Tercer Concurso Local de Programación ACM ICPC

Universidad Autónoma Metropolitana Unidad Azcapotzalco

Examen final, 20 y 21 de Octubre de 2006

Instrucciones: Abajo encontrarás los enunciados de tres problemas. Cada problema tiene un valor de 100 puntos, los cuales se obtendrán de las salidas que su 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. Para cada problema que resuelvas deberás enviar por correo el código fuente de un programa a la dirección `uam06acm@gmail.com` indicando claramente tu nombre completo. De preferencia envía todos tus códigos en un 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 -ansi -o nombre_del_ejecutable nombre_del_fuente.c -lm
g++ -ansi -o nombre_del_ejecutable nombre_del_fuente.cpp -lm
javac nombre_del_fuente.java
```

Deberás enviar tus códigos fuente a más tardar el 21/10/2006 a las 21:00.

`/* :-) Los organizadores te deseamos éxito en tu examen (-: */`

Problema 1: Tortuga vuelve a casa (100 puntos)

Código fuente: `tvc.c, tvc.cpp, tvc.java`
Archivo de entrada: `tvc.ent` **Archivo de salida:** `tvc.sal`

Logo, la mítica tortuga cibernética que vive en un mundo cuadriculado, ha aprendido un sencillo lenguaje que consta de dos instrucciones: un punto (`.`) la instruye a girar 45 grados contra las manecillas del reloj y un guión (`-`) la instruye a avanzar una unidad. Logo tiene su casa en la coordenada (0,0) de su mundo y cuando sale de paseo siempre lo hace hacia el norte, es decir, como si estuviera viendo hacia la coordenada (0,1). Después de seguir sus instrucciones, Logo llega a la coordenada (**x**,**y**) viendo en alguna dirección. Escribe un programa que encuentre una lista tan corta como sea posible de instrucciones para Logo que la lleven de su posición actual a su casa y viendo hacia el norte. Por ejemplo, si Logo recibe las instrucciones `.--..-` entonces se moverá sucesivamente a las coordenadas (-1,1), (-2,2), (-3,1) y quedará viendo hacia el suroeste, es decir, hacia la coordenada (-4,0). Una posible lista de instrucciones que la lleva a su posición original es `-.....--..` pasando

por las coordenadas (-4,0), (-3,0), (-2,0), (-1,0) y (0,0). La lista más corta de instrucciones que cumple el mismo objetivo es `..-.--..` pasando en orden por las coordenadas (-2,0), (-1,0) y (0,0).

Entrada: El archivo `tvc.ent` contendrá una cadena formada exclusivamente por puntos y guiones. La longitud de esta cadena será arbitraria, pero pueden suponer que todas las coordenadas por las que pase la tortuga tienen un valor que se puede representar en un entero con signo de 32 bits (`int` en `gcc`).
Salida: El archivo `tvc.sal` deberá contener una cadena formada únicamente por puntos y guiones. La longitud de esta cadena podrá ser arbitraria, pero todas las coordenadas por las que pase la tortuga deberán tener un valor que se pueda representar en un entero con signo de 32 bits.
Evaluación: 2 puntos si la tortuga queda en casa, 1 punto si la tortuga queda viendo al norte. En caso de que la tortuga quede en casa y viendo al norte, entonces se otorgarán adicionalmente `(7*min)/sal` puntos, donde `sal` es la longitud de la cadena de salida y `min` es la mínima longitud posible de una cadena de salida. El ejemplo de salida mostrado abajo recibiría `2+1+(7*8)/10 = 2+1+5 = 8` puntos, puesto que `sal = 10` y `min = 8`.

Ejemplo de archivo de entrada	Ejemplo de archivo de salida
`.--..-`	`-.........`

Problema 2: Distribuyendo rebanadas de pastel (100 puntos)
Código fuente: `drp.c, drp.cpp, drp.java`
Archivo de entrada: `drp.ent` **Archivo de salida:** `drp.sal`

La fiesta ha terminado y solamente quedan dos personas y muchas rebanadas de pastel. Como el pastel no se puede desperdiciar, las dos personas han llegado al siguiente acuerdo: cada una se va a llevar a casa el mismo número de rebanadas de pastel. La dificultad radica en que no todas las rebanadas son del mismo tamaño y también desean llevarse aproximadamente la misma cantidad de pastel a casa. Escribe un programa que lea el tamaño de las 2**n** rebanadas de pastel y que diga cuáles de esas rebanadas se debe llevar cada persona de modo que cada una se lleve **n** rebanadas y que la diferencia entre la sumas de los tamaños de las rebanadas que se lleva cada quién sea tan pequeña como sea posible. Por ejemplo, si **n** = 3 y los tamaños de las 2**n** = 6 rebanadas son 3, 1, 4, 1, 5 y 9 entonces lo mejor que pueden hacer es que uno de ellos se lleve las rebanadas de tamaños 3, 4 y 5 y que el otro se lleve las rebanadas de tamaños 1, 1 y 9, pues la diferencia es de (3+4+5)-(1+1+9)=1. Otra posible solución (pero definitivamente no la mejor) es que uno se lleve las rebanadas de tamaños 3, 1 y 5 y que el otro se lleva las rebanadas de tamaños 1, 4 y 9, con diferencia (1+4+9)-(3+1+5)=5.

Entrada: El archivo `drp.ent` contendrá un entero **n**, seguido de un renglón con 2**n** enteros $\mathbf{t}_1$, $\mathbf{t}_2$, ..., $\mathbf{t}_{2n}$ separados por espacios. Puedes suponer que $1 \leq \mathbf{n} \leq 1000$ y también que $1 \leq \mathbf{t}_i \leq 100$ para toda $1 \leq \mathbf{i} \leq 2\mathbf{n}$.
Salida: El archivo `drp.sal` deberá contener 2**n** enteros $\mathbf{a}_1$, ..., $\mathbf{a}_{2n}$ separados por espacios tales que **n** de ellos son iguales a 0 (indicando las **n** rebanadas de pastel que se lleva una persona) y los otros **n** son iguales a 1 (indicando las **n** rebanadas de pastel que se lleva la otra persona).

Evaluación: 1 punto si la salida contiene **n** ceros y **n** unos. En este caso se otorgarán otros 9 puntos si la diferencia `sal` correspondiente a la salida es igual a la menor diferencia posible `min`, o 0 puntos si `sal` es igual a la mayor diferencia posible `max` o `(9*(max-sal))/(max-min)` puntos si `sal` tiene un valor entre `min` y `max`. El ejemplo mostrado abajo tiene `min` = 1, `max` = 13 y `sal` = 5, por lo que recibiría `1+(9*(13-5))/(13-1)` = 7 puntos.

Ejemplo de archivo de entrada	Ejemplo de archivo de salida
3 3 1 4 1 5 9	0 1 1 0 0 1

Problema 3: Recorriendo las calles de un pueblo (100 puntos)

Código fuente: `rcp.c, rcp.cpp, rcp.java`
Archivo de entrada: `rcp.ent` **Archivo de salida:** `rcp.sal`

Un viajero está recorriendo un pueblo en bicicleta. Las calles del pueblo siempre unen dos esquinas distintas y son tan estrechas que sólo puede pasar un vehículo a la vez, por lo que se les ha asignado una dirección que el viajero debe respetar. Él no tiene un mapa pero tiene muy buena memoria, así que puede recordar si ya pasó por alguna calle del pueblo. Escribe un programa que lea la descripción de las calles del pueblo (puedes suponer que siempre habrá al menos una calle) y que encuentre un recorrido tan largo como sea posible de modo que el viajero no pase más de una vez por ninguna calle.

Entrada: El archivo `rcp.ent` contendrá un número entero **n**, con $2 \leq \mathbf{n} \leq 50$, seguido de **n** renglones con **n** números enteros $\mathbf{a}_{i,1}$, $\mathbf{a}_{i,2}$, ..., $\mathbf{a}_{i,n}$ separados por espacios. Para cada $1 \leq \mathbf{i}, \mathbf{j} \leq \mathbf{n}$, el valor de $\mathbf{a}_{i,j}$ es 1 si hay una calle que va de la esquina **i** a la esquina **j** y es 0 en caso contrario.
Salida: El archivo `rcp.sal` deberá contener un número entero **m**, seguido de un renglón con **m**+1 números enteros $\mathbf{b}_0$, $\mathbf{b}_1$, $\mathbf{b}_2$, ..., $\mathbf{b}_m$. El valor de **m** es el número de calles en el recorrido encontrado, mientras que $\mathbf{b}_0$, $\mathbf{b}_1$, $\mathbf{b}_2$, ..., $\mathbf{b}_m$ representa el orden en el que se visitan las esquinas del pueblo, es decir, el recorrido comienza en la esquina $\mathbf{b}_0$, sigue una calle hasta $\mathbf{b}_1$, sigue otra calle hasta $\mathbf{b}_2$, y así hasta que la última calle que sigue va de $\mathbf{b}_{m-1}$ a $\mathbf{b}_m$.
Evaluación: 1 punto si el recorrido es válido. En este caso se otorgarán otros 9 puntos de la siguiente forma. Sea `max` la mayor longitud posible de entre todos los recorridos válidos, entonces obtendrá `(9*m)/max` puntos. El ejemplo mostrado abajo tiene `max` = 8 (por ejemplo, siguiendo el recorrido 1, 2, 3, 4, 5, 2, 4, 1, 5) por lo que recibiría `1+(9*4)/8` = 5 puntos.

Ejemplo de archivo de entrada	Ejemplo de archivo de salida
5 0 1 0 0 1 0 0 1 1 0 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0	4 1 5 2 4 5

Nota: Todas las divisiones en el cálculo de puntos se redondean hacia abajo.