Segundo Concurso Local de Programación ACM ICPC

Universidad Autónoma Metropolitana Unidad Azcapotzalco

Examen final, 13 de Octubre de 2005

Instrucciones: Abajo encontrarás los enunciados de tres problemas.

Para cada problema que resuelvas deberás dejar en el directorio de trabajo (`acm\` si trabajas en Linux ó `c:\acm` si trabajas en Windows XP) los códigos fuente de un programa. El nombre de estos archivos debe ser de la forma `xxxNN.zzz`, donde `xxx` es el nombre del problema, `NN` es un número de identificación que se te dará al comenzar el examen y `zzz` es la extensión, según el tipo de archivo. Si así lo deseas, puedes resolver cada problema en un lenguaje distinto. Tu programa deberá leer e imprimir exactamente los datos que se indican (ni más ni menos) usando archivos de texto para la entrada y la salida. En particular, tu programa no deberá borrar la pantalla, escribir ningún tipo de letrero adicional, usar la biblioteca `conio`, etc.

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.

Está permitido el uso de libros o notas de curso siempre y cuando no contengan códigos fuente de problemas resueltos.

Cuestiones técnicas para Linux: Se te otorgará una cuenta y su clave en una máquina con Linux, a la que podrás conectarte a través del Secure Shell usando la dirección `148.206.67.155`. En todas las cuentas debe de existir un directorio de trabajo llamado `acm\` (en caso contrario, éste se puede crear con las tres instrucciones (a) `cd ~` (b) `mkdir acm` (c) `cd acm`). 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
```

Cuestiones técnicas para Windows XP: En todas las computadoras debe de existir un directorio de trabajo llamado `c:\acm` (en caso contrario, éste se puede crear desde el símbolo del sistema con las tres instrucciones (a) `cd c:\` (b) `mkdir acm` (c) `cd acm`). Quienes programen en C o C++ podrán usar el **DJGPP**, el cual se puede ejecutar con la instrucción `rhide` desde el directorio de trabajo. Quienes programen en Java podrán usar el **Sun Java SDK**. Puede ser que el día del examen existan otros ambientes de desarrollo disponibles para Windows XP. Si algo no funciona, repórtalo inmediatamente.

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

Problema 1: Números escondidos en números

Código fuente: `nenNN.c, nenNN.cpp, nenNN.java`
Archivos de entrada y de salida: `nen.ent, nen.sal`

Considera la cadena infinita **s** de dígitos que se construye concatenando los enteros positivos consecutivos escritos en decimal y que comienza así:

```
12345678910111213141516171819202122232425262728293031323334353637...
```

En esta cadena cada número decimal aparece una infinidad de veces, por ejemplo el número 3 aparece en las posiciones 3, 17, 37, 50, 52, 54, 56, 57, 58, etc., mientras que el número 31 aparece en las posiciones 17, 52, etc.

Escribe un programa que dado un entero positivo **n** encuentre una posición **p** en la cadena **s** a partir de la cual se puede leer el número **n** en decimal.

Entrada: El archivo `nen.ent` contendrá una línea con el entero **n**. Puedes suponer que $1 \leq \mathbf{n} < 2^{29}$.

Salida: El archivo `nen.sal` deberá contener una línea con el entero **p**. El valor de **p** debe cumplir que $1 \leq \mathbf{p} < 2^{32}$ y puedes suponer que tal valor de **p** existe.

Evaluación: Si **q** es la primera posición donde se puede encontrar al entero **n** y tu programa encontró la posición **p**, entonces se te otorgará la parte entera de $10\mathbf{q}/\mathbf{p}$ puntos (si tu respuesta es incorrecta obtendrás 0 puntos). En el primer ejemplo se te otorgarían 10 puntos y en el segundo sólo 3 puntos:

Ejemplo de archivo de entrada	Ejemplos de archivos de salida
31	17
	52

Problema 2: Empacando una mochila

Código fuente: `eumNN.c, eumNN.cpp, eumNN.java`
Archivos de entrada y de salida: `eum.ent, eum.sal`

Pronto comenzarán las clases y ya has comprado todos tus útiles escolares y una mochila. No sabes cuales útiles serán útiles y cuales serán inútiles, así que a la hora de empacar tu mochila decides llenarla tanto como sea posible. No se permite tomar un útil y romperlo para que una de sus partes quepa (por más inútil que éste útil sea).

Por ejemplo, si tu mochila tiene 100 unidades de volumen y tienes 4 útiles que ocupan 3, 14, 15 y 92 unidades, respectivamente, entonces lo más que puedes empacar son el primero y cuarto útiles, para un total de 95 unidades.

Escribe un programa que dado el volumen **v** de tu mochila, el número **n** de útiles y la lista $\mathbf{u}_1$, $\mathbf{u}_2$, ..., $\mathbf{u}_n$ de espacio que cada uno necesita, decida cuáles útiles hay que empacar para maximizar el volumen ocupado de la mochila (por supuesto, sin sobrepasar su capacidad).

Entrada: El archivo `eum.ent` contendrá dos líneas. En la primera línea estarán los dos enteros **v** y **n** separados por un espacio. En la segunda línea estarán los **n** enteros $\mathbf{u}_1$, $\mathbf{u}_2$, ..., $\mathbf{u}_n$ separados por espacios. Puedes suponer que $1 \leq \mathbf{v} \leq 1000$, que $1 \leq \mathbf{n} \leq 100$ y que $1 \leq \mathbf{u}_1 \leq \mathbf{u}_2 \leq ... \leq \mathbf{u}_n \leq \mathbf{v}$.

Salida: El archivo `eum.sal` deberá contener una línea con **n** enteros $\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_n$ separados por espacios (cuyos valores sólo pueden ser 0 ó 1). El significado de estos enteros es como sigue: Para toda $1 \leq \mathbf{i} \leq \mathbf{n}$, si $\mathbf{x}_i = 1$ entonces el útil **i** está en la mochila, si $\mathbf{x}_i = 0$ entonces el útil **i** no está en la mochila. Además, se debe cumplir que el volumen de los objetos seleccionados no supere al de la mochila.

Evaluación: Si el mayor volumen que se puede empacar en la mochila es **m** y tu programa logró empacar un volumen de **p**, entonces se te otorgará la parte entera de 10**p**/**m** puntos (si tu respuesta es incorrecta obtendrás 0 puntos). En el primer ejemplo se te otorgarían 10 puntos y en el segundo sólo 3 puntos:

Ejemplo de archivo de entrada	Ejemplos de archivos de salida
`100 4`	`1 0 0 1`
`3 14 15 92`	`1 1 1 0`

Problema 3: Bajando la montaña

Código fuente: `blmNN.c, blmNN.cpp, blmNN.java`
Archivos de entrada y de salida: `blm.ent, blm.sal`

Tenemos un mapa de un terreno rectangular que indica la altura a la que está cada punto del mismo. Queremos saber si existe algún punto en ese terreno (al que llamaremos la **cima** de la montaña) desde donde se pueda llegar a todos los demás puntos del terreno caminando horizontalmente o hacia abajo (sólo podemos caminar en las cuatro direcciones del mapa: Norte, Sur, Este y Oeste). Observa que en un terreno pueden haber cero, una o más cimas.

En el ejemplo mostrado abajo, todos los puntos a altura 3 son cimas de la montaña. Pero si cambiamos el 0 por un 3, entonces no habría ninguna cima.

Entrada: El archivo `blm.ent` contendrá dos enteros **m** y **n** en la primera línea y cada una de las siguientes **m** líneas contendrá **n** enteros. El entero $\mathbf{x}_{ij}$ en la posición **j** de la línea **i** es la altura del punto (**j**, **i**) en el mapa. Puedes suponer que $1 \leq \mathbf{m}, \mathbf{n} \leq 100$ y que $-999 \leq \mathbf{x}_{ij} \leq 999$ para toda $1 \leq \mathbf{i} \leq \mathbf{m}$ y toda $1 \leq \mathbf{j} \leq \mathbf{n}$.

Salida: El archivo `blm.sal` deberá contener una línea con un entero **n**, el cual será el número de cimas de la montaña encontradas en el mapa.

Evaluación: Se te otorgarán 10 puntos si encuentras el valor correcto de **n**.

Ejemplo de archivo de entrada	Ejemplo de archivo de salida

`3 4`
`1 0 2 3`
`1 2 3 3`
`1 2 3 1`
	4