

x0101100

x1110001

x1100000

x1101110

x1100001

x1010000

x0011010

x0101100

x0111011

x0101111

x1011001

x0110010

x1101011

x0000111

x1000010

x1111000

x0100001

x0000010

x0011011

x1010101

x1011111

x1011011

x0010001

x1110100

x0111111

x1000010

x0111111

x0111101

x0011100

x1110000

x0101000

x0110001

x1000111

x0111111

x1101000

x1001101

x0001101

x0100000

x1110010

x0001010

entonces

>programa discos2

100001001000110110110111010111101010111

Restricciones:

- Habrá un máximo de 100 discos cada uno con un máximo de 1024 bits.
- Siempre habrá exactamente un disco dañado (representado por x) en el archivo.