XI Concurso de Programación de la UAM
Segunda fase: Examen eliminatorio

División de Ciencias Básicas e Ingeniería
UAM Azcapotzalco

9 a 14 de junio de 2014

¡Bienvenido!

El Concurso de Programación de la UAM tiene el propósito principal de fomentar el aprendizaje de la programación y los algoritmos entre los alumnos de las diversas licenciaturas de la UAM. Además, los alumnos participantes en el XI Concurso de Programación de la UAM tendrán la oportunidad de representar a nuestra institución en la etapa nacional del prestigioso evento ACM International Collegiate Programming Contest, a celebrarse en noviembre de 2014 en Querétaro. Los tres equipos ganadores de la etapa nacional representarán a México en la etapa mundial del mismo que se celebrará en Phuket, Tailandia en 2015.

El XI Concurso de Programación de la UAM se realizará en tres fases:

Primera fase Examen calificatorio (26 a 31 de mayo de 2014).

Segunda fase Examen eliminatorio (9 a 14 de junio de 2014).

Tercera fase Examen final (septiembre de 2014).

Gilgamesh

Todos los problemas del XI Concurso de Programación de la UAM estarán basados libremente en la antigua historia sumeria de Gilgamesh, el rey de Uruk. La primera parte de la historia trata sobre las aventuras de Gilgamesh y su amigo Enkidu, en las que derrotan a Humbaba y al toro celeste, pero que culminan con la muerte de Enkidu. En la segunda parte de la historia se relata el viaje de Gilgamesh en busca de la inmortalidad. Existen varias traducciones de esta historia al español, así que te invitamos a leerla.

Instrucciones

Aquí encontrarás los enunciados de tres problemas. Cada problema tiene un valor de 60 puntos, los cuales se obtendrán de las salidas que entregue tu programa para cada una de 30 entradas distintas. Todos los programas se evaluarán en Linux y se ejecutarán un máximo de 1 segundo. Que tu programa funcione con el ejemplo no quiere decir que funcionará siempre. Deberás enviar por correo electrónico el código fuente de los problemas que resuelvas a la dirección `uam14acm@gmail.com` indicando claramente los siguientes datos:

Datos personales Nombre completo, fecha de nacimiento y teléfono.

Datos académicos Número de matrícula y créditos aprobados.

Otros Licenciatura y Unidad (Azcapotzalco, Cuajimalpa o Iztapalapa).

De preferencia envía todos tus códigos en un solo correo. El nombre de los archivos que envíes debe ser de la forma `prg.zzz`, donde `prg` es el nombre del programa fuente 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`, hacer pausa, etc.

Quienes programen en C deberán usar `gcc`, quienes programen en C++ deberán usar `g++` y quienes programen en Java deberán usar `gcj` como sigue:

C `gcc prg.c -o prg -lm`

C++ `g++ prg.cpp -o prg -lm`

Java `gcj prg.java -o prg`

Deberás enviar tus códigos fuente y los datos solicitados a la dirección de correo `uam14acm@gmail.com` a más tardar el 14 de junio de 2014 a las 22:00.

Entrada, salida y evaluación

La mayoría de los programas requieren leer datos proporcionados por el usuario y emiten una salida con el resultado de dicho programa. Existen muchas maneras en las que un programa puede leer y emitir datos; la manera por defecto es mediante la *entrada* y la *salida estándar*.

La entrada y salida estándar son flujos de datos que están disponibles automáticamente. Dichos flujos están usualmente vinculados a la consola. Los programas en C pueden usar la entrada y la salida estándar con las funciones de biblioteca `scanf` y `printf` respectivamente (además de algunas adicionales, como `getchar` y `putchar`). Los programas en C++ pueden usar las mismas utilidades que C y algunas adicionales: el objeto `cin` está vinculado por defecto a la entrada estándar y el objeto `cout` a la salida estándar. De manera similar, los programas en Java pueden usar los objetos `System.in` y `System.out` para la entrada y salida de datos.

Para calificar los programas que envíes al concurso, se comparará lo que tu programa haya emitido en la salida estándar con lo que emita un programa correcto usando los mismos datos de entrada. La comparación será literal, es decir, la salida de tu programa y la salida del programa correcto deben ser *idénticas*. Es por esto que debes evitar imprimir cosas adicionales a lo especificado en cada problema: ya que nuestro programa no imprimirá cosas adicionales, tu salida será diferente si tu programa sí lo hace.

La mayoría de los sistemas operativos permiten vincular la entrada y salida estándar de un programa a algo que no sea la consola de comandos. Es posible, por ejemplo, vincular dichos flujos de datos a archivos. Esto es muy útil cuando tu programa usa `scanf` y `printf` pero quieres guardar la salida de tu programa en un archivo o cuando quieres evitar tener que escribir la entrada manualmente cada vez que ejecutes tu programa.

La forma de lograr esto es la misma tanto en Linux como en Windows o Mac OS si ejecutas tu programa desde la consola de comandos:

Comando	Descripción
`./prg < ent.txt`	La entrada estándar de `prg` viene de `ent.txt`
`./prg > sal.txt`	La salida estándar de `prg` va a `sal.txt`
`./prg < ent.txt > sal.txt`	La entrada estándar de `prg` viene de `ent.txt` y la salida estándar va a `sal.txt`

Debes recordar que al vincular la entrada estándar a un archivo, éste debe existir y debe contener los datos de la entrada, pues de lo contrario ocurrirán errores durante la lectura de datos. Cuando la salida estándar se vincula a un archivo, éste se creará o sobreescribirá cada vez que ejecutes tu programa.

1. La muralla interminable de Humbaba

`muralla.c muralla.cpp muralla.java`

Gilgamesh y Enkidu han decidido ir en busca de una nueva aventura por lo que se proponen enfrentar a Humbaba, cuidador del *bosque de los cedros*. Sin embargo, para poder llegar a donde se esconde Humbaba deben encontrar la entrada a la muralla que rodea dicho bosque. Antes de encontrar la entrada a la muralla, Gilgamesh y Enkidu deben determinar la cantidad de muros que la forman. Para ello, se han dado cuenta de que cada muro de la muralla se encuentra pintado de color blanco o negro. Con esta información, nuestros héroes rodearon la muralla. Después de un rato, saben que han dado al menos una vuelta completa y que tienen el registro de los colores de cada muro que se vieron en su recorrido, pero no saben si terminaron en el mismo lugar donde iniciaron. Escribe un programa que determine la cantidad mínima N de muros que puede tener la muralla.

1.1. Entrada

Una cadena M. El caracter $M[i]$ es `N` si el color del muro visto es negro o `B` si es blanco. Puedes suponer que M tiene entre 1 y 100 000 caracteres.

1.2. Salida

Un entero N.

1.3. Ejemplo

Entrada	Salida
BNNBBNBNNB	6

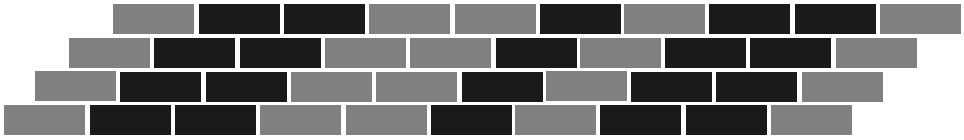

Figura 1: Esta muralla tiene al menos 6 muros.

2. La fila de guerreros de Uruk

`fila.c fila.cpp fila.java`

El toro celeste está por derrotar al ejército de Uruk, por lo que la última esperanza es que el resto de los hombres también pelee. Por lo confuso de la situación, los R hombres restantes se fueron metiendo en la fila de G guerreros en desorden, en lugar de formarse al final. Los escribanos, que tenían prohibido pelear, quedaron encargados de llevar el registro de cómo se iba formando la fila. Afortunadamente para los escribanos, cada hombre de Uruk tenía un identificador distinto, por lo que el registro era sencillo de llevar. Escribe un programa que, dada la fila inicial y la información de cómo se fueron metiendo los hombres a ésta, calcule el estado final de la fila.

2.1. Entrada

Un entero G seguido de G enteros que son los identificadores de los hombres según el orden de la fila original. Luego un entero R seguido de R parejas de enteros A_i, B_i que significan que el hombre con identificador A_i se metió en la fila justo atrás del hombre con identificador B_i. Puedes suponer que G, R y todos los identificadores están en el rango de 1 a 100 000.

2.2. Salida

Los $G+R$ identificadores de los hombres en el orden final de la fila.

2.3. Ejemplo

Entrada	Salida
3 2 7 5 4 1 2 9 2 3 5 4 1	2 9 1 4 7 5 3

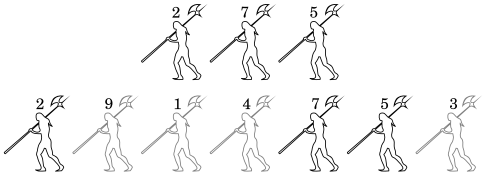

Figura 2: La fila antes y después de las modificaciones.

3. Los remos faltantes de Urshanabí

`remos.c remos.cpp remos.java`

El último obstáculo de Gilgamesh para visitar a Utanapíshtim era cruzar las *aguas de la muerte* y el único que lo podía ayudar era el barquero Urshanabí. Éste le pidió a Gilgamesh que fuera al bosque a cortar R árboles para hacer los R remos que le faltaban. Como Gilgamesh estaba cansado, quería caminar lo menos posible para traer uno por uno los árboles necesarios. Escribe un programa que, dadas las coordenadas de N árboles, calcule la mínima distancia D que debe caminar Gilgamesh desde el origen.

3.1. Entrada

Dos enteros N y R, seguidos de N parejas de enteros X_i, Y_i que son las coordenadas (X_i, Y_i) del árbol i. Puedes suponer que $1 \leq R \leq N \leq 100\,000$ y que $0 \leq X_i, Y_i \leq 1000$.

3.2. Salida

El valor de D (redondeado al entero más cercano).

3.3. Ejemplo

Entrada	Salida
4 3 2 2 3 2 2 1 1 3	16

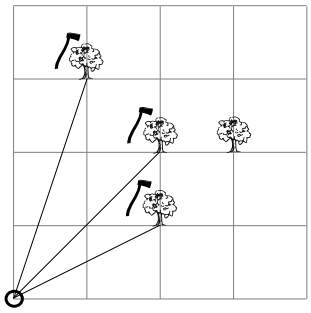

Figura 3: La distancia mínima que debe caminar Gilgamesh es ≈ 16.45.