Octavo Concurso Local de Programación ACM ICPC

Universidad Autónoma Metropolitana Unidad Azcapotzalco

Examen final, 1 al 4 de octubre de 2011

Inicio: 01:00 del 1 de octubre. Fin: 04:00 del 4 de octubre.

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. El 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 `uam11acm@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, esperar ninguna tecla, 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 `gcj` de la siguiente manera:

```
gcc xxx.c -lm -o xxx
g++ xxx.cpp -lm -o xxx
gcj xxx.java –main=xxx -Wall -o xxx
```

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

Problema 1: Laberinto salvajemente peligroso (100 puntos)

Código fuente: lsp.c, lsp.cpp, lsp.java

Archivo de entrada: lsp.ent Archivo de salida: lsp.sal

Seguramente ya sabes cómo encontrar el camino más corto para escapar de un laberinto, pero lamentablemente esta vez tu misión es mucho más peligrosa pues ciertos lugares de éste están ocupados por lobos que intentarán comerte. Como estás en forma, crees poder enfrentarte con **L** lobos pero probablemente el lobo **L**+1 ya sea demasiado para ti. Tu labor será encontrar la ruta más corta para escapar del laberinto evitando enfrentarte con más de **L** lobos durante tu recorrido.

Entrada: El archivo de texto `lsp.ent` contendrá el valor del entero **L** seguido de un entero **N** y de un tablero de **N** × **N** caracteres. Una `E` y una `S` representan la entrada y la salida del laberinto respectivamente mientras que un `#` representa un muro, un `*` representa un lobo y un `.` la ausencia de obstáculos. Puedes suponer que **L** ≥ 0, 2 ≤ **N** ≤ 100 y que existirán exactamente una entrada y una salida.

Salida: El archivo de texto `lsp.sal` deberá contener un entero **P** que sea el número de pasos usados para salir del laberinto sin enfrentar más de **L** lobos o -1 si no existe solución. En caso de que sí haya solución, el segundo renglón deberá contener la secuencia de **P** pasos que van de la entrada a la salida como una secuencia de **P** letras `N`, `S`, `E` y `O`.

Evaluación: 3 puntos si el camino indicado cumple todas las condiciones. En ese caso, 7**M**/**P** puntos adicionales, donde **M** es el número mínimo de pasos necesarios. En caso de no haber solución y esto se identifique correctamente se otorgarán los 10 puntos.

Ejemplo de entrada	Ejemplo de salida
1 6 ###### #*S..# #*##.# #.##*# E....# ######	9 EEEENNNOO

Problema 2: País que confunde turistas (100 puntos)

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

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

Escuchaste de un país interesante y quieres visitarlo. Buscando en internet encontraste que el país tiene **N** ciudades y te gustaría visitarlas todas, pero como te aburres fácilmente sólo planeas estar un día en cada ciudad y por eso únicamente solicitaste **N** días de vacaciones. Dentro del país los camiones entre ciudades sólo salen con la puesta del sol y como no regresarás a una ciudad ya visitada sólo necesitas transportarte **N**-1 veces. Curiosamente las ciudades no tienen nombre y están numeradas de 1 a **N**. Sin embargo este país es muy extraño pues aunque siempre se puede ir de una ciudad a otra, las casetas cambian de precio de un día para otro de manera aparentemente arbitraria. Un amigo tuyo que trabaja en la embajada de aquel país te ha filtrado los precios que tendrán las casetas los primeros **N**-1 días de tus vacaciones que son en los que necesitarás transporte, así que te propones visitar el país a un costo mínimo.

Entrada: El archivo de texto `pct.ent` contendrá un entero **N** seguido de **N** × (**N**-1) × (**N**-1) renglones. Cada renglón estará compuesto de cuatro enteros **A, B, D, P,** donde **P** es el precio de la caseta que está entre las ciudades **A** y **B** el día **D** de tus vacaciones (el precio de la caseta puede variar dependiendo del sentido del trayecto). Puedes suponer que $2 \leq \mathbf{N} \leq 18$, $1 \leq \mathbf{A} \neq \mathbf{B} \leq \mathbf{N}$, $1 \leq \mathbf{D} < \mathbf{N}$, $1 \leq \mathbf{P} \leq 100{,}000{,}000$.

Salida: El archivo de texto `pct.sal` deberá contener un entero **C** que sea el costo al cual puedes visitar las N ciudades. Además debes dar el orden de visita de las ciudades.

Evaluación: 1 punto si la ruta dada es válida y tiene el costo C. En ese caso 9**M**/**C** puntos adicionales, donde **M** es el costo mínimo posible de una ruta.

Ejemplo de archivo de entrada	Ejemplo de archivo de salida
3 1 2 1 13 1 3 1 60 2 1 1 15 2 3 1 26 3 1 1 30 3 2 1 90 1 2 2 16 1 3 2 19 2 1 2 16 2 3 2 31 3 1 2 14 3 2 2 76	34 2 1 3

Problema 3: Combinación de una caja fuerte (100 puntos)

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

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

Estás intentando abrir una caja fuerte con un sistema de combinación curioso. La puerta de la caja tiene un disco con **N** dígitos anotados en ella. Para abrir la caja comienzas en el primer número, digamos **A**, y debes moverte **A** posiciones ya sea en el sentido de las manecillas del reloj o al revés. La caja se abrirá una vez que llegues a alguno de los **N** dígitos que valga 0. Tu tarea es la de abrir la caja usando tan pocos movimientos como te sea posible.

Por ejemplo, si la caja tiene anotados los 8 dígitos 32450563 entonces comienzas en el primer dígito (3), después vas 3 posiciones en el sentido de las manecillas del reloj, llegas al cuarto dígito (5), vas 5 posiciones en el sentido contrario de las manecillas del reloj, llegas al penúltimo dígito (6), después vas 6 posiciones en el sentido de las manecillas del reloj y llegas al quinto dígito (0).

Entrada: El archivo de texto `ccf.ent` contendrá una cadena de **N** dígitos. Puedes suponer que 1 <= **N** <= 10 y que habrá al menos un 0 en la cadena.

Salida: El archivo de texto `ccf.sal` deberá contener la cantidad **M** de movimientos seguido de las direcciones de los **M** movimientos: 0 significa en el sentido de las manecillas del reloj y 1 es al revés.

Evaluación: 2 puntos si los **M** movimientos dados abren la caja. En ese caso 8**C**/**M** puntos adicionales, donde **C** es el número mínimo de movimientos necesarios.

Ejemplo de archivo de entrada	Ejemplo de archivo de salida
32450563	3 0 1 0