Quinto Concurso Local de Programación ACM ICPC

Universidad Autónoma Metropolitana

Examen eliminatorio, 20 a 27 de agosto de 2008

Instrucciones: Abajo encontrarás los enunciados de seis problemas. Cada problema tiene un valor entre 20 y 50 puntos, los cuales se obtendrán de las salidas que entregue tu programa 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 `uam08acm@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 e imprimir exactamente los datos que se indican (ni más ni menos) usando la entrada y la salida estándar. 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 nombre_del_fuente.c -o nombre_del_ejecutable -lm
g++ nombre_del_fuente.cpp -o nombre_del_ejecutable -lm
javac nombre_del_fuente.java
```

Deberás enviar tus códigos fuente a más tardar el 27/08/2008 a las 22:00.

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

Problema 1: Sistema de ecuaciones diofantinas (20 puntos)

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

Escribe un programa que determine la cantidad **n** de soluciones enteras que tiene el sistema de ecuaciones **a**x + **b**y = **c**, **d**x + **e**y = **f**. Observa que es posible que este sistema no tenga ninguna solución entera (es decir **n** = 0) o que tenga un número infinito de soluciones enteras (en cuyo caso debes reportar **n** = -1). Por ejemplo, si **a** = 2, **b** = 1, **c** = 3, **d** = 2, **e** = 3, **f** = 1 entonces **n** = 1 ya que el sistema únicamente tiene la solución x = 2, y = -1.

Entrada: Seis números enteros **a**, **b**, **c**, **d**, **e**, **f** separados por espacios y todos ellos en el intervalo de -1,000,000 a 1,000,000 (incluyéndolos).

Salida: Un número entero **n**.

Evaluación: 2 puntos por el valor correcto de **n**.

Ejemplo de entrada	*Ejemplo de salida*
2 1 3 2 3 1	1

Problema 2: Suma de potencias (20 puntos)

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

Algunos enteros se pueden escribir como suma de dos potencias de enteros donde los exponentes valen al menos dos, por ejemplo $17 = 4^2 + 1^2 = 3^2 + 2^3$. Escribe un programa que determine la cantidad de formas distintas **t** en las que se puede escribir un número entero **n** como suma de dos potencias $\mathbf{a^p} + \mathbf{b^q}$ donde $\mathbf{a} \geq \mathbf{b} \geq 1$, $\mathbf{p} \geq 2$ y $\mathbf{q} \geq 2$. Como todas las potencias de 1 son iguales, solamente deberás considerar como válida la potencia 1^2.

Entrada: Un número entero **n** tal que $1 \leq \mathbf{n} \leq 1{,}000{,}000$.

Salida: Un número entero **t**.

Evaluación: 2 puntos por el valor correcto de **t**.

Ejemplo de entrada	*Ejemplo de salida*
17	2

Problema 3: Escribiendo con un dedo (30 puntos)

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

Muchos de nosotros escribimos en un teclado de computadora utilizando sólo un dedo. Imagina que quieres escribir una palabra **p** en un teclado QWERTY y que te interesa saber la cantidad de columnas que tu dedo se mueve para escribir la palabra (la primera columna está formada por Q, A, Z, etc.). Para ello supondrás que tu dedo comienza en la primera letra de la palabra. Por ejemplo, si la palabra es CONCURSO entonces tu dedo se movió 6 columnas a la derecha, luego 3 a la izquierda, luego otras 3 a la izquierda, luego 4 a la derecha, luego 3 a la izquierda, luego otras 2 a la izquierda y finalmente 7 a la derecha. Escribe un programa que calcule las cantidades **z** y **d** de columnas que se movió tu dedo para la izquierda y para la derecha, respectivamente.

Entrada: Una cadena **p** formada exclusivamente por letras mayúsculas de la A a la Z (sin eñes ni acentos). La cadena **p** contendrá entre 1 y 1000 letras.

Salida: Dos números enteros **z** y **d** separados por un espacio.

Evaluación: 1 punto por el valor correcto de cada uno de **z** y **d**. Además, 1 punto si ambos valores son correctos.

Ejemplo de entrada	*Ejemplo de salida*
CONCURSO	11 17

Problema 4: Operaciones con una pila (40 puntos)
Código fuente: `ocp.c, ocp.cpp, ocp.java`

Considera una pila de enteros que tiene dos operaciones: `mete(a)` introduce el entero `a` en la pila y `saca(a)` elimina los `a` enteros en el tope de la pila (suponiendo que hay suficientes). Se tiene un entero **n** y un vector **a** con n enteros positivos $\mathbf{a_1}$, ..., $\mathbf{a_n}$. Escribe un programa que comience con una pila vacía y que procese uno por uno los elementos de **a** (comenzando con $\mathbf{a_1}$) de la siguiente forma: si al procesar el elemento $\mathbf{a_i}$ la pila tiene menos de $\mathbf{a_i}$ elementos entonces `mete(`a_i`)`, de lo contrario `saca(`a_i`)`. Por ejemplo, si **n** = 7 y **a** = (2, 7, 1, 8, 2, 8, 2) entonces se realizan las operaciones `mete(2)`, `mete(7)`, `saca(1)`, `mete(8)`, `saca(2)`, `mete(8)` y `mete(2)`, quedando 2 y 8 en la pila. Tu programa debe encontrar la cantidad **p** de elementos que quedan en la pila y el orden en el que éstos saldrían si los elimin——áramos uno por uno.

Entrada: Un número entero **n** seguido de un renglón con **n** números enteros $\mathbf{a_1}$, ..., $\mathbf{a_n}$ separados por espacios. Puedes suponer que todos los enteros involucrados valen de 1 a 1000.

Salida: Un número entero **p** seguido de un renglón con **p** números enteros.

Evaluación: 1 punto por el valor correcto de **p** y 3 puntos por el contenido de la pila en el orden correcto.

Ejemplo de entrada	***Ejemplo de salida***
7 2 7 1 8 2 8 2	2 2 8

Problema 5: Puntos dentro de un triángulo (40 puntos)
Código fuente: `pdt.c, pdt.cpp, pdt.java`

Considera un triángulo T dado por sus tres vértices (**a**, **b**), (**c**, **d**) y (**e**, **f**) con coordenadas enteras. Algunos puntos de coordenadas enteras están sobre los lados de T y otros puntos de coordenadas enteras están dentro de T. Escribe un programa que encuentre las candidades **s** y **t** de puntos sobre los lados y dentro de T, respectivamente. Por ejemplo, si los vértices de T tienen las coordenadas (0, 1), (4, 3) y (2, 5) entonces **s** = 6 y **t** = 4 ya que los 6 puntos (0, 1), (2, 2), (4, 3), (3, 4), (2, 5) y (1, 3) están sobre los lados de T mientras que los 4 puntos (1, 2), (2, 3), (2, 4) y (3, 3) están dentro de T.

Entrada: Seis números enteros **a**, **b**, **c**, **d**, **e**, **f** separados por espacios y todos ellos en el intervalo de -1000 a 1000 (incluyéndolos). Puedes suponer que los tres vértices de T no están alineados y por lo tanto forman un triángulo.

Salida: Dos números enteros **s** y **t** separados por un espacio.

Evaluación: 2 puntos por el valor correcto de cada uno de **s** y **t**.

Ejemplo de entrada	*Ejemplo de salida*
`0 1 4 3 2 5`	`6 4`

Problema 6: Acomodando libros en un librero (50 puntos)

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

Se tienen **n** libros numerados del 1 al **n** y acomodados de forma arbitraria en un librero. Se desea reacomodar los **n** libros en el librero de modo que sus números queden en orden ascendente o descendente. La única operación que se permite es la de intercambiar dos libros que son adyacentes. Por ejemplo, si **n** = 5 y los libros estaban en el orden (3, 1, 4, 2, 5) entonces después de un intercambio los libros pudieron haber quedado en cualquiera de los órdenes (1, 3, 4, 2, 5), (3, 4, 1, 2, 5), (3, 1, 2, 4, 5) o (3, 1, 4, 5, 2). Como los libros son muy pesados se desea minimizar la cantidad **c** de intercambios usados para llevarlos a cualquiera de las dos configuraciones deseadas. En nuestro ejemplo se necesitan 3 intercambios para acomodarlos ascendentemente

(3, 1, 4, 2, 5) → (**1, 3**, 4, 2, 5) → (1, 3, **2, 4**, 5) → (1, **2, 3**, 4, 5)

y se necesitan aún más intercambios para acomodarlos descendentemente, por lo que nos conviene la primera opción. Escribe un programa que encuentre un plan de intercambios que lleve la configuración inicial de los libros a cualquiera de las dos configuraciones finales y que minimize la cantidad **c** de intercambios usados.

Entrada: Un número entero **n** con $1 \leq \mathbf{n} \leq 10$, seguido de un renglón con **n** números enteros $\mathbf{a}_1$, $\mathbf{a}_2$, ..., $\mathbf{a}_n$ en el intervalo 1 a **n** y separados por espacios.

Salida: Un número entero $\mathbf{c} \leq \mathbf{n}^2$ seguido de **c** renglones, cada uno de ellos con **n** números enteros $\mathbf{b}_1$, $\mathbf{b}_2$, ..., $\mathbf{b}_n$ en el intervalo 1 a **n** y separados por espacios. El primero de estos renglones representa la configuración de los libros después del primer intercambio, etc.

Evaluación: 3 puntos si la salida constituye un plan correcto. Además 2 puntos si el plan es óptimo. El primer ejemplo de salida obtendría un total de 5 puntos, mientras que el segundo ejemplo sólo obtendría 3 puntos.

Ejemplo de entrada	*Ejemplo de salida 1*	*Ejemplo de salida 2*

`5`
`3 1 4 2 5`
	`3`
	`1 3 4 2 5`
	`1 3 2 4 5`
	`1 2 3 4 5`
		`7`
		`3 4 1 2 5`
		`3 4 1 5 2`
		`3 4 5 1 2`
		`3 4 5 2 1`
		`4 3 5 2 1`
		`4 5 3 2 1`
		`5 4 3 2 1`