Quinto Concurso Local de Programación ACM ICPC

Universidad Autónoma Metropolitana Unidad Azcapotzalco

Examen final, 9 de octubre de 2008

Inicio: 16:00 Fin: 19:00

Instrucciones: A continuación encontrarás los enunciados de tres problemas. Cada problema tiene un valor de 100 puntos, los cuales se obtendrán de las salidas que tu programa entregue para cada una de 10 entradas distintas. Todos los programas se evaluarán en Linux. Que tu programa funcione con el ejemplo no quiere decir que funcionará siempre. Ningún programa se ejecutará por más de 5 segundos.

Para cada problema que resuelvas deberás enviar por correo el código fuente de un programa a la dirección `uam08acm@gmail.com` indicando claramente tu nombre completo. De preferencia envía todos tus códigos en un solo correo. El nombre de los archivos que envíes debe ser de la forma `xxx.zzz`, donde `xxx` es el nombre del problema y `zzz` es la extensión, según el lenguaje que hayas usado. Si así lo deseas, puedes resolver cada problema en un lenguaje distinto.

Tu programa deberá leer y escribir exactamente los datos que se indican (ni más ni menos) usando los **archivos** de entrada y salida indicados. En particular, tu programa no deberá borrar la pantalla, escribir ningún tipo de letrero adicional, usar la biblioteca conio, etc.

Quienes programen en C deberán usar el `gcc`, quienes programen en C++ deberán usar el `g++` y quienes programen en Java deberán usar el `javac` de la siguiente manera:

```
 gcc -o nombre_del_ejecutable nombre_del_fuente.c -lm
g++ -o nombre_del_ejecutable nombre_del_fuente.cpp -lm
            javac nombre_del_fuente.java
```

Nota: Todos los puntos calculados que resulten fraccionarios se redondearán siempre hacia abajo.

Problema 1: De tín marín (100 puntos)

Código fuente: `dtm.c`, `dtm.cpp`, `dtm.java`

Archivo de entrada: `dtm.ent` Archivo de salida: `dtm.sal`

Se tiene un grupo de **n** personas numeradas del 1 al **n** y acomodadas en un círculo. La primera persona en el círculo tiene el número $\mathbf{a_1}$, la segunda tiene el número $\mathbf{a_2}$ y así sucesivamente hasta que la última persona tiene el número $\mathbf{a_n}$. No hay dos personas con el mismo número.

Estas personas desean escoger un número **p** y jugar el juego *de tín marín* de la siguiente forma: Se comienza a contar del 0 al **p** a partir de la primera persona y se elimina del juego a la persona que le toque la cuenta **p**. El juego se reinicia a partir de la persona que queda justo después de la recién eliminada y se continúa hasta que todas las personas hayan sido eliminadas del juego.

Con los datos del ejemplo de archivo de entrada y si tomamos **p** = 2 entonces las personas quedarían eliminadas en el orden 5, 1, 3, 4, 2.

Pero estas personas son muy exigentes y no desean escoger cualquier valor de **p**. Los únicos valores de **p** que les interesan son aquellos que eliminan a las personas exactamente en el orden de sus números, es decir, primero se elimina a la persona que tenga el 1, después a la que tenga el 2 y así sucesivamente.

Tu labor es encontrar un valor de **p** tan pequeño como sea posible y que satisfaga estas restricciones.

Entrada: El archivo de texto `dtm.ent` contendrá un entero **n**, seguido de un renglón con **n** enteros distintos $\mathbf{a_1}$, $\mathbf{a_2}$, ..., $\mathbf{a_n}$ separados por espacios. Puedes suponer que $1 \leq \mathbf{n} \leq 22$, que los demás enteros están en el rango 1 a **n** y que existe al menos un valor de p que satisface todas las restricciones.

Salida: El archivo de texto `dtm.sal` deberá contener un entero **p**.

Evaluación: 3 puntos si el valor de **p** satisface la restricción pedida. Además (7*__min__)/**p** puntos donde **min** es el menor valor posible que satisface la restricción pedida. El primer ejemplo mostrado abajo recibiría 3+(7*30)/30 = 10 puntos, el segundo ejemplo recibiría 0 puntos (puesto que la salida no satisface la restricción pedida) y el tercero recibiría 3+(7*30)/90 = 5 puntos.

Ejemplo de archivo de entrada	Ejemplos de archivos de salida		
5 1 4 5 2 3	30	60	90

Problema 2: Voltea monedas consecutivas (100 puntos)

Código fuente: `vmc.c`, `vmc.cpp`, `vmc.java`

Archivo de entrada: `vmc.ent` Archivo de salida: `vmc.sal`

Sean **n** y **m** dos números enteros con $1 \leq \mathbf{m} \leq \mathbf{n}$. Supón que se tienen **n** monedas en una línea. Algunas de ellas tienen el sol hacia arriba y otras tienen el águila hacia arriba. Tu objetivo es lograr que todas las monedas se vean iguales, es decir, que todas tengan el sol hacia arriba o todas el águila hacia arriba.

Para ello dispones de una operación que consiste en escoger **m** monedas consecutivas en la línea y voltearlas de modo que si tenían el sol hacia arriba ahora tienen el águila hacia arriba y viceversa.

Por ejemplo, si **m** = 4 y las monedas están inicialmente en la configuración ①②❸④❺❻⑦ entonces podemos voltear las cuatro monedas a partir de la posición 3 para obtener ①②③❹⑤⑥⑦, luego podemos voltear las cuatro monedas a partir de la posición 1 para obtener ❶❷❸④⑤⑥⑦ y finalmente podemos voltear las cuatro monedas a partir de la posición 4 para obtener ❶❷❸❹❺❻❼.

Tu labor es encontrar un valor de **m** tan grande como sea posible para el que exista alguna forma de llevar la configuración inicial de monedas a alguna de las configuraciones deseadas. Además, para ese valor de **m** deberás encontrar una secuencia de **q** operaciones (tan corta como sea posible) que lleven la configuración inicial a alguna de las configuraciones deseadas.

Entrada: El archivo de texto `vmc.ent` contendrá un entero **n**, seguido de un renglón con **n** enteros $\mathbf{a}_1$, $\mathbf{a}_2$, ..., $\mathbf{a}_n$ con valores 0 o 1, los cuales representan sol o águila, respectivamente. Puedes suponer que $2 \leq \mathbf{n} \leq 1000$. También puedes suponer que al menos habrá un 0 (sol) y un 1 (águila).

Salida: El archivo de texto `vmc.sal` deberá contener dos enteros **m** y **q** separados por un espacio, seguidos un renglón con **q** enteros $\mathbf{p}_1$, $\mathbf{p}_2$, ..., $\mathbf{p}_q$ representando las posiciones de las **q** operaciones.

Evaluación: 3 puntos si la salida representa una solución correcta al problema. Además (4*__m__)/__max__ puntos donde **max** es el mayor valor posible de **m**. Además (3*__min__)/__q__ puntos donde **min** es el menor valor posible de **q** para el valor correspondiente de **m**. El primer ejemplo mostrado abajo recibiría 3+(4*4)/4+(3*3)/3 = 10 puntos y el segundo ejemplo recibiría 3+(4*1)/4+(3*3)/4 = 6 puntos.

Ejemplo de archivo de entrada	Ejemplos de archivos de salida	
7 0 0 1 0 1 1 0	4 3 3 1 4	1 4 2 4 1 7

Problema 3: Sube y baja (100 puntos)

Código fuente: `syb.c`, `syb.cpp`, `syb.java`

Archivo de entrada: `syb.ent` Archivo de salida: `syb.sal`

Sea **n** un entero positivo y suponga que tiene dos vectores **a** y **b** con **n** enteros positivos cada uno. Supón que los elementos del vector **a** se colocan arriba de los elementos del vector **b**.

De cada elemento del vector **a** uno se puede mover a cualquier elemento del vector **b** que sea mayor y que esté a la derecha. Dicho de otra forma, uno se puede mover de $\mathbf{a}_i$ a $\mathbf{b}_j$ si $\mathbf{i} < \mathbf{j}$ y $\mathbf{a}_i < \mathbf{b}_j$. De manera similar, de cada elemento del vector **b** uno se puede mover a cualquier elemento del vector **a** que sea mayor y que esté a la derecha. Dicho de otra forma, uno se puede mover de $\mathbf{b}_i$ a $\mathbf{a}_j$ si $\mathbf{i} < \mathbf{j}$ y $\mathbf{b}_i < \mathbf{a}_j$.

En el ejemplo de entrada se puede ir del 3 de arriba (posición 1) al 7, 8 y 8 de abajo (posiciones 2, 4 y 6), pero del 9 de arriba no se puede ir a ningún lado. Similarmente, del 7 de abajo (posición 2) se puede ir al 9 de arriba (posición 6), pero del segundo 1 de abajo no se puede ir a ningún lado.

Usando estos movimientos uno puede formar caminos que suben y bajan, comenzando en algún elemento de alguno de los vectores y terminando en algún elemento de alguno de los vectores. Los caminos pueden comenzar y terminar en el mismo vector o en vectores diferentes.

Tu labor es encontrar un camino que siga estos movimientos y que pase por **m** elementos con **m** tan grande como sea posible.

Entrada: El archivo de texto `syb.ent` contendrá un entero **n**, seguido de un renglón con **n** enteros positivos $\mathbf{a}_1$, $\mathbf{a}_2$, ..., $\mathbf{a}_n$, seguido de un renglón con otros **n** enteros positivos $\mathbf{b}_1$, $\mathbf{b}_2$, ..., $\mathbf{b}_n$. Puedes suponer que $1 \le \mathbf{n} \le 1000$ y que $1 \le \mathbf{a}_i \le 1{,}000{,}000$ para toda $1 \le \mathbf{i} \le \mathbf{n}$.

Salida: El archivo de texto `syb.sal` deberá contener dos enteros **m** y **r**, seguidos de un renglón con **m** enteros $\mathbf{c}_1$, $\mathbf{c}_2$, ..., $\mathbf{c}_m$. El valor de **r** será 0 si el camino comienza en **a** y será 1 si el camino comienza en **b**. Si **r** = 0 entonces el camino estará formado por $\mathbf{a}_{c1}$, $\mathbf{b}_{c2}$, ... y si **r** = 1 por $\mathbf{b}_{c1}$, $\mathbf{a}_{c2}$, ... Observa que puedes suponer que $1 \le \mathbf{m} \le \mathbf{n}$. También observa que $\mathbf{c}_1 < \mathbf{c}_2 < ... < \mathbf{c}_m$.

Evaluación: 3 puntos si la salida representa una solución correcta al problema. Además (7*__m__)/__max__ puntos donde **max** es el mayor valor posible de **m**. El primer ejemplo mostrado abajo recibiría 3+(7*4)/4 = 10 puntos y el segundo ejemplo recibiría 3+(7*3)/4 = 8 puntos.

Ejemplo de archivo de entrada	Ejemplos de archivos de salida	
7 3 1 4 1 5 9 2 2 7 1 8 2 8 1	4 1 1 3 4 6	3 0 4 5 6