COMPENDIO DE PROBLEMAS

OLIMPIADAS MEXICANAS DE INFORMÁTICA

(1995-2002)

ING. MARGARITA GARCÍA BURCIAGA
SECRETARIA GENERAL DEL COMITÉ OLÍMPICO MEXICANO DE INFORMÁTICA (COMI)
PROFESORA INVESTIGADORA DE LA ESIME-ZACATENCO - IPN
2003

COMPENDIO DE EXÁMENES NACIONALES DE **LA**
OLIMPIADA MEXICANA DE INFORMÁTICA

índice

	Pag.
Prólogo	i
índice	¡i
Introducción	ii
Historia de la IOI y de la OMI	¡v
Ejemplos de exámenes regionales	1
Teórico - Desarrollo de algoritmos 1999	2
Cajas 1999	6
División 1999	7
Razonamiento Lógico 2002	8
Primer concurso de informática 1995	13
Codiciosos dadores de regalos	13
El gato en el sombrero	14
1ª. Olimpiada Mexicana de Informática	15
Pirámide	16
Taxis	17
Mudanza	18
Mina	19
Circuito	20
Pento	21
2ª. Olimpiada Mexicana de Informática 1997	23
Submarino	24
Átomos	25
Cintas	26
Superconejo	27
3ª. Olimpiada Mexicana de Informática 1998	29
Fichas	30
Cálculos	31
Sumas	32
Explorador	33
4ª. Olimpiada Mexicana de Informática 1999	35
Omicel	36
Práctica	37
Atinale	38
Torneo	40
5ª. Olimpiada Mexicana de Informática 2000	41
Moscas	42
Maratón de Cine	43
Sumas 2000	44
Vendedor de alfombras	45
Cajas 2000	46
Cajas y Más Cajas	47
Misión Imposible	48
Escalera	49
6ª. Olimpiada Mexicana de Informática 2001	51
Tareas	52
Comisólo	53
Defender	54
Elevador	55
Tapetes	56
7ª. Olimpiada Mexicana de Informática	57
Descifrando Jeroglíficos	58
En camino hacia el tesoro	60
Abriendo la puerta	62
El tesoro	63
Tips de solución para algunos problemas	65
Biblografía	66

INTRODUCCIÓN

En este trabajo se ha hecho una recopilación de los problemas usados en las evaluaciones de las siete Olimpiadas Mexicanas de Informática (OMI), con objeto de presentar a los interesados en esta disciplina el tipo de problemas, la complejidad, la dificultad y la implementación de los mismos, de manera que tanto los profesores, como los posibles participantes, puedan autoevaluar su potencial para este tipo de competencias.

La clasificación de estos exámenes nacionales se realiza en orden cronológico desde la primera OMI en 1996, hasta la séptima OMI del 2002, y se establecen los grados de dificultad del algoritmo que se usa para la solución, desde el más fácil con cero hasta el más difícil con cinco, así mismo, se establece una clasificación sobre la implementación del código respectivo que va desde S (Sencilla), SL (Sencilla pero Larga), SE (Sencilla y propensa a Errores), C (Compleja), CL (Compleja y Larga), CE (Compleja y propensa a Errores) y N/A (No Aplica) .

Es importante hacer notar que cada uno de estos problemas, en el día de la competencia debe ser resuelto en un promedio de 90 minutos, usando la computadora, y que los programas evaluadores están diseñados para asignar casos aleatorios y medir el tiempo en que cada uno de los problemas debe correr, y de esta forma asignar la calificación correspondiente.

Mediante estas clasificaciones, el tiempo que se destina a la solución de cada problema, así como, los métodos de evaluación usados en cada caso, el lector se podrá dar idea de sus posibilidades al participar en una competencia de este tipo, así como, que áreas de conocimiento y habilidades debe desarrollar para aumentar estas posibilidades.

Esperamos que este compendio de exámenes de la OMI, brinden una panorámica ciara sobre el tipo de competencia de que se trata. Cabe mencionar que el trabajo frente a la PC de cada uno de los contendientes se debe *realizar* en Turbo Pascal o C++, lenguajes que son los oficiales tanto en la OMI como en la IOI (International Olympiad in Informatics).

Agradezco a todas las personas que han hecho posible la realización de esta publicación, muy especialmente a los ingenieros María del Mar Cepeda García y César Arturo Cepeda García, quienes son los generadores incansables de la mayoría de los problemas y los tips de solución aquí planteados.

Atentamente

Ing. Margarita García Burciaga Secretaria
General del Comité Olímpico Mexicano de
Informática (COMI) México D.F. 2003

Historia de la IOI (Olimpiada Internacional de Informática)

La ¡dea de iniciar una Olimpiada Internacional de Informática para estudiantes de preparatoria, fue propuesta en la 24ª Conferencia General de la Organización Educacional, Científica y Cultural de las Naciones Unidas (UNESCO) en París, por el Delegado de Bulgaria Profesor Sendov, en octubre de 1987.

La IOI es una de las seis Olimpiadas de la Ciencia que existen a nivel Internacional, éstas son:

1. Olimpiada de Matemáticas
2. Olimpiada de Física
3. Olimpiada de Química
4. Olimpiada de Biología
5. Olimpiada de Informática
6. Olimpiada de Astronomía

El objetivo principal de la IOI es, el de estimular el interés en la informática (esto es, las ciencias de la computación y las tecnologías de la información). Otro objetivo importante es el de reunir a estudiantes excepcionalmente talentosos de distintos países, para que puedan compartir experiencias científicas y culturales.

La IOI es organizada anualmente por alguno de los países participantes, cada país manda una delegación integrada por cuatro estudiantes y dos adultos que fungirán como líder y sublíder de la misma. Los estudiantes compiten individualmente y tratan de maximizar su puntaje, resolviendo una serie de problemas de informática durante los dos días de la competencia. Los problemas se resuelven utilizando las computadoras personales.

Además de la competencia, el país sede organiza eventos culturales y recreativos en el resto de los días que dure el evento.

Durante la IOI del 2002, México compitió contra Inglaterra y Egipto y gano la sede de la IOI para el 2006, esta se llevará a cabo en Mérida, Yucatán.

HISTORIA DE LA OMI (Olimpiada Mexicana de Informática)

La Olimpiada Mexicana de Informática, es un concurso nacional de carácter académico, dirigido a jóvenes mexicanos estudiantes del sistema de enseñanza media escolarizada o abierta, interesados en las áreas de ciencias de la computación y programación de computadoras.

En 1992, durante la 4ª Olimpiada Internacional de Informática en Bonn, Alemania, México asistió por primera vez a éste evento en carácter de observador.

Durante la IOI de 1993, que se celebró en Mendoza Argentina, México envió dos alumnos participantes y obtuvo una medalla de bronce.

Durante 1996, se realizo formalmente la 1ª Olimpiada Mexicana de Informática, la cual fue convocada por: ESFM-IPN, ITESM-CEGS, FC-UNAM, FCFM - BUAP.

Actualmente México ha logrado obtener tres medallas de bronce: En Mendoza, Argentina, en Antalya, Turquía y en Yong In, Corea del Sur.

México ha adquirido una gran responsabilidad al ganar la sede de la IOI para el 2006, y está obligado a realizar un digno papel como anfitrión y como participante con la delegación correspondiente de ese año. Por tal motivo solicitamos a todos aquellos profesores y estudiantes que tengan inquietud en ser parte del movimiento olímpico mexicano de informática, que consulten la página www.olimpiadadeinformatica.org.mx

EJEMPLOS DE

EXÁMENES REGIONALES

4ta OLIMPIADA DE INFORMÁTICA DEL DF Y ESTADO DE MÉXICO

EXAMEN TEÓRICO

Durante los últimos meses el departamento de Inteligencia ha estado desarrollando una nueva e innovadora herramienta que nos asegurará la superioridad sobre nuestros enemigos. Esta herramienta con el nombre clave "PC" se dice que será capaz de realizar cálculos, operaciones y manejo de datos a velocidades infinitamente superiores que cualquier otra tecnología actual.

Hasta el momento sólo contamos con prototipos experimentales, cuyo desempeño pronostica un muy buen resultado.

Tu misión, si decides aceptarla; es comprobar si esta herramienta será en verdad útil para los fines que interesan a nuestro gobierno.

La "PC" esta integrada por 3 elementos principales:

- **PROCESADOR.-** Es el cerebro de la máquina y se encarga de llevar a cabo todos los procesos.
- **ÁREA DE CÓDIGO.-** Esta es un área de memoria en la cual el usuario o programador puede por medio de comandos indicarle a la PC los procesos a realizar.
- **ÁREA DE DATOS.-** Esta es un área de memoria en la cual se pueden almacenar hasta 100 datos (uno por cada localidad) enteros con valores de O a 255.

Las personas en el departamento de Inteligencia diseñaron un lenguaje por medio del cual se le indica a la PC que es lo que debe de hacer, este lenguaje permite la utilización de comandos predefinidos y de variables definidas por el programador, el número de variables que se pueden definir no tiene límite y el nombre de cada una de ellas no es importante para el desempeño de los procesos.

COMANDOS:

- ***leedato (localidad de memoria, variable).-*** Utilizando este comando se puede conocer el contenido de una localidad de memoria específica dentro del área de datos y copiar su valor a una variable.

- ***escribedato (localidad de memoria, variable o dato).-*** Utilizando este comando se puede copiar el valor de una variable, o de un dato constante, a una localidad de memoria específica dentro del área de datos **si *(comparación) entonces...fin.-*** En éste comando se especifica una comparación entre dos variables, utilizando los signos de <, > e =. Si esta comparación es verdadera, entonces se realizarán todos los comandos subsecuentes hasta encontrar un ***fin,*** si la comparación es falsa, los comandos antes del ***fin*** serán ignorados y la ejecución del programa continuará en el comando siguiente del ***fin.***

" salta a.- Con este comando se puede alterar el orden del proceso indicándole a la computadora que debe continuar la ejecución no en el comando siguiente sino en el lugar indicado por el comando ***salta a.***

- ***termina.-*** Le indica a la computadora que se acabo el programa.

Además de éstos comandos, puedes realizar cualquier operación aritmética con las variables. *(+,-,*,!)*

EJEMPLO DE UTILIZACIÓN DE LOS COMANDOS: Valores iniciales.

localidad	valor	variable	valor
1	5	x	0
2	10	y	0
3	0	z	0

INSTRUCCIONES	PRIMERA ITERACION	SEGUNDA ITERACION
* leedato(1,x)	valor de x=5	Valor de x=15
Leedato(2,y)	valor de y= 10	Valor de y=10
z=x+y	valor de z= 15	Valor de z= 25
escribedato(1,z)	valor de localidad 1=15	Valor de localidad 1= 25
si (z>20) entonces	no se cumple	se cumple entonces:
escribedato(3,z)		Valor de localidad 3=25
salta a #		Salta a la última línea
Fin		
salta a *	regresa a la primera instrucción.	
# termina		Termina

GRADO *DE DIFICULTAD:*
Implementación:N/A Algoritmo:1

PROBLEMA 1

Para demostrar el poder de esta nueva herramienta las personas en el departamento de inteligencia han hecho un programa de demostración.

Tu trabajo consiste en probar este programa para lo cual debes estudiarlo y escribir en papel la configuración final del área de memoria (de las localidades 1 a 20) para compararla con el área de memoria resultante al ejecutar el programa.

El programa de demostración es el siguiente:

*El **valor inicial de todas las localidades de memoria es 0.***

```
     J=1
     x=1
ET1  Si (x<4) entonces
          Leedato <96+x,b)
          Si (b=0) entonces
             Escribedato(96+x,1)
             y=1
ET2          Si (y<4) entonces
                 Leedato (96+y,b)
                 Si (b=0) entonces
                  Escribedato(96+y,1)
                  z=l
ET3               Si (z<4) entonces
                    Leedato (96+z,b)
                    Si (b=0) entonces
                    Escribedato (j,x)
                    j=j+l
                    Escribedato (j,y)
                    j=j+l
                    Escribedato (j, z)
                    j=j+l
                   fin
                   z=z+l
                   Salta a ET3
                  fin
                  Escribedato (96+y, 0)
                  fin y=y+l
                 Salta a ET2
            fin
            Escribedato (96+x, 0)
       fin x=x+l
       Salta a ET1
  fin
 termina
```

GRADO DE DIFICULTAD:
Implementación :N/A Algoritmo:1

PROBLEMA 2 (BARAJEO PERFECTO)

Cuando se barajea un mazo de cartas, se toma el mazo completo y se divide en dos, posteriormente se juntan los dos montones en un nuevo mazo poniendo una carta del primer montón, y una carta del segundo, y así posteriormente hasta que no quede ninguna carta en ninguno de los montones. Cuando se "barajean" datos se hace de la misma forma, se toma la lista de datos y se divide en dos, posteriormente se juntan de nuevo pero poniendo el primer dato de la primera mitad y luego el primer dato de la segunda, y así hasta terminar los datos en las dos mitades.

Escribe un programa que "barajee" la lista de datos que se encuentra en las localidades de memoria de la 1 a la 50 y escriba la lista "barajéada" de la localidad 51 a la 100. Tu programa debe barajar los datos en la forma antes descrita.

EJEMPLO:

LOCALIDAD	VALOR INICIAL	VALOR FINAL
1	7	7
2	8	10
3	9	8
4	10	12
5	12	9
6	13	13

PROBLEMA 3 (BOMBAS Y DETONADORES)

Cierto tipo de bomba, requiere de un detonador para activarse. Estas bombas están hechas de tal forma, que sólo su detonador puede activarlas. Esto se realiza haciendo que tengan el detonador y el enchufe de éste diferente para cada bomba.

Tienes en tu poder 20 bombas con sus 20 respectivos detonadores. Sin embargo, no sabes cual detonador, corresponde a qué bomba y necesitas saberlo. No puedes comparar ningún detonador con ninguna bomba, ya que para esto requerirías enchufarlos y activarías la bomba. Tan sólo puedes comparar los detonadores entre sí, y las bombas entre sí.

Tu deber es proponer una idea que resuelva éste problema y diseñar el código del programa que lo realice.

Toma en cuenta que en las localidades de la 1 a la 20 están los tamaños de los enchufes de las bombas y de la 21 a la 40, los tamaños de los detonadores.

Como resultado, tu programa debe escribir en las localidades de la 41 a la 60, el número de detonador que le corresponde a cada bomba (enchufe).

EJEMPLO:

BOMBAS (ENCHUFES)	DETONADORES	VALORES FINALES
9	3	4
3	11	1
16	5	5
11	9	2
5	16	3

Así, el resultado indica, que la bomba 1, va con el detonador 4. La bomba 2, con el detonador 1, etc.

NOTA: Al decir que no puedes comparar ninguna bomba con ningún detonador, queremos Indicar que tu programa no puede comparar el valor de la localidad de memoria en donde se encuentra una bomba con el valor de la localidad de memoria en donde se encuentra un detonador.

GRADO DE DIFICULTAD:
Implementación: S/E
Algoritmo:2

PROBLEMA 4 (FRACTALES)

Debido al gran éxito que ha tenido la "PC" como herramienta en los diferentes departamentos del estado, se le ha solicitado al departamento de inteligencia que desarrolle una interfaz gráfica para ampliar su utilidad.

Por el momento la interfaz con la que se cuenta es bastante limitada, solo puede dibujar líneas rectas y sólo en dirección vertical u horizontal. Esta interfaz gráfica se programa por medio de tres instrucciones y una variable general:

Adelante (largo).- Dibuja una línea de longitud largo en la pantalla de la "PC" hacia donde indique la variable dirección

GiraDerecha.- Cambia la variable *dirección* girándola 90° hacia la derecha. A continuación se presenta una tabla de cómo cambia la variable dirección al aplicar está instrucción:

VALOR INICIAL DE DIRECCION	VALOR DESPUES DE GIRADERECHA
Arriba	Derecha
Derecha	Abajo
Abajo	Izquierda
Izquierda	Arriba

GiraIzquierda- Cambia la variable *dirección* girándola 90° hacia la derecha. A continuación se presenta una tabla de cómo cambia la variable dirección al aplicar está instrucción:

VALOR INICIAL DE DIRECCION	VALOR DESPUES DE GIRAIZQUIERDA
Arriba	Izquierda
Izquierda	Abajo
Abajo	Derecha
Derecha	Arriba

Dirección.- La variable *Dirección* indica en que dirección se va a pintar cuando se utilice la instrucción Adelante. Es decir, si la variable *Dirección* tiene el valor *Arriba* y se utiliza la instrucción *Adelante* (10), en la pantalla se dibujará una línea de longitud 10 y con dirección hacia arriba.

Para demostrar el poder de esta nueva interfaz gráfica las personas en el departamento de inteligencia han hecho un programa de demostración.

Tu trabajo consta en probar este programa para lo cual debes estudiarlo y dibujar en pape! la "supuesta" imagen resultante para compararla con la imagen de la pantalla al ejecutar el programa.

El programa de demostración es el siguiente:

```
Programa Principal
        Dirección = Derecha
        Dibuja (90,2)
        Dibuja (90,2)
        Dibuja (90,2)
        Dibuja (90,2)
Termina
Subrutina Dibuja (Iargo, nivel)
Si (nivel>O) entonces
   Adelante (largo / 3)
   GiraIzquierda. Adelante
   (largo / 3)
   Giralzquierda Adelante
   (largo / 3);
   GiraDerecha

   Dibuja(largo / 3, nivel - 1)

   Adelante (largo / 3)
   Giralzquierda Adelante
   (largo / 3)
   GiraDerecha

   Dibuja (largo / 3, nivel - 1)

   Adelante (largo / 3)
   Giralzquierda Adelante
   (largo / 3)
   GiraDerecha

   Dibuja (largo / 3, nivel - 1)

   Adelante (largo / 3)
   Giralzquierda
   Adelante (largo / 3)
   Giralzquierda
   Adelante (largo / 3)
   GiraDerecha Fin Si
(nivel = 0) entonces
   Adelante (1)
   GiraDerecha Fin
Termina Subrutina
```

GRADO DE DIFICULTAD:
Implementación: N/A Algoritmo:2

4ª Olimpiada de Informática del Distrito Federal y Estado de México
Examen Práctico
(1999)

PROBLEMA 1.- CAJAS 1999

Descripción

Trabajas en la bodega de una tienda de autoservicio y tu trabajo consta en acomodar las cajas con mercancía nueva que llega a la bodega. La entrada de mercancía a la bodega se hace por la parte trasera de ésta, y se saca por la parte delantera para ser acomodada en los estantes de la tienda. Para cada mercancía diferente se tiene una hilera de cajas. Cuando llega nueva mercancía tu trabajo es empujar la hilera de cajas de esa mercancía hacia delante hasta que llegue al extremo de la bodega y acomodar las nuevas cajas inmediatamente después.

Para realizar tu trabajo cuentas con un carro que carga las cajas y las empuja por ti, desgraciadamente el carro tiene un desperfecto y éste es el de no poder caminar en reversa solo hacia delante y hacia los lados, por lo que cada vez que quieres hacerlo retroceder tienes que bajarte y empujarlo, esto por supuesto es un trabajo bastante cansado y prefieres no realizarlo en la medida que sea posible.

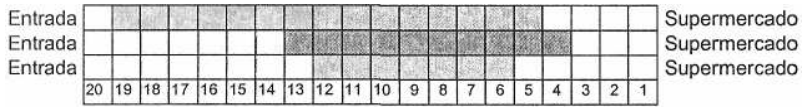
Realiza un programa que conociendo cuantas cajas hay en existencia de cada mercancía y cuanto hay que empujarlas hacia delante, elabore una secuencia en la que el número de veces que tienes que hacer retroceder el carro es la mínima. La bodega tiene una longitud de 20 metros, y todas las cajas de cualquier mercancía tienen una longitud de 1 metro.

Entrada

Tu programa deberá leer del archivo CAJAS.ENT que se encuentra en el subdirectorio de trabajo el número de mercancías M con que cuenta la bodega 0<M<1000 y a continuación M renglones con dos números cada uno en los que se indica el número de cajas existentes de esa mercancía y el número de metros que se debe empujar hacia delante su hilera.

Ver ejemplo.

```
CAJAS . ENT
3
15  4
10  3
7   5
```



Ejemplo

El ejemplo anterior indica que se tiene una configuración inicial de la bodega con tres mercancías diferentes, de la primera mercancía se cuenta con 15 cajas y se debe empujar hacia adelante 4 lugares, por lo que la última caja de esta mercancía se encuentra a 19 metros del inicio de la bodega, es decir, los 15 metros de cada una de las cajas y 4 metros que están vacíos al principio de la hilera, la hilera de la segunda mercancía empieza en la posición 13 y la tercera en la posición 12.

Salida

Tu programa deberá escribir en el archivo CAJAS.SAL un único número que indique el número mínimo de veces que el carro tuvo que retroceder para lograr empujar todas las hileras hacia adelante. Para el ejemplo el archivo de salida

```
CAJAS . SAL
1
```

GRADO DE *DIFICULTAD:*
Implementación:* S *Algoritmo: 1

4ª Olimpiada de Informática del Distrito Federal y Estado de México
Examen Práctico
(1999)

PROBLEMA 2.- DIVISIÓN

Descripción

La compañía en la que trabajas como contador ha tenido un verdadero boom en los últimos meses, lo cual aunque para el resto de la compañía ha sido una bendición para ti ha sido lo peor que te pudo haber pasado, ya que las cantidades que debes manejar son extremadamente grandes y las herramientas de cálculo que utilizas no son capaces de manejar tal cantidad de cifras.

De tu trabajo depende la precisión de los cálculos, por lo que será necesario que desarrolles una herramienta de cálculo que te permita manejar cantidades de tal magnitud.

Tu primera tarea será desarrollar un programa que dados dos número enteros N y M obtenga el residuo de dividir N entre M. N es un número entero que se encuentra en el rango de $[1,1x10^{250}]$ y M es un número entero que se encuentra en el rango de [1,10,000],

Entrada

Tu programa deberá leer del archivo DIVISIÓN.ENT los números N y M los cuales se encuentran en dos renglones diferentes, el número N en el primer renglón y el número M en el segundo. (Ver ejemplo)

```
DIVISIÓN.ENT
12345578901234567890
7
```

Salida

Y deberá escribir en el archivo DIVISIÓN.SAL un único número entero indicando el residuo de la división. Para el ejemplo anterior el archivo de salida sería.

```
DIVISIÓN. SAL
1
```

GRADO DE DIFICULTAD:
Implementación: SE Algoritmo: O

Razonamiento Lógico

Este es un examen de razonamiento lógico y matemático, las siguientes 25 preguntas están agrupadas en cuatro categorías dependiendo de su clasificación. Todas las preguntas tienen el mismo valor y no importa si contestas una pregunta de manera incorrecta. Trata de responder el máximo número de preguntas correctas posible y no permanezcas demasiado tiempo en una misma pregunta. ¡Mucha suerte!

1.- Se tienen tres cajas con canicas de diferentes colores, cada caja tenía un letrero de su contenido. Una tapa dice "verde y rojo", otra tapa dice "azul" y la tercera tapa dice "rojo". Sin embargo, las tapas de las cajas se revolvieron y ahora ninguna de ella esta en donde debería. Para determinar que caja tiene que canicas, puedes abrir la tapa de solo una de las cajas, y sin ver el interior, sacar una canica. Cual caja es la que NO debes abrir? Y por que?

a) La caja con la tapa que dice "verde y rojo"
b) La caja con la tapa que dice "azul"
c) La caja con la tapa que dice "rojo"
d) La caja con la tapa que dice "rojo" y la caja con la tapa que dice "azul"
e) Ninguna de ellas, cualquiera que abriéramos nos da la respuesta correcta

Por que? _____

2.- Vero es más rápida que Liz, y Ruth es mas lenta que Vero. Cual de los siguientes enunciados es correcto:

a) Ruth es más rápida que Liz.
b) Ruth es más lenta que Liz.
c) Ruth es tan rápida como Liz.
d) Es imposible saber quien es más rápida de Ruth o de Liz.

3.- Supongamos que los siguientes enunciados son verdaderos:

a) Algunos guardias son guerreros.
b) Algunos guerreros son cobardes.

Por lo tanto concluimos que: Algunos guardias deben ser cobardes.

La conclusión es:
Correcta _____ Incorrecta

4.- Supongamos que los siguientes argumentos son verdaderos:

a) Todos los desarrolladores son ingenieros.
b) Todos los ingenieros son listos.

Concluimos que: Todos los desarrolladores son listos.

Nuestra conclusión es:
Correcta _____ Incorrecta _____

5.- Supongamos que los siguientes argumentos son verdaderos:

a) Algunos criminales son millonarios.
b) Todos los empresarios son millonarios.

Concluimos que: Algunos empresarios deben ser criminales.

Nuestra conclusión es:
Correcta _____ Incorrecta _____

Razonamiento Matemático

6.- Los tres lados de un triángulo están en la razón de 1 : 3 : 5. El perímetro del triángulo es 72. Encuentra la longitud de los tres lados.

Lado 1:_
Lado 2:_
Lado 3:

7.- Juan es tres veces más grande que Nacho, y Nacho tiene la mitad de la edad que Beto. José es dos veces más viejo que la edad de Nacho y Beto sumadas. Si José tiene 60 años, cuantos años tiene su prima Martha que es dos años mayor que Nacho?

Edad de Martha:_____

8.- En el siguiente arreglo de números, cual es el número que aparecerá al inicio de la línea número 100?

1 2 3
4 5 6 7
8 9 10 11 12
13 14 15 16 17 18

Tu respuesta:_

GRADO *DE DIFICULTAD:*
Implementaclón: N/A Algoritmo: 2 y 4 respectivamente

9.- Se tiene una caja cúbica de 10 x 10 x 10 sin tapa. Esta repleta de cubos de dimensión 1x1x1. Cuantos cubos unitarios están en contacto con alguno de los lados de la caja?

Tu respuesta:_____

10.- Dibuja 5 lineas rectas en cualquier forma que desees de modo que el número de áreas cerradas diferentes sea el máximo.

11.- María tiene 4 años, su hermana Martha tiene tres veces su edad. Que edad tendrá Martha cuando su edad sea el doble de la de María?

Edad de Martha:

12.- En una convención hay 100 industriales, 85 de ellos traen teléfono celular, 80 traen un localizador, 75 hablan por lo menos dos idiomas y 70 están vestidos de traje. De estos industriales, cual es el número mínimo de ellos que seguro cumple con traer todo lo anterior?

Tu respuesta:_____

Analogías:

Selecciona el número que mejor complete la analogía, por ejemplo

7:10::6:9

13.-

a) 2
b) 1
c) -1
d) 12
e) 4

14.- 20: 12:: 5: ?

a) 3
b) 15/4
c) 3.5
d) 2
e) 5/3

15.- 4 :256:: 5 : ?

a) 526
b) 625
c) 125
d) 726
e) 620

16.- 322 : 12 :: 421 : ? a)
15
b) 7
c) 8
d) 15
e) 9

17.- 91827 : 364554 :: 369 :?

a) 46835
b) 151821
c) 84962
d) 121518
e) 163472

18.- 48:610::39:¿

a) 362
b) 975
c) 511
d) 602
e) 353

Secuencias:

Indica el número que debe seguir en la secuencia.

19.- 2,5, 8, 11, 12, 15, 18,21 .-.. ?

Tu respuesta: _____

20.- 1,-4, 8, 3, -6,-11, 22, 17 ... ?

Tu respuesta: _____

21.- 2, 6, 3, 1, 3, O, O, O, -3, -1, -3, -6, -2 ... ?

Tu respuesta: _____

22.- -2, $^1/_2$, -1, -1/2,^{1}/2, -1/4, -1/8, 1/32 ... ?

Tu respuesta: _____

23.- 225, 4, 196, 9, 169, 16, 144, 25 ... ?

Tu respuesta: _____

24.- 8, 1, 3,9,2,4, 10,3,5, 11 … ?

Tu respuesta: _____

25.- 1,2, 3,7, 11,20,29,45,61 … ?

Tu respuesta: _____

GRADO DE DIFICULTAD:
Implementación: N/A Algoritmo: 2 y 4 respectivamente

Algoritmos

En esta sección se examinará tu capacidad para el desarrollo de algoritmos que puedan solucionar un problema dado. Ambos problemas tienen el mismo valor. Se calificará la claridad con la que establezcas los pasos a seguir para la solución del problema, por lo que se te recomienda que seas lo mas conciso posible. ¡Mucha suerte!

NOTA: La eficiencia de tus algoritmos será tomada en cuenta, es muy importante que escribas el mejor algoritmo que encuentres (por mejor algoritmo se entiende aquel que llega a la respuesta en el menor número de pasos). Puedes escribir mas de un algoritmo para resolver un mismo problema, así es que si se te ocurre una solución que funcione aunque no sea la mas eficiente, escríbela y sigue pensando otra, de todas las soluciones que encuentres, se tomará en cuenta solo la mejor.

1.- LA PAREJA MÁS CERCANA.

Dada una lista de números enteros positivos que puede contener desde 5 hasta 1000 elementos, por ejemplo (4,6,1,90,56,4379,23) se desea saber, cuáles dos de ellos son los mas cercanos en valor, es decir, que el valor absoluto de su diferencia sea menor que el de cualquier otra pareja de números en la lista. Para la lista anterior, la pareja más cercana son 4 y 6, ya que ningún otro par de números tiene una resta cuyo valor absoluto sea menor a 2 (6-4=2). Escribe de la manera más eficiente posible, un algoritmo que permita, mediante el uso de una computadora obtener la pareja más cercana de una lista dada.

2.- LA DISTANCIA MÁS CORTA.

Este problema es muy similar al anterior, salvo que ahora en lugar de números se tiene una lista de puntos sobre un plano. El número de puntos en el plano puede ser hasta 10,000. Escribe un algoritmo que mediante el uso de una computadora encuentre la pareja de puntos mas cercana, es decir, que la distancia entre ambos puntos sea menor que para cualquier otra pareja de puntos de la lista.

GRADO DE DIFICULTAD: Implementación: N/A Algoritmo: 2 y 4 respectivamente

EXÁMENES NACIONALES

1995 - 2002

EXÁMENES DEL CONCURSO NACIONAL DE INFORMÁTICA 1995
(8-12 de mayo, 1995)

Codiciosos dadores de regalos

Descripción

Tienes que determinar, de un grupo de amigos dadores de regalos, cuánto más dio cada persona de lo que recibió (y viceversa, problema pensado para aquellos que ven el dar regalos con cinismo). En este problema, cada persona aparto dinero para los regalos que tiene que dar y dividió dicho dinero en cantidades ¡guales para todos aquellos a los que debe dar regalo. Sin embargo, en cualquier grupo de amigos, algunas personas son más espléndidas que otras y algunas tienen más dinero que otras.

Dado un grupo de amigos, el dinero que cada persona del grupo gastó en regalos y una lista de amigos a quienes cada persona debe dar regalos, tienes que escribir un programa que determine cuánto más (o menos) cada persona dio comparado con lo que ellos recibieron.

Entrada

La entrada es un grupo de dadores de regalos el cual consiste en varias líneas:

- el número de personas del grupo
- una lista de los nombres de cada persona del grupo
- una línea por cada persona en el grupo, que consiste en el nombre de la persona, el monto de dinero que gastó en regalos, el número de personas a quienes les fueron dados los regalos y los nombres de esas personas a quienes les fueron dados los regalos.

Todos los nombres deben estar en letras minúsculas, no hay más de 10 personas en el grupo y los nombres no son mayores a 12 caracteres de longitud. El dinero es un entero no negativo y menor que 2000.

Salida

El nombre de cada persona en el grupo debe ser impreso en una línea seguido de la cantidad total ganada (o pérdida) por la persona. Los nombres en el grupo deberán ser impresos en el mismo orden en el cual ellos aparecen en la lista.

Todos los regalos son enteros. Cada persona da la misma cantidad entera de dinero a cada amigo y da lo más posible. Cualquier dinero que no es regalado, es retenido por la persona y debe ser considerado al calcular el total ganado o perdido de dicha persona.

Ejemplos

Ejemplo 1	Ejemplo 2
Entrada	Entrada
5 dave laura owen vick amr dave 200 3 laura owen vick owen 500 1 dave amr 150 2 vick owen laura 0 2 amr vick vick 0 0	3 liz steve dave liz 30 1 steve steve 55 2 liz dave dave 0 2 steve liz
Salida	Salida
Dave 302 laura 66 owen -359 vick 141 amr -150	liz -3 steve -24 dave 27

instrucciones

Tu programa deberá llamarse REGALOS.C, si lo escribes en Turbo C, REGALOS.CPP, si lo escribes en Turbo C++, REGALOS.PAS, si lo escribes en Turbo Pascal. Deberá ¡eer su archivo de entrada llamado REGALOS.DAT y escribir su archivo de salida REGALOS.SOL. Los archivos del ejemplo están en tu directorio de trabajo. Los nombres de estos archivos son PRUEBA1.DAT y PRUEBA2.DAT. La solución de estos ejemplos está en los archivos PRUEBA1 .SOL y PRUEBA2.SOL. Si tu programa genera resultados correctos para estos archivos, no significa forzosamente que está correcto.

GRADO DE DIFICULTAD:
Implementación: S Algoritmo 1

El gato en el sombrero

Descripción

Un inteligente gato camina dentro de un cuarto sucio el cual necesita limpiar. En lugar de hacer el trabajo solo, decide tener sus gatos que le ayuden a hacer el trabajo. El retiene sus (pequeños) ayudantes gatos dentro del sombrero. Cada gato ayudante también tiene sus ayudantes gatos en su propio sombrero y así sucesivamente. Eventualmente, los gatos hacen contacto con los de tamaño más pequeño. Los gatos más pequeños no tienen gatos adicionales en sus sombreros y desafortunadamente tienen que hacer la limpieza. El número de gatos dentro de cada sombrero del gato (no del más pequeño) es una constante N. La altura de los gatos es 1/(N+1) veces la altura del gato cuyo sombrero están dentro. Todas las alturas son números enteros positivos.

Dadas la altura del gato inicial y el número de gatos trabajadores (de altura uno), encuentra el número de gatos que no van a hacer ningún trabajo (gatos cuya altura es mayor a uno) y también determina la suma de todas las alturas de los gatos (la altura de una pila de gatos estando uno sobre otro).

Entrada

La entrada es una sola línea que consiste de dos enteros positivos, separados por un espacio en blanco. Él primer entero es la altura inicial y el segundo entero es el número de gatos trabajadores.

Salida

Imprime el número de gatos que no van a trabajar, seguido de un espacio, el cual a su vez está seguido de la altura de la pila de gatos.

Ejemplos

Ejemplo 1	Ejemplo 2
Entrada	Entrada
216 125	5764801 1679616
Salida	Salida
31 671	335923 30275911

Instrucciones

Tu programa deberá llamarse GATOS.C, si lo escribes en Turbo C, GATOS.CPP, si lo escribes en Turbo C++, GATOS.PAS, si lo escribes en Turbo Pascal. Deberá leer su archivo de entrada llamado GATOS.DAT y escribir su archivo de salida GATOS.SOL. Los archivos de ejemplo están en tu directorio de trabajo. Los nombres de estos archivos son PRUEBA1.DAT y PRUEBA2.DAT. La solución de estos ejemplos está en los archivos PRUEBA1.SOL y PRUEBA2.SOL Si tu programa genera resultados correctos para estos archivos, no significa que está correcto.

GRADO DE DIFICULTAD:
Implementación:* S *Algoritmo: 2

PRIMERA OLIMPIADA MEXICANA DE INFORMÁTICA 1996
MONTERREY, N.L. 3 A 6 DE MAYO

Examen del Concurso Nacional 1996

El examen del Concurso Nacional de la Primera Olimpiada Mexicana de Informática consistió de dos días de pruebas para resolver dos problemas cada día en cuatro horas. El Comité Organizador propuso los siguientes seis problemas al jurado:

- **Primer día:** Pirámide y Taxis
- **Segundo día:** Mudanza y Mina
- **No usados:** Circuito y Pento

Olimpiada Mexicana de Informática 1996

Pirámide

Descripción

Un matemático muy ingenioso tiene una pirámide de base triangular cuyas aristas tienen una longitud L, siendo L un número entero. En el interior de la pirámide se encuentran números enteros acomodados en L capas como se muestra en la figura 1. Además, el matemático notó que se pueden formar pirámides más pequeñas en su interior, algunas de ellas formadas exclusivamente por números primos (subpirámides primas). Sin embargo, llevar la cuenta de todas las subpirámides primas de arista de longitud M (2 <= M <= L) que existen, es muy tedioso. Por eso, el matemático ha decidido que le hagas un programa que realice esta tarea.

Entrada

El archivo de texto INPUT.TXT contiene los siguientes datos: en la primera línea se encuentra la longitud L (3 <= L <= 30) de los lados de la pirámide. En las líneas subsecuentes, se encuentran los números (1 <= N <= 32757) que contienen cada una de L capas de la pirámide comenzando con la punta.

Salida

El archivo de texto OUTPUT.TXT consta de L-1 líneas que contienen la longitud de los lados (en orden ascendente) de las pirámides de números primos que pueden ser formadas, y la cantidad de pirámides con dicha longitud que existen en el interior.

Ejemplo

INPUT.TXT	OUTPUT.TXT
4 6 5 7 11 13 7 11 5 17 20 29 3 13 5 7 11 13 21 17 8	2 5 3 1 4 0

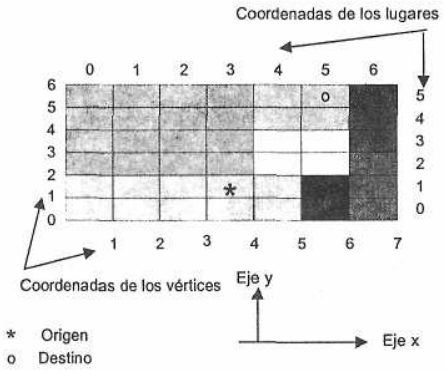
GRADO DE DIFICULTAD:
Implementación: SE Algoritmo: 4

Olimpiada Mexicana de Informática 1996

Taxis

Descripción

Juan llega a la central de autobuses de Taxtlán (para permanecer allí durante N días). Como él no conoce la ciudad, decide comprar un mapa e inmediatamente encuentra el hotel "candiles" donde se desea hospedar, así que se dirige al sitio de taxis de la central. Allí descubre que los taxis cobran de una manera muy peculiar: la ciudad se ha dividido en N regiones rectangulares y el precio de un viaje entre dos puntos en dos regiones distintas es igual a la cantidad de fronteras que cruza el taxi para llegar de un lugar a otro (donde una frontera es un segmento común entre dos regiones contiguas). Además, se entera que los taxistas se niegan a dar viajes gratis, así que para moverse entre dos puntos en la misma región se tendrá que ir caminando (lo cual no le cuesta nada). El marca las regiones en su mapa (como en la figura 1) ya que con esta información puede averiguar cuanto es lo menos que le costaría un viaje en taxi entre cualesquiera dos lugares en Taxtlán. Finalmente Juan se va a su hotel.



Ya en el hotel estudia el mapa con más calma y descubre que en cada región hay al menos un punto de interés y toma la decisión de que cada día que esté en Taxtlán escogerá una región para visitar, para que al final de su estancia, naya visitado todos los lugares atractivos de la ciudad. Como Juan tiene su mapa el llegará a cada una de las regiones caminando, y después de terminar su visita a esa región regresará al hotel en taxi (de ser posible).

Escribe un programa que:

1. Calcule el costo mínimo M del viaje de la central al hotel "candiles".
2. Si Juan hubiera notado que en cada región también hay un hotel en el cual el se puede hospedar durante los N días, calcule el costo mínimo posible V para visitar todas las regiones según su plan.

Entrada

En el archivo de texto INPUT.TXT, en la primera línea el entero N (1 <= N <= 100), en cada una de las siguientes N líneas las coordenadas de dos vértices opuestos de cada región (coordenadas de los vértices), en la última línea las coordenas de la central y el hotel "candiles", respectivamente. Supondremos que la ciudad tiene forma rectangular y que todas las coordenadas de los lugares son enteros entre O y 100 (inclusive).

Salida

En el archivo de texto OUTPUT.TXT, en la primera línea el entero M y en la segunda el entero V.

Ejemplo

INPUT.TXT	OUTPUT.TXT
7 0 0 5 2 0 2 4 6 5 0 6 2 4 4 6 6 4 2 6 4 6 3 7 0 7 6 6 3 3 1 5 5	2 6

GRADO *DE DIFICULTAD:*
Implementación: SE Algoritmo: 3

Olimpiada Mexicana de Informática 1996

Mudanza

Descripción

Una familia va a cambiar de residencia y para ello va a utilizar dos camiones de mudanza. La empresa de mudanzas desea que todas las pertenencias de !a familia se repartan equitativamente entre los dos camiones de manera que:

1. ambos transporten el mismo número de objetos, y
2. el peso que carga un camión sea lo más parecido al peso que carga el otro, es decir, la diferencia entre los pesos debe ser la mínima.

Escribe un programa que dada la lista de los pesos de las pertenencias de la familia, determine el peso que cada camión deberá llevar para cumplir con los requerimientos de la empresa. Nota: No hay límite para el peso que pueden cargar los camiones.

Entrada

La entrada deberá realizarse del archivo de texto INPUT.TXT . La primera línea contiene el número N de pertenencias de la familia (10 <= N <= 10000, con N par). Las siguientes N líneas contienen el peso de cada uno de los objetos representado como un entero positivo menor o igual a 60000.

Salida

En el archivo de texto OUTPUT.TXT deberá escribirse el peso que cargará cada uno de los camiones, en cualquier orden, separados por un espacio. Ambos pesos son números enteros.

Ejemplo

INPUT.TXT	OUTPUT.TXT
10 5320 21 34 17 100 286 9870 17605 12 295	15871 17689

GRADO DE DIFICULTAD:
Implementación: C Algoritmo: S

Olimpiada Mexicana de Informática 1996

Mina

Descripción

Un minero afortunado se encuentra el mapa de una mina encantada la cual tiene N niveles, en cada nivel hay M grutas y en cada gruta hay una bolsa que puede contener monedas de oro. Viendo el mapa, el minero se percata que la mina también tiene M entradas y que al salir de cada gruta, sólo puede ir a la gruta que está en el siguiente nivel a la derecha, a la izquierda o enfrente. El mapa tiene escrita una advertencia: "Cuando un ser mortal avance al siguiente nivel, aquél en el que estaba será destruido..." y como buen mortal, le hará caso a esta advertencia. El minero decide entrar a la mina para salir con tantas monedas como sea posible.

En la figura 1 se muestra el mapa de una mina con 6 niveles y 9 grutas por nivel. Las entradas han sido numeradas del 1 al 9 como lo indica la figura. En este caso, el minero puede recoger un máximo de 41 monedas de oro, comenzando en cualquiera de las entradas 2 ó 4 y siguiendo las rutas marcadas en el mapa.

Escribe un programa para ayudar al minero a:

1. Encontrar la cantidad máxima de monedas que puede recoger.
2 Decidir por cuál o cuáles de las entradas de la mina debe iniciar su recorrido para obtener la mayor fortuna.

Entrada

En el archivo de texto INPUT.TXT, en la primera línea los enteros N y M (2<=N<=100 y 2<=M<= 100), en cada una de las siguientes N lineas las M cantidades de monedas de oro que hay en las bolsas en cada nivel. Cada bolsa puede contener hasta 100 monedas.

Salida

En el archivo de texto OUTPUT.TXT, en la primera línea la cantidad máxima de monedas que puede obtener el minero, en la segunda línea los números de las entradas distintas por la que puede comenzar, en orden creciente, separados por un espacio.

Ejemplo

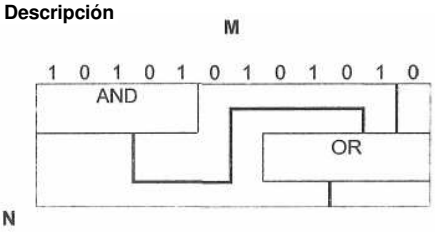
INPUT.TXT	OUTPUT.TXT
6 9 1 8 2 5 3 0 6 3 4 0 2 4 0 8 4 4 6 0 8 2 2 6 0 1 4 5 9 6 1 7 0 1 2 5 1 2 9 7 0 2 3 6 7 3 8 2 8 5 7 4 7 1 5 0	41 2 4

GRADO DE DIFICULTAD:
Implementación: S Algoritmo: 3

Olimpiada Mexicana de Informática 1996

Circuito

Descripción



Un estudiante de electrónica tiene como tarea el implementar distintos circuitos que utilicen compuertas lógicas (AND, OR, XOR, NOT) conectadas en serie y reportar la salida de cada una de ellas. Sin embargo, el realizar dicha implementación físicamente puede ser muy complicado, por lo que ha decidido programar un simulador que le ayude a probar los circuitos. Para simplificar el proceso, el estudiante decide usar las siguientes abreviaturas:

C: material conductor R:
material no conductor

Así mismo, define las compuertas lógicas de la siguiente manera:

PETEP
PPSPP

En donde "P" es la cubierta, "E" son las entradas, "S" la salida y "T" queda definida de la siguiente manera:

A: AND, O: OR
N: NOT, X: XOR

Todas las compuertas lógicas reciben su entrada de la casilla que se encuentra localizada inmediatamente arriba de la "E", y colocan la salida en la casilla que se encuentra inmediatamente abajo de la "S". (La compuerta NOT, solamente utiliza la "E" de la izquierda). En la simulación, las posiciones nones de la parte superior del circuito reciben un "1" lógico, y las posiciones pares reciben un "O" lógico. Así mismo, no existe ningún corto circuito. Dada una representación del circuito mediante las abreviaciones en una matriz de M * N, tienes que hacer el programa que determine la salida de cada una de las compuertas lógicas que contiene.

Entrada

El archivo de texto INPUT.TXT contiene los siguientes datos: en la primera linea se encuentran dos enteros M y N. (4 <= M <= 80, 6 <= N <= 100). En las siguientes M líneas, se encuentra la matriz que contiene el circuito.

Salida

En el archivo de texto OUTPUT.TXT, debes indicar en la primera línea, el número de compuertas que se encuentran en el circuito (subtask a). En las siguientes líneas deberás indicar en el mismo orden en el que se encuentran la esquina superior izquierda de las compuertas de izquierda a derecha, el nombre de la compuerta y su salida (subtask b).

Ejemplo

INPUT.TXT	OUTPUT.TXT
5 12 PEAEPRRRRRCR PPSPPRCCCRCR RRCRRRCPEOEP RRCCCCCPPSPP RRRRRRRRRCRR	2 AND 0 OR 1

GRADO DE DIFICULTAD:
Implementación: CE Algoritmo: 2

Olimpiada Mexicana de Informática 1996

Pento

Descripción

¿Te acuerdas de las fichas de los dóminos? Son esas fichas formadas pegando dos cuadrados. Si no fuera por los puntos que les ponen, todos los dóminos serian iguales. Sin embargo, ¿qué *pasaría* sí en lugar de pegar dos cuadros nos permitieran pegar tres, cuatro o cinco? No es muy difícil ver que con cinco cuadros obtendríamos las fichas mostradas en la figura 1, que reciben el nombre de "pentominós". Como puedes ver, existen 12 pentominós diferentes (si nos olvidamos de que se pueden girar en múltiplos de 90 grados y reflejar). Cada una de esas doce fichas tiene un nombre, el cual está dado por la letra mayúscula a la que se parecen más.

Cierto día, una persona jugaba con unos pentominós construidos en madera y se dio cuenta de lo siguiente: se puede llenar un rectángulo de 10 (6 con una copia de cada una de estas piezas. Pronto descubrió que esto se podía lograr de muchas formas e inventó el siguiente juego: Tomemos un tablero de 10x6 en el cual alguien ha puesto algunos pentominós (al menos dos). El juego consiste en usar los demás pentominós para llenar todo el tablero. En este juego si es permitido girar o reflejar los pentominós, pero no se puede usar dos o más veces el mismo, ni tampoco mover las piezas puestas inicialmente. En la figura 2 se muestra un ejemplo en el cual alguien ha colocado los pentominós T y U en el tablero y una posible forma de llenar el tablero.

Escribe un programa que a partir de una configuración inicial del tablero, encuentre una forma de llenarlo con pentominós.

Nota: Todos los casos de prueba tendrán al menos una solución. En caso de existir más de una, tu programa sólo deberá desplegar una de ellas.

Entrada

En el archivo de texto INPUT.TXT, seis líneas de diez caracteres cada una, estos caracteres pueden ser puntos "." (denotando a las posiciones vacías del tablero) o alguna de las letras que corresponden con los nombres de los pentominós (las cuales denotan a las piezas puestas inicialmente en el tablero).

Salida

En el archivo de texto OUTPUT.TXT, seis líneas de diez caracteres cada una describiendo (de la misma manera que en la entrada) la configuración final del tablero.

Ejemplo

INPUT.TXT	OUTPUT.TXT
..... .TTT..... ..T..... ..T...U.U. .....UUU.	YIIIIIMMPP YTTTNMMXPP YYTNNMXXXP YFTNZZUXUV FFFNLZUUUV FLLLLZZVVV

GRADO DE DIFICULTAD:
Implementación: CE Algoritmo: 1

SEGUNDA OLIMPIADA MEXICANA DE INFORMÁTICA 1997
MÉXICO, D.F. 30 DE JUNIO A 3 DE JULIO

Examen del Concurso Nacional 1997

El examen del Concurso Nacional de la Segunda Olimpiada Mexicana de Informática consistió de dos días de pruebas para resolver dos problemas cada día en cuatro horas. El Comité Organizador propuso tos siguientes cuatro problemas al jurado:

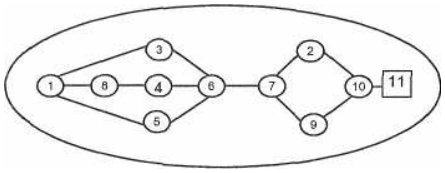
- **Primer día:** Submarino y Átomos
  Segundo día: Cintas y Superconejo

2ª. Olimpiada Mexicana de Informática 1997

Submarino

Descripción

Un submarino tiene N secciones numeradas del 1 al N (2<= N <= 100). De estas secciones, algunas se encuentran interconectadas entre sí. Además se sabe que hay un total de E interconexiones y que la sección N es la salida de emergencia.



Subproblema A

Si hubiera un marinero en cada sección, excepto la de salida, y si éste tardara una unidad de tiempo en llegar a una sección adyacente, determina cuántos marineros llegarían a la salida de emergencia en

menos de T (1 <= T <= 100) unidades de

tiempo. **Subproblema B**

Determina cuántas y cuáles secciones, con excepción de la salida, son *críticas.* Es decir, qué secciones, independientemente del tiempo T, impedirían que algunos marineros llegaran a la salida de emergencia si éstas estuvieran obstruidas.

Entrada

La primera línea del archivo de texto INPUT.TXT contiene los enteros N, E y T. Las siguientes E líneas contienen, cada una, los números de dos secciones interconectadas entre sí.

Salida

El archivo de texto OUTPUT.TXT deberá contener en su primera línea el número de marineros que pueden llegar a la salida de emergencia en menos de T unidades de tiempo. El segundo renglón deberá indicar la cantidad de secciones críticas, seguido de la lista correspondiente a las secciones.

Ejemplo

INPUT.TXT	OUTPUT.TXT
11 13 5 1 3 1 5 1 8 2 7 2 10 3 6 4 6 5 6 6 7 7 9 8 4 9 10 10 11	5 3 6 7 10

GRADO DE DIFICULTAD: Implementación: SE Algoritmo: 4

2ª. Olimpiada Mexicana de Informática 1997

Átomos

Descripción

Un nuevo compuesto requiere exactamente N1, N2 y N3 átomos (O <= N1, N2, N3 <= 16) de tres elementos químicos distintos E1, E2 y E3 respectivamente.

Un proveedor vende átomos individuales de estos tres elementos a precios P1, P2 y P3 por átomo del elemento respectivo (1 <= P1, P2, P3 <= 500). Dicho proveedor tiene suficiente inventario para cubrir completamente cualquier pedido.

Otro proveedor ofrece K compuestos (O <= K <= 100) formados por átomos de estos tres elementos *a un* menor precio. Sin embargo tiene inventario limitado para cada uno de estos compuestos, por lo que no se podrían conseguir todos los átomos con solamente uno de estos compuestos.

El nuevo compuesto requiere de los átomos independientemente de la forma en la que éstos se encuentren combinados. Por consiguiente se debe encontrar el costo mínimo para comprar los átomos. No se deben comprar más átomos de los que se necesitan para formar una unidad del nuevo compuesto.

En el ejemplo, el costo mínimo se obtiene al comprar dos unidades del primer compuesto, y una unidad tanto del segundo como del tercer compuesto. Al proveedor de átomos individuales se le comprarían dos átomos del elemento E3 y un átomo del elemento E1.

Entrada

La primera línea del archivo de texto INPUT.TXT contiene N1, N2 y N3. El segundo renglón contiene P1, P2 y P3. La tercera línea contiene K y los siguientes K renglones contienen una lista de cinco enteros cada uno. De estos enteros, los primeros tres corresponden, en ese orden, a la cantidad de átomos de los elementos E1, E2 y E3 que contiene el compuesto combinado, y los últimos dos corresponden a la cantidad en inventario y al costo del compuesto, respectivamente.

Salida

El archivo de texto OUTPUT.TXT deberá contener en su primera linea el costo mínimo para comprar los átomos del nuevo compuesto.

Ejemplo

INPUT.TXT	OUTPUT.TXT
6 3 4 8 9 7 3 2 1 0 2 18 0 1 1 2 14 1 0 1 1 11	83

GRADO DE DIFICULTAD:
Implementación: CE Algoritmo: 5

2ª. Olimpiada Mexicana de Informática 1997

Cintas

Descripción

Tu colección de discos compactos de música moderna ha aumentado considerablemente y ya no es fácil llevarla a todas las fiestas. Por eso has decidido grabar tus discos en cintas, las cuales son más fáciles de transportar.

Tienes N (1 <= N <= 25) discos de duraciones O < T1 < T2 < ... < TN <= 200 minutos cada uno de ellos.

Para grabarlos puedes comprar K (1 <= K <= 10) clases distintas de cintas con precios O < P1 < P2 < ... < PK <= 100 pesos por cinta y duraciones O < L1 < L2 < ... < LK <= 100 minutos por lado (cada cinta tiene dos lados). El precio PJ corresponde con la cinta de duración LJ minutos por lado. Se sabe también que el disco más extenso cabe en la cinta de mayor duración.

La forma en que grabes tus discos debe satisfacer que:

- Todo disco quede grabado por completo en una cinta (utilizando uno o los dos lados).
- Si un disco ocupa sólo un lado de la cinta, entonces se puede ocupar el otro lado para grabar otro disco o dejarlo vacío.
- En cada lado de las cintas no se puede grabar más de un disco.

Subproblema A

Encuentra el costo total mínimo V de las cintas en las que puedes grabar todos tus discos.

Subproblema B

Adicionalmente calcula la cantidad mínima C de cintas en las que puedes grabar todos tus discos y que puedes comprar con V pesos.

Por ejemplo, si tuvieras 5 discos de duraciones 25, 33, 47, 55 y 74 minutos cada uno, y existieran 3 clases de cintas con duraciones de 24, 36 y 51 minutos por lado y precios de 36, 52 y 72 pesos por cinta, respectivamente, entonces el costo mínimo de las cintas necesarias seria de 212 pesos. En este caso, debes grabar tus discos en cuatro cintas como se indica en la siguiente tabla:

Disco	Cinta
25 minutos	36 minutos (lado A)
33 minutos	36 minutos (lado B)
47 minutos	24 minutos (ambos lados)
55 minutos	36 minutos (ambos lados)
74 minutos	51 minutos (ambos lados)

Entrada

En el archivo de texto INPUT.TXT estarán en la primera línea los valores de N y K. En la segunda línea los N valores T1, T2, ..., TN. En la tercera los K valores L1, L2, ..., LK. En la cuarta los K valores P1, P2, ..., PK. Todos estos valores serán números enteros separados por un espacio, escritos en el orden mencionado.

Salida

En el archivo de texto OUTPUT.TXT deberás escribir V en la primera línea y C en la segunda línea.

Ejemplo

INPUT.TXT	OUTPUT.TXT
5 3 25 33 47 55 74 24 36 51 36 52 72	212 4

GRADO DE DIFICULTAD:
Implementación: S Algoritmo: 4

2ª. Olimpiada Mexicana de Informática 1997

Superconejo

Descripción

Un campo está dividido en M secciones horizontales por N secciones verticales (2 <= M, N <= 100).

Los saltos de un *superconejo* de una sección a otra de este campo se pueden ver como un movimiento de P secciones en una dirección horizontal o vertical seguido de un movimiento de Q secciones en dirección perpendicular a la seleccionada anteriormente. Cuando se mueve horizontalmente lo puede hacer hacia la izquierda o hacia la derecha y cuando se mueve verticalmente lo puede hacer hacia arriba o hacia abajo.

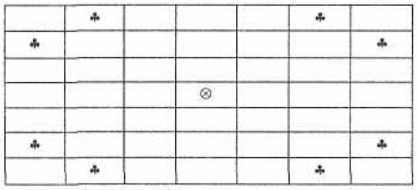


Fig.1 Saltos de un Superconejo con P=2 y Q=3.

Subproblema A

Colocamos un superconejo en la sección (X,Y) y queremos que llegue a otra sección distinta (U,V) (1 <= X, U M y 1 <= Y, V <= N), usando los saltos descritos anteriormente. Determina la cantidad mínima S de saltos que se necesitan para lograr el objetivo.

Subproblerna B

Calcula el número total T de secciones a las que puede llegar el superconejo si comienza a saltar desde la sección (X,Y).

Por ejemplo, considera un campo de 7 secciones horizontales por 4 secciones verticales. El superconejo se encuentra en la sección (3,1) y queremos que llegue a la sección (4,3). Usando saltos de 2 por 3 secciones se requieren 3 de ellos, y se puede llegar a un total de 14 secciones.

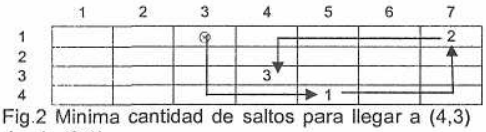



Fig.2 Minima cantidad de saltos para llegar a (4,3) desde (3,1).

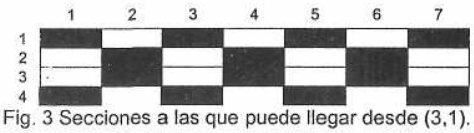


Fig. 3 Secciones a las que puede llegar desde (3,1).

Entrada

En el archivo de texto INPUT.TXT encontrarás M y N en la primera línea, P y Q en la segunda línea, X y Y en la tercera línea y U y V en la cuarta línea.

Salida

En el archivo de texto OUTPUT.TXT debes escribir S en la primera línea y T en la segunda linea.

Ejemplo

INPUT.TXT	OUTPUT.TXT
7 4 2 3 3 1 4 3	3 14

GRADO DE DIFICULTAD:
Implementación: SE Algoritmo: 2

TERCERA OLIMPIADA MEXICANA DE INFORMÁTICA 1998
GUADALAJARA, JALISCO, 24 A 28 DE ABRIL

Examen del Concurso Nacional 1998

El examen del Concurso Nacional de la Tercera Olimpiada Mexicana de Informática consistió de dos días de pruebas para resolver dos problemas cada día en cuatro horas. El Comité Organizador propuso los siguientes cuatro problemas al jurado:

Primer día: Fichas y Cálculos
Segundo día: Sumas y Explora

Tercera Olimpiada Mexicana de Informática 1998

Fichas

Descripción del problema

Fichas es un juego de tablero para dos jugadores cuyo objetivo es ser el último en retirar fichas del tablero. El tablero de *Fichas* consta de cuatro renglones con 10 casillas cada uno. En cada casilla puede haber a lo más una ficha. Al inicio del juego se distribuyen una o más fichas en el tablero. Durante el juego ambos jugadores de manera alternada retirarán fichas del tablero de una de las dos formas siguientes:

- Un jugador podrá retirar la misma cantidad de fichas de dos renglones diferentes.
- Un jugador podrá retirar sólo una ficha de cualquier renglón.

Es decir, si se toma ficha de un renglón se podrá tomar únicamente una, si se toman fichas de dos renglones se podrán tomar una o más fichas, siempre la misma cantidad de ambos renglones.

Problema

Escribe un programa que, dada una configuración del tablero sea capaz de encontrar, si es que la hay, la siguiente jugada que asegure la victoria al jugador en turno, siempre y cuando continúe jugando de manera óptima.

Ejemplo

Para el siguiente tablero una posible jugada que asegura la victoria es retirar dos fichas de los renglones 2 y 3.

Renglón 1				
Renglón 2	*	*		
Renglón 3	*	*		
Renglón 4	*	*	*	*

Entrada

Tu programa deberá leer del archivo de entrada INPUT.TXT cuatro números enteros. Cada número corresponde a ¡a cantidad de fichas en cada renglón del tablero.

Salida

Tu programa deberá escribir en el archivo de salida OUTPUT.TXT cuatro números enteros. Cada número corresponde al número de fichas que se retiran de cada renglón. Para los casos en que halla más de una solución, tu programa deberá listar únicamente una de ellas. En el caso de que no exista tal jugada deberá escribir cuatro ceros.

Ejemplo de entrada y salida

Los siguientes archivos corresponden al ejemplo descrito.

INPUT.TXT	OUTPUT.TXT
0 2 2 4	0 2 2 0

GRADO DE DIFICULTAD:
Implementación: SE Algoritmo: 4

Tercera Olimpiada Mexicana de Informática 1998

Cálculos

Descripción del problema

Tienes una calculadora de 4 dígitos decimales que sólo puede realizar 2 operaciones: multiplicar por A y dividir entre B. Si el resultado de multiplicar un número por A es un número de más de 4 dígitos la calculadora da como resultado 1. Si el resultado de dividir entre B no es un número entero, entonces la calculadora trunca el resultado entregando únicamente la parte entera. Por ejemplo, si A = 2 y B = 3 entonces 20*A = 40 y 20/B = 6 mientras que 6,000*A = 1 y 6,000/B = 2,000. La calculadora siempre comienza con el número 1 y almacena el último resultado obtenido para utilizarlo en la siguiente operación.

Problema

Escribe un programa que dados A y B encuentre el número mínimo de pasos que se tienen que realizar con la calculadora para obtener un número N comenzando en el 1 y utilizando únicamente las dos operaciones válidas.

Ejemplo

Dados A = 2 y B = 3, para obtener el 10 se requiere un mínimo de 6 pasos:

1*A = 2
2*A = 4
4*A = 8
8*A=16
16/B = 5
5*A=10

Entrada

En el primer renglón del archivo INPUT.TXT los enteros A, B y N, en ese orden, donde 1 <= A, B, N <= 9,999

Salida

En el primer renglón del archivo OUTPUT.TXT la cantidad mínima de pasos para obtener N.

Ejemplo de entrada y salida.

INPUT.TXT	OUTPUT.TXT
2 3 10	6

GRADO DE DIFICULTAD: Implementación: SL Algoritmo: 2

Tercera Olimpiada Mexicana de Informática 1998

Sumas

Descripción del problema.

Si tomamos un entero positivo N lo podemos escribir de M formas distintas como suma de K enteros positivos elevados al cuadrado. Dos formas se consideran iguales si sólo difieren en el orden de los sumandos. Nota: El O no es un entero positivo.

Ejemplo.

El número 98 se puede escribir de 6 formas distintas como suma de 5 números elevados al cuadrado:

$98 = 1^2 + 1^2 + 4^2 + 4^2 + 8^2$
$98 = 1^2 + 2^2 + 2^2 + 5^2 + 8^2$
$98 = 1^2 + 3^2 + 4^2 + 6^2 + 6^2$
$98 = 1^2 + 4^2 + 4^2 + 4^2 + 7^2$
$98 = 2^2 + 2^2 + 4^2 + 5^2 + 7^2$
$98 = 4^2 + 4^2 + 4^2 + 5^2 + 5^2$

La forma $1^2 + 2^2 + 2^2 + 5^2 + 8^2$ y la forma $2^2 + 8^2 + 1^2 + 2^2 + 5^2$ son la misma porque sólo difieren en el orden de los sumandos, por lo cual, en el ejemplo, se contó sólo una de ellas.

Problema.

Escribe un programa que, dados N y K, encuentre el número M de formas en que se puede escribir N como la suma de K números elevados al cuadrado.

Entrada.

En el primer renglón del archivo INPUT.TXT los enteros N y K, donde 1 <= N <= 10,000 y 1 <= K <= 6.

Salida.

En el primer renglón del archivo OUTPUT.TXT el entero M.

Ejemplo de entrada y salida.

INPUT.TXT	OUTPUT.TXT
98 5	6

GRADO DE DIFICULTAD:
Implementación: SE Algoritmo: 2

Tercera Olimpiada Mexicana de Informática 1998

Explorador

Descripción del problema.

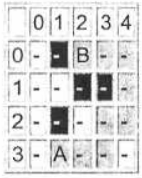
Un explorador tiene que decidir donde construir un camino para llegar de un punto A a un punto B. Para auxiliarse el explorador ha hecho un mapa con los obstáculos que existen. Cuadriculó su mapa y quiere un camino que pase por el menor número de cuadros. El camino sólo puede ir de un cuadro a otro si tienen un lado en común, es decir, no puede avanzar en diagonal, y no puede pasar por un cuadro que contenga un obstáculo. Cada cuadro del mapa se identifica por sus coordenadas, primero la columna y después el renglón. Las columnas están numeradas de izquierda a derecha iniciando con el 0. Los renglones están numerados de arriba hacia abajo iniciando con el 0.

Problema

Escribe un programa que dado un mapa con obstáculos encuentre el menor número de cuadros por los que debe pasar un camino que vaya del punto A al punto B, incluyendo a los cuadros que contienen a A y a B.

Ejemplo

En el siguiente mapa de 5 por 4, el punto A está en el cuadro (1,3) y el punto B en el cuadro (2,0). Cada cuadro negro representa un obstáculo. El número de cuadros por los que pasa un camino que utilice el menor número de cuadros es 9. Un camino de este tipo está sombreado en el mapa:



Entrada

En el primer renglón del archivo INPUT.TXT los enteros N y M, el número de columnas y renglones del mapa, donde 1 <= N <= 50, 1 <= M <= 50. En **cada** uno de los siguientes M renglones hay N números que pueden ser O ó 1, O si no hay obstáculo en el cuadro correspondiente y 1 si lo hay. En el siguiente renglón (el penúltimo) la columna y renglón del punto A. En el último renglón la columna y renglón del punto B.

Salida

En el primer renglón del archivo OUTPUT.TXT **el** número de cuadros por los que pasa un camino mínimo entre A y B.

Ejemplo de entrada y salida

INPUT.TXT	OUTPUT.TXT
5 4 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 1 3 2 0	9

NOTA: todos los casos de prueba tendrán al menos un camino para llegar de A a B.

GRADO DE DIFICULTAD:
Implementación: **SE**
Algoritmo:2

Cuarta Olimpiada Mexicana de Informática 1999
Estado de México del 11 al 15 de mayo 1999

Examen del Concurso Nacional 1999

El examen del Concurso Nacional de la Cuarta Olimpiada Mexicana de Informática consistió de dos días de pruebas para resolver dos problemas cada día en cuatro horas. El Comité Organizador propuso los siguientes cuatro problemas al jurado:

- Primer día: Omicel y Práctica
  Segundo día: Torneo y Atínale

OMICEL

DESCRIPCIÓN DEL PROBLEMA

Una red de telefonía celular esta conformada por dispositivos llamados *células.* Cada célula puede enviar y recibir información a través de 4 distintos canales o frecuencias de radio y tiene un alcance limitado. La compañía OMICEL, tiene una red de células en forma de cuadricula, donde en cada cuadro puede existir una ó ninguna célula.

Cada célula puede comunicarse con las 8 células a su alrededor. Para que dos células adyacentes puedan comunicarse, es necesario que las dos tengan el mismo canal disponible. Cada uno de los 4 canales en una célula puede estar o no disponible (ocupado). Cuando un canal está ocupado toma el valor de una potencia de 2 y cuando está disponible, el de cero.

CANAL	POTENCIA	VALOR
1	2^0	1
2	2^1	2
3	2^2	4
4	2^3	8

El estado de una célula, define cuales canales están ocupados y cuales disponibles. El estado de una célula es la suma de los valores de cada canal, de modo que el estado puede tomar valores entre O y 15. Cuando todos los canales están disponibles su estado será cero y cuando todos estén ocupados será 15. Si el estado es 9, quiere decir que los canales 1 y 4 están ocupados.

Para establecer comunicación entre dos usuarios, uno en el cuadro A y otro en el cuadro B, es necesario que exista una cadena de células que comience en el cuadro A y termine en el cuadro B, todas con el mismo canal disponible. Si los dos usuarios están en el mismo cuadro es necesario que la célula que se encuentra en ese cuadro tenga al menos un canal disponible.
Las causas por las que no se puede establecer comunicación entre dos usuarios son que uno de los usuarios se encuentre fuera del área de servicio (no hay célula dónde uno de los usuarios se encuentra) o que no existe una cadena de células con el mismo canal disponible entre los dos usuarios. Un usuario se identifica por las coordenadas del cuadro donde se encuentra, primero la columna y después el renglón. Las columnas están numeradas de izquierda a derecha iniciando con el 1. Los renglones están numerados de arriba hacia abajo iniciando con el 1.

PROBLEMA

Escribe un programa que dada la red de telefonía, el estado de cada célula de la red y las coordenadas de dos usuarios, determine cual canal debe utilizarse para que la comunicación entre los dos usuarios se establezca usando el menor número posible de células y que en caso de no ser posible la comunicación indique la causa.

ENTRADA

En la primera línea del archivo OMICEL.ENT los enteros *N* y *M,* que son el número de columnas y de renglones de la red, respectivamente, donde 1 <= *N, M* <= 100. En la segunda línea los enteros $\mathbf{X_A}$, $\mathbf{Y_A}$, $\mathbf{X_B}$, Y_B que son las coordenadas del usuario A y B, respectivamente. En cada una de las siguientes M líneas hay *N* números enteros entre -1 y 15. Un -1 indica que en el cuadro correspondiente no hay célula, en otro caso el número indica el estado de la célula de ese cuadro.

SALIDA

En la primera línea del archivo OMICEL.SAL dos números enteros: Si la comunicación se pudo establecer, el primer número es el canal por el cual se estableció y el segundo número es el número de células necesarias para establecer la comunicación. Si la comunicación no se puede establecer, el primer número es un O y el segundo indica la causa: O si alguno de los usuarios está fuera del área de servicio y 1 si todos los canales están ocupados.

EJEMPLO En la siguiente figura se muestra una red de 4 por 3,

1	4	1	10	7
2	1	15	-1	9
3	0	1	13	8

1 2 3 4

Se quieren comunicar los usuarios de (2,1) y (4,3) Se pueden usar los canales 2 y 3. Utilizando el canal 2 se necesitan 5 células, con el canal 3 se necesitan 4. Si el usuario en (3,2) se quisiera comunicar, ño podría por que no hay célula Tampoco podrían comunicarse los usuarios de (2,1) y (2,2) porque no hay canales disponibles

EJEMPLO DE ENTRADA Y SALIDA

OMICEL.ENT	OMICEL.SAL
4 3 2 1 4 3 4 1 10 7 1 15 -1 9 0 1 13 8	3 4

GRADO DE DIFICULTAD:
Implementación: SE Algoritmo: 3

PRACTICA

INTRODUCCIÓN

Este es un problema diferente al de las olimpiadas anteriores. Aquí, los algoritmos que aseguran la óptima solución del problema tardan mucho más de los 20 segundos que se te dan para resolverlo. De modo que no se te calificará que dés la óptima solución, sino que tanto te puedes acercar a ella. Este tipo de problemas se llaman heurísticos.

DESCRIPCIÓN DEL PROBLEMA

Eres el director de una escuela de nivel superior que quiere asignar a sus alumnos en diferentes empresas para realizar sus prácticas profesionales. Existe un número A (1 <= A <= 100) de alumnos y un número E (1 <= E <= 20) de empresas.

Cada empresa ha realizado exámenes a todos los alumnos y los calificó en una escala de O a 100. Cada empresa, a su vez, tiene un límite de alumnos L que puede aceptar (1 <= L <= 25).

También se le ha dado el derecho a cada alumno de nombrar una empresa con la cual no desea trabajar.

Sin embargo, para mejorar el prestigio de la escuela te interesa que todas las empresas estén satisfechas lo más posible.

Para evaluar ésto, sumas las calificaciones de **todos** tus alumnos con respecto a la empresa donde los asignaste. Cuando esta suma es lo mayor posible, el prestigio es mayor.

PROBLEMA:

Debes realizar un programa que dados el número de estudiantes, el número de empresas, la empresa en la que cada alumno no desea trabajar, el límite de alumnos que puede aceptar cada empresa y la calificación que dio cada empresa *a* cada alumno, obtenga la mayor suma posible de calificaciones.

ENTRADA

La primera línea del archivo de entrada PRACTICA.ENT contiene los enteros A y E.

En la segunda línea hay A enteros, entre 1 y E, cada número indica la empresa a la que no desea pertenecer cada alumno, empezando por el primero y en orden.

En la tercera línea hay E números enteros, cada número corresponde al número de alumnos que admite cada empresa, empezando por la primera y en orden.

En cada una de las siguientes E líneas hay A enteros que son las calificaciones de los diferentes alumnos según cada empresa.

SALIDA

Tu programa debe escribir en el archivo PRACTICA.SAL A líneas con un entero cada una, indicando el número de la empresa en que trabajará el alumno.

EJEMPLO

PRACTICA.ENT	PRACTICA.SAL
5 3 1 2 1 2 3 10 5 5 90 70 80 90 60 100 50 70 85 70 85 70 90 80 90	2 1 3 1 2

NOTA: Esta solución puede no ser la óptima.

En este caso hay 5 alumnos y 3 empresas.

El alumno 1 no desea trabajar en la empresa 1, el alumno 2 no desea *trabajar* en la empresa 2, etc.

La empresa 1 acepta 10 alumnos, la 2 acepta 5, etc.

El alumno 1, obtuvo calificación de 90 con la empresa 1, 100 con la empresa 2, etc.

Se puede lograr el resultado mostrado con la siguiente asignación:

alumno	1	2	3	4	5
empresa	2	1	3	1	2
calificación	100	70	90	90	70

Donde la suma de calificaciones es 420.

NOTA: Todos los casos de prueba tienen solución

GRADO DE DIFICULTAD:
Implementadón: CE Algoritmo: 5

ATINALE

INTRODUCCIÓN

Este problema es distinto a los de olimpiadas anteriores en el sentido de que tu programa tendrá que interactuar con funciones de una biblioteca diseñada para este problema, usar una función equivale a hacer una pregunta cuya respuesta te dará información que tendrás que saber utilizar para resolver el problema.

DESCRIPCIÓN DEL PROBLEMA

Estás saliendo en la TV en un programa de concursos. El concurso se trata de encontrar el código oculto. El código oculto está formado con las 10 primeras letras del abecedario (abcdefghij). Te presentan un panel en el que ves las 10 letras en el orden anterior que puede o no ser igual al oculto. El código oculto siempre empieza con "a" por lo que siempre estará en el lugar correcto al inicio del juego. Además te dicen cuántas de las demás letras están en su lugar. Para encontrar el código oculto puedes intercambiar entre sí dos letras del panel y preguntar cuántas letras quedaron en su lugar después del intercambio que hiciste.

Repitiendo varias veces este proceso puedes ir obteniendo información sobre las posiciones correctas de las letras y así encontrar el código oculto. El concurso termina cuando haces 102 preguntas o cuando seas capaz de decir cual es el código oculto, lo que ocurra primero.

PROBLEMA

Escribe un programa que utilice las funciones ***inicia, cambia, adivina*** y ***fin*** de la biblioteca ***juega*** **(juega.h** si utilizas Turbo C/C++ y **juega.tpu** si utilizas Turbo Pascal) para obtener los datos y encuentre en menos de 101 intercambios el número oculto.

La función ***inicia*** devuelve el número de letras que se encuentran en su lugar (contando a la "a").

_a función ***cambia*** tiene como parámetros dos números enteros que son los lugares de las letras que deseas intercambiar y devuelve el número de letras que quedaron en su lugar después del intercambio.

La función *adivina* tiene como parámetro un arreglo de caracteres, de tamaño 10. y devuelve un valor que puede ser 0: si no es el código correcto y 1 si le atinaste.

La función ***fin*** tiene como parámetro un arreglo de caracteres de tamaño 10 y no devuelve ningún valor. Si utilizas esta función no podrás volver a utilizar la librería.

NOTA: Cada vez que utilices cualquiera de estas funciones, te cuenta como una pregunta.
Tienes que utilizar *inicia* al comenzar tu programa y *fin* al terminar, de otro modo tu calificación será cero.

EJEMPLO

Si el código oculto es abijedcfgh, al inciar el concurso, el panel mostrará abcdefghi. Al ejecutar la función ***inicia*** ésta te devuelve el número 3, (coinciden a,b,e).

Si enseguida se ejecuta la función ***cambia*** con parámetros 2 y 3, se obtiene a cambio el número 2, pues después de intercambiar "b" y "c" quedaron en su lugar sólo "a" y "e".

Instrucción	¿Qué pasa?
cuantos=inicia	variable cuantos = 3
corr=cambia(2,3)	variable corr=2
cod="a,b,c,d,e,f,g,h,j,i"	el arreglo de caracteres cod, toma valores.
band=adivina(cod)	variable band=0
cod="a,b,i,j,e,d,c,f,g,h"	nuevos valores de arreglo
band=adivina(cod)	variable band=1
fin(cod)	el evaluador comparará el valor del arreglo cod, con el código oculto.

CALIFICACIÓN

Tu programa será probado 10 veces, por cada una de esas veces que tu programa encuentre el número en menos de 52 intercambios recibirás 10 puntos. Si tu programa no lo logra recibirás O puntos.

NOTA: En la primer línea del archivo ATINALE.ENT se encuentran los 10 dígitos en el orden deí número oculto, separados entre si por un espacio. En la segunda línea se encuentran los 10 dígitos en el orden que muestra el panel, separados entre si por un espacio. En la tercera línea está el dígito que se te indica que está correcto. Durante la aplicación del examen la biblioteca juega, tomará los datos de entrada de este archivo. Pero la biblioteca que usaremos durante la evaluación tendrá los casos de prueba programados, de manera que todos serán evaluados con los mismos casos y nadie podrá saberlos de antemano.

GRADO DE DIFICULTAD:
Implementación: SE Algoritmo: 3

DEFINICIÓN DE LAS FUNCIONES

TURBO C/C++

```
int  inicia{};

int  cambia( int  uno,  int  otro);

void  fin(  char panel [10] ) int

adivina(  char panel [10] )
```

TURBO PASCAL

```
function  inicia():  integer;

function cambia(uno:  integer;  otro:
integer):  integer;

procedure  fin(  panel:char);

function  adivina(  panel:char):
integer;
```

EJEMPLO DE USO

Turbo C/C++

```
#include
"juega.h"

int  panel [10] ,  cuantos,i,j;

void main(void){
  cuantos  =  inicia();

  cuantos = cambia( i,j);
```

Turbo Pascal

```
program atinale;

uses juega;

panel: array [0..9] of integer;
cuantos, i, j: integer;

begin

  cuantos = inicia;

  cuantos = cambia(i,j);

end.
```

TORNEO

DESCRIPCIÓN DEL PROBLEMA

Al principio de un torneo de tenis los participantes son numerados a partir del número 1. Un participante es eliminado la primera vez que pierde un partido.

Cada día se desarrolla una ronda de partidos. En cada ronda cada participante juega un partido, a menos que el número de participantes en esa ronda sea impar, en cuyo caso habrá un participante que no juegue en esa ronda y pasa automáticamente a la siguiente. Así, después de cada ronda es eliminada la mitad de los participantes que juegan. Cada ronda el participante con el número más bajo se enfrenta con el que tiene el número más alto. El jugador con el segundo número más bajo se enfrenta con el que tiene el segundo número más alto, y así sucesivamente. Si el número de participantes en una ronda es impar, el participante que pasa automáticamente a la siguiente ronda es el que sobra después de esta asignación de partidos. Por ejemplo, en una primera ronda con 7 participantes el participante 4 pasa a la siguiente ronda.

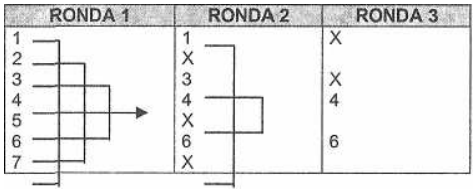
Las rondas también son numeradas a partir de la primera iniciando con el número 1. Antes de que inicie el torneo comienzan las apuestas y te interesa saber si dos participantes tienen posibilidad de llegar a jugar la final y si no tienen esa posibilidad entonces cual es la ronda más alta en la que se pueden llegar a enfrentar. Por ejemplo el primer y el último participante no pueden jugar la final pues en la primera ronda uno de los dos tendrá que ser eliminado. Por tanto la ronda más alta en que pueden enfrentarse es la 1.

PROBLEMA

Escribe un programa que dado el número de participantes en el torneo y los números de dos participantes, determine si los participantes pueden llegar a jugar la final y en caso de que esto no sea posible determine el número de la ronda más alta en la que podrían llegar a enfrentarse.

EJEMPLO

Si al principio del torneo hay 7 participantes, los participantes 4 y 6 pueden llegar a enfrentarse en la final. Mientras que los jugadores 1 y 6 no pueden llegar a la final pues si ganan la primera ronda, no importa los resultados de los demás participantes, tendrán que enfrentarse forzosamente en la ronda 2. En la figura se muestra una forma en la que los participantes 4 y 6 pueden jugar la final, en cada ronda se muestran los números de los participantes que compiten y una X indica que ese participante fue eliminado en la ronda anterior.



ENTRADA

En la primera línea del archivo TORNEO.ENT hay 3 números enteros *P, A* y *B. P* es la cantidad de participantes en el torneo, con *2<= P<=* 1000000. *A* y 6 son los números que le tocaron a dos participantes antes de iniciar el torneo, con 1 *<= A != B<=P*.

SALIDA

En la primera línea del archivo TORNEO.SAL dos números enteros. El primero de ellos es el número de rondas en el torneo y el segundo es el número de la ronda más alta en la que podrían llegar a enfrentarse los jugadores que llevaban los números A y B antes de iniciar el torneo.

TORNEO.ENT	TORNEO.SAL
7 4 6	3 3

EJEMPLO DE ENTRADA Y SALIDA

GRADO DE DIFICULTAD:
Implementación: S Algoritmo; 3

QUINTA OLIMPIADA MEXICANA DE INFORMÁTICA 2000

Examen del Concurso Nacional 2000

El examen del Concurso Nacional de la Quinta Olimpiada Mexicana de Informática consistió de dos días de pruebas para resolver dos problemas cada día en cuatro horas. El Comité Organizador propuso los siguientes cuatro problemas al jurado.

En esta sección también se presentan los cuatro problemas que fueron aplicados a la preselección, lo que permitió seleccionar a los finalistas del evento del 2000.

Primer día: Moscas y Maratón de Cine
Segundo día: Sumas y Vendedor de Alfombras

Preselección Cajas y Cajas y más cajas
Preselección Escalera y Misión Imposible

MOSCAS

Para tu clase de biología te encuentras realizando un estudio con una cierta raza de moscas. El objetivo de tu experimento es encontrar el modelo que rige el crecimiento demográfico de esta raza de moscas.

Para hacer un modelo de la población de tu grupo de moscas, identificas cada mosca con dos datos: el momento en que nace la mosca y el momento en que muere. Todas las moscas viven por lo menos una unidad de tiempo.

PROBLEMA

Para un grupo de 1 N 10,000 moscas, de las cuales tienes apuntados el momento de su nacimiento y el momento de su muerte como dos números enteros 1 *ni, mj* 60,000, escribe un programa que encuentre cual es la población máxima del grupo de moscas y que escriba el rango de tiempo en que se puede encontrar esta población.

ENTRADA

Tu programa deberá leer del archivo de texto **MOSCAS.ENT** los siguientes datos: en la primera línea el número *N* indicando la cantidad total de moscas en el experimento, en las siguientes W líneas una pareja de números enteros separados por un espacio, indicando el momento del nacimiento y el momento de la muerte de la mosca respectiva.

SALIDA

Tu programa deberá escribir en el archivo de texto **MOSCAS.SAL** los siguientes datos: en la primera línea el número *P* indicando cual fue la máxima población alcanzada para el experimento y en la siguiente línea los rangos de tiempo en los que esta población se alcanzo. Los rangos de tiempo en los que se alcanzó la población máxima deberán ir escritos en una línea como *inicio del rango fin del rango inicio del rango fin del rango.....*

MOSCAS.ENT	MOSCAS.SAL
5 1 10 12 18 20 30 5 12 25 33	2 5 10 25 30

EJEMPLO

GRADO DE DIFICULTAD:
Implementación: S Algoritmo: 2

MARATÓN DE CINE

Tu gran cantidad de deberes estudiantiles ha impedido que en las últimas semanas hagas tus frecuentes visitas a las salas de cine. Pero por suerte este fin de semana los maestros de tu escuela han decidido darte un respiro, y por tu parte has tomado la decisión de gastar cada minuto disponible en satisfacer tu sed de películas.

Por suerte para ti la cantidad de películas a tu disposición *es* muy grande, ya que en tu ciudad existen 1 C 10, complejos cinematográficos con 1 S 10 salas de cines cada uno. Cada uno de estos complejos tiene funciones las 24 horas del día y cada sala expone siempre la misma película durante todo el día. Cada sala tiene a lo más 10 funciones diarias.

Ya que tu tiempo es limitado, te interesa ver únicamente las mejores películas, por lo que basándote en críticas y en comentarios que has escuchado haces una tabla de las películas disponibles, asignándole a cada una de ellas una puntuación de acuerdo a que tan recomendable esta.

PROBLEMA

Conociendo la puntuación de cada una de las películas disponibles, el tiempo que tardas en transportarte de un complejo cinematográfico a otro, los horarios de las funciones y el tiempo que tienes disponible para dedicar a tu maratón de cine, haz un programa que elabore un itinerario con el cual la suma de puntos de las películas que viste sea el máximo posible, cada película sólo se puede ver una vez.

ENTRADA

Tu programa deberá leer del archivo de texto **CINE.ENT** los siguientes datos: En la primera línea los números C y S indicando cada uno el número de complejos y el número de salas por complejo, en las siguientes *C* líneas los tiempos en minutos que tardas en transportarte de un complejo cinematográfico a otro, cada una de estas líneas contendrá C números enteros separados por un espacio que indique el tiempo entre el complejo i (número de línea) y el complejo *j* (número de columna). En las siguientes C * S líneas estará el calendario de funciones para cada una de las salas de cada complejo. Las primeras S líneas corresponden a las salas del complejo 1, las siguientes S a las salas del complejo 2 y así subsecuentemente.

Cada una de éstas C * S líneas tendrá los siguientes datos: un número entero 1<= *P* <= 1000 indicando la puntuación de la película en la respectiva sala, un número entero 60 <= *D* <= 300 indicando la duración en minutos de la película en la respectiva sala y una lista de números terminada por un -1 indicando la hora como un minuto del día en que comienza cada una de las funciones.

En la última línea del archivo estarán dos números enteros separados por un espacio que indican la hora en que piensas iniciar tu maratón (el cual inicia siempre en el complejo 1) y la cantidad de minutos que piensas dedicar al mismo.

SALIDA

Tu programa deberá escribir en el archivo de texto **CINE.SAL** en la primera línea la puntuación máxima que se puede alcanzar. En las siguientes líneas debes de dar el número de sala, el número de complejo y el horario en que vas a ver cada película en tu itinerario maratónico.

EJEMPLO

CINE.ENT	CINE.SAL
2 3 0 40 40 0 485 128 60 240 420 600 800 1100-1 398 180 600 820 1140 1360-1 900 199 100 300 500 700 900 1100 1300-1 876 150 300 550 800 1050 1300-1 100 200 200 500 800 1100 1400-1 750 120 100 400 700 1000 1300-1 360 1000	3011 3 2 400 1 2 550 1 1 800 3 1 1100

GRADO DE DIFICULTAD:
Implementación: **SL** ***Algoritmo: 3***

SUMAS 2000

PROBLEMA

Dado un número entero 1 N 100,000,000 y un lista de *M* números enteros 3 <= M_i <= 100,000,000. Encuentra un conjunto de 3 números de la lista, tales que su suma sea igual al número N.

ENTRADA

Tu programa deberá leer del archivo de texto **SUMAS.ENT** los siguientes datos: en la primera línea el número *N,* en la segunda línea el número *M* y en las siguientes *M* lineas los números de la lista.

SALIDA

Tu programa deberá escribir en el archivo de texto **SUMAS.SAL** tres números de la lista dada tales que su suma sea igual al número N. Los tres números deberán estar escritos en la misma línea, separados cada uno por un espacio.

NOTA: En el caso de que exista más de una solución, tu programa solo deberá escribir una de ellas.

EJEMPLO

SUMAS.ENT	SUMAS.SAL
25 10 15 8 1 4 18 3 32 12 4 10	4 18 3

GRADO DE DIFICULTAD:
Implementación: S Algoritmo: 1

VENDEDOR DE ALFOMBRAS

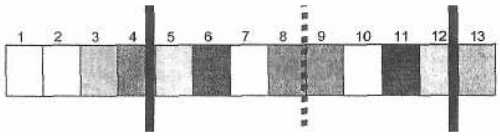
Durante años te has dedicado al negocio de la distribución de alfombras y tapetes. En éste tiempo has notado que la gente prefiere con mucho los diseños de alfombras que son simétricos sobre los que son asimétricos.

Para mejorar tu margen de ganancia siempre has comprado rollos completos de alfombra y los cortas según las necesidades de tus clientes. Un rollo de alfombra consta de una cantidad 1 <=*M* <= 20000 de franjas de tela de 1cm de ancho, cada franja es un diseño diferente, el número máximo de diseños que pueden existir en un rollo de alfombra es de 256.

PROBLEMA

Como las alfombras simétricas te ofrecen un mejor margen de ganancia deberás realizar un programa que conociendo el patrón de diseños de un cierto rollo busque la alfombra simétrica más grande que puedas cortar.

Por ejemplo para el rollo de alfombra de la figura, el corte simétrico más grande que puedes cortar es de tamaño 8 y se obtiene haciendo cortes en la franja 4 y 12.



ENTRADA

En el archivo de entrada **ALFOMB.ENT** encontrarás en la primera línea el número 1 <= *M* <= 20000 de centímetros de alfombra en el rollo, en la siguiente línea encontrarás una lista de M números que indican los diseños que están separados por espacios y se encuentran en el rango de O a 255 incluyéndolos.

SALIDA

En el archivo de texto **ALFOMB.SAL** deberás escribir tres números *T, C1 y C2* indicando cada uno: el tamaño máximo de alfombra simétrica que se puede cortar, y el lugar donde se deben hacer cada uno de los cortes. Los números deberán estar en la misma línea y separados por un espacio.

EJEMPLO

ALFOMB.ENT	ALFOMB.SAL
13 0 0 1 2 3 4 0 5 5 0 4 3 1	8 4 12

GRADO DE DIFICULTAD:
Implementación: C Algoritmo: 3..

CAJAS 2000

INTRODUCCIÓN

Eres el encargado de empaquetar toda una serie de productos en cajas. Para esto puedes requerir una o más cajas. Por normas de la empresa, se requiere que cada serie se empaquete con un sólo tamaño de caja. Sin embargo los productos no son todos iguales, pueden tener diferentes tamaños entre ellos. Al empaquetar los productos, el orden que tenían al llegar debe mantenerse. Se desea también que el tamaño de caja a utilizar sea el más pequeño posible y que no se desperdicie espacio, cada una de las cajas debe quedar llena.

DESCRIPCIÓN DEL PROBLEMA

Dada una lista con el tamaño de los productos en el orden que debes empaquetarlos, debes determinar el número de cajas que utilizarás y el tamaño del que serán de tal forma que se cumpla con los requerimientos anteriores.

ENTRADA

Tu programa deberá leer del archivo INPUT.TXT, en la primera línea el número N de productos (1<= N <= 100,000).
En las siguientes N líneas el tamaño t de cada producto (1<= t<=20)

Nota: El volumen total que ocupa toda la serie de productos es menor de 1,000,000.

SALIDA

Tu programa deberá escribir en el archivo OUTPUT.TXT el número de cajas a utilizar y el tamaño de caja que utilizarás, en una sola línea.

EJEMPLO

INPUT.TXT
```
13
1
3
2
3
1
2
2
1
1
2
2
3
1
```

OUTPUT.TXT
```
4 6
```


GRADO DE DIFICULTAD:
Implementación: SE Algoritmo: 4

Cajas y más Cajas

INTRODUCCIÓN

En una tienda de cajas organizan un concurso entre sus clientes. El concurso consta en lo siguiente: Se cuenta con varias cajas de diferentes tamaños. El ganador del concurso será quien, metiendo unas cajas en otras, logre meter el mayor número de cajas en una sola pila. Para poder meter una caja en otra, su tamaño debe de ser menor.

Así, una caja puede contener dentro de ella varias cajas, la tercera caja deberá ser más pequeña que la segunda, la cuarta más chica que la tercera, etc.

Las cajas son de forma cilindrica, por lo que tienen cierto radio y cierta altura. Para que una caja pueda entrar en otra, debe ser más pequeña tanto en el radio como en la altura.

DESCRIPCIÓN DEL PROBLEMA

Deberás realizar un programa que dada la lista de los tamaños de las cajas, dé el número máximo de cajas que se pueden juntar en una pila.

ENTRADA

Tu programa deberá leer del archivo de entrada INPUT.TXT , de la primera línea, el número N de cajas en la línea (1<= N <= 500). En las siguientes N líneas las dimensiones de cada una de las cajas, el radio R y la altura H, respectivamente (1<= N,R <=1000).

SALIDA

Deberás escribir en el archivo de salida OUTPUT.TXT, en la primera línea el número C máximo de cajas, apilables en una sola.

EJEMPLO

INPUT.TXT

```
6
4 9
10 4
5 5
1 1
3 8
2 3
```

OUTPUT.TXT

```
5
```


GRADO DE DIFICULTAD:
Implementación: SE Algoritmo:..4

Misión Imposible

Eres un agente especial, experto en cuestiones de tecnología. Te han mandado llamar debido a la existencia de una situación poco común.

Un criminal, reconocido por su inteligencia, ha colocado una red de bombas alrededor de un centro turístico muy importante. Sin embargo, a este maleante le gusta creer que no hay nadie más inteligente que él y para demostrarlo, le ha dado la oportunidad al gobierno de desactivar las bombas si logran descifrar cómo.

Cada una de las bombas tiene un contador que indica cuanto falta para que estalle esa bomba. El contador de cada bomba, decrece a cierta velocidad, por ejemplo, para una bomba el contador puede disminuir 2 unidades por minuto, para otra 5, etc.

A partir del aviso del criminal, todos los contadores comienzan a disminuir a su velocidad correspondiente. Tú puedes detener el contador de cualquier bomba mediante un botón y cada vez que haces esto, te lleva un minuto (esto es, no puedes detener dos contadores al mismo tiempo). Si así lo deseas puedes dejar pasar tiempo sin detener ninguna bomba.

La red de bombas esta configurada de tal manera que si al final, los contadores de todas las bombas marcan el mismo valor, no explotarán. Pero con una sola que quede diferente, tú y todos los turistas, serán historia.

SI cualquier bomba llega al valor de cero, o se pasa (negativo) todas las bombas explotarán inmediatamente. Sin embargo, el criminal asegura que su acertijo tiene solución.

Al gobierno, obviamente le interesa que la desactivación de las bombas se lleve a cabo en el menor tiempo posible, o sea que el contador final de las bombas sea el mayor.

Puedes comenzar a detener la primera bomba desde el instante en que comienza el juego. O sea que puedes congelar el primer contador en su valor inicial.

PROBLEMA

Tu misión, si decides aceptarla es:

Realizar un programa que dado el número de bombas, el número inicial de cada contador y la velocidad a la cual disminuyen, encuentre el contador final que marcarán las bombas, si logras desactivarlas.

ENTRADA

Tu programa deberá leer del archivo INPUT.TXT, en la primera línea, el número N de bombas (2 <= N <= 250), en las siguientes N líneas el valor inicial del contador C (1 <= C <= 50,000) y la velocidad V (1<= V <=100) respectivas.

SALIDA

Tu programa deberá escribir al archivo OUTPUT.TXT el valor final que deberán marcar todos los contadores.

EJEMPLO

INPUT.TXT
```
3
10 2
8 3
9 1
```

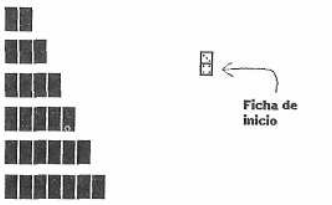
OUTPUT.TXT
```
2
```


GRADO DE DIFICULTAD: Implementación: SE Algoritmo: 3

ESCALERA

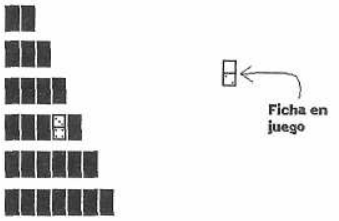
Con un paquete de fichas de dominó se pueden organizar una infinidad de juegos. En esta ocasión E.O.F. Dijkstra (la mascota de la Olimpiada) se divierte con el clásico solitario de la "Escalera".

El juego de la "Escalera" trata de lo siguiente: se forman 6 hileras con las fichas de dominó volteadas hacia abajo, cada hilera tiene un número diferente de fichas empezando con 2 la primera, 3 la segunda, etc., hasta la sexta fila con 7 fichas; en este momento nos queda una ficha, la cual volteamos y con ésta empezamos el juego.



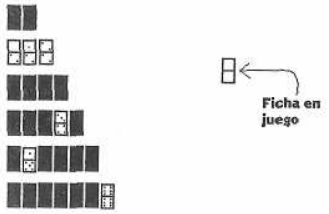
Esta ficha que tenemos en la mano debe ser colocada en su lugar; para ello buscamos la fila correspondiente al número mayor de esta ficha (en nuestro ejemplo la ficha es 4-3, así que tomamos la fila 4) y después la casilla correspondiente al número menor, en este caso 3, contando la columna de la izquierda como 0. Tomamos la ficha que se encontraba en este lugar, y la volteamos después de haber colocado la ficha anterior cara arriba en su lugar correspondiente.

Este proceso se repite con todas las fichas que vayamos volteando hasta encontrar la ficha "0-0", la cual no tiene un lugar asignado; en el momento que voltees esta ficha el juego termina.



A continuación se realiza el conteo de los puntos obtenidos: cada ficha volteada cara arriba agrega su valor a la puntuación final (así la ficha "3-3" vale 6 puntos). Además por cada fila volteada totalmente se añade el número de la fila a la puntuación.

En este caso la puntuación final es de 36 puntos (34 por las fichas y 2 por la fila completa).



Como E.O.F. no tiene manos, ha decidido crear un programa de computadora que le permita calcular las puntuaciones de este juego. Tú debes ayudarlo a lograr este objetivo.

Nota: Las fichas que ocupan su lugar pero no son volteadas cara arriba no se agregan a la puntuación final.

Archivo de entrada ("Input.txt")

En la primera línea se encuentra la ficha con la que empiezas en la mano con formato n-m, donde n y m son números entero entre O y 6.

Las siguientes 6 líneas contienen cada una ficha más que la anterior desplegadas con el mismo formato n-m y separadas con un espacio. Estas líneas representan las fichas que se encuentran volteadas para abajo y se encuentran en su hilera correspondiente en el orden que aparecen.

Nota: Todos los archivos de entrada son válidos, es decir, no hay fichas repetidas. Además "6-3" y "3-6" son la misma ficha.

Archivo de salida("Output.txt")
Este consiste de un número que indica la puntuación lograda después de *realizar* el iueqo.

Ejemplo:

Input.txt	*Output.txt*
4-3 1-3 3-2 6-6 0-0 1-5 3-0 0-1 4-2 3-6 1-1 1-4 3-3 2-0 5-2 0-4 2-1 6-0 5-3 6-5 4-6 4-5 6-2 4-4 6-1 5-0 5-5 2-2	36

GRADO *DE DIFICULTAD:*
Implementación: SE Algoritmo: 2

SEXTA OLIMPIADA MEXICANA DE INFORMÁTICA 2001
CULIACÁN, SINALOA DEL 6 AL 10 DE MAYO DEL 2001

Examen del Concurso Nacional 2001

El examen del Concurso Nacional de la Cuarta Olimpiada Mexicana de Informática consistió de dos días de pruebas para resolver dos problemas cada día en cuatro horas. El Comité Organizador propuso los siguientes cinco problemas al jurado de los cuales se usaron cuatro:

- TAREAS
- COMISOLO
- DEFENDER
- ELEVADOR
- TAPETES

TAREAS

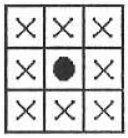
DESCRIPCIÓN

Existe un juego de mesa para 2 jugadores llamado áreas, cuyo objetivo es ir colocando fichas en un tablero rectangular cuadriculado, de modo que eí número de fichas contiguas de un solo color abarque la mayor área posible.

Cada jugador tiene su propio color de fichas, y en su turno puede colocar una ficha de su color en cualquier lugar de la cuadrícula que no se encuentre ocupado.

Una ficha se considera contigua a otra si los cuadros en los que se encuentran tienen al menos un punto en común, es decir, cada casilla (excepto las de la orilla) tienen 8 casillas contiguas.

Un juego termina cuando el tablero se encuentra lleno.



Posiciones contiguas indicadas por 'X'

En tu salón de clases están organizando un torneo de este juego, y te han pedido que desarrolles un programa que contabilice las áreas para evitar problemas entre los jugadores en el momento de la cuenta.

PROBLEMA

Escribe un programa que reciba como entrada la configuración final del tablero de juego y haga un recuento de las áreas de cada color para obtener al ganador de la partida.

ENTRADA

Tu programa deberá leer del archivo de texto INPUT.TXT la configuración final del tablero de juego. En la primera línea del archivo INPUT.TXT hay dos números enteros 1 <= *N, M* <= 100, indicando respectivamente el alio y el ancho del tablero de juego que se uso. En las siguientes N líneas hay *M* números enteros separados por un espacio que indican el color por el que esta ocupado cada casilla. Los valores de las casillas pueden ser O - primer jugador, ó 1 - segundo jugador.

SALIDA

Tu programa deberá escribir en el archivo de salida OUTPUT.TXT dos números enteros separados por un espacio. El primero de ellos debe ser un O ó 1 indicando al jugador ganador, y el segundo debe indicar el tamaño del área más grande que se encontró en el tablero.

EJEMPLO DE ENTRADA Y SALIDA

Ejemplo 1

INPUT.TXT	OUTPUT.TXT
3 3 0 1 0 1 0 1 0 1 0	0 5

Ejemplo 2

INPUT.TXT	OUTPUT.TXT
5 6 0 1 1 1 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0	0 11

GRADO DE DIFICULTAD:
Implementación: SE Algoritmo: 2

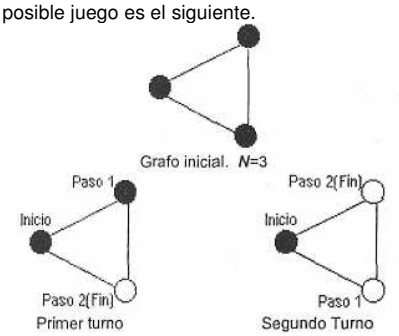
COMISÓLO

DESCRIPCIÓN DEL PROBLEMA

El cOMIsolo es un juego derivado del comesolo en el que el objetivo es retirar todas las piezas del juego salvo una, siguiendo reglas predefinidas para retirar las piezas.

A diferencia de la versión original el cOMIsolo no se juega sobre un tablero, sino sobre una estructura de datos, mas específicamente un grafo. La estructura del juego es la siguiente: el juego se inicia con un grafo de N nodos los cuales contienen una ficha cada uno. En cada turno, puedes posicionarte en cualquier nodo del grafo, siempre y cuando este todavía tenga su ficha, y a partir de ese nodo debes avanzar dos nodos utilizando las aristas del grafo, una vez hecho esto debes retirar la ficha del nodo en donde terminaste. Durante el recorrido puedes pasar por nodos que no tengan ficha, la única restricción es que el nodo inicial cuente con ficha. El juego se termina, cuando todas las fichas, salvo una, han sido retiradas del juego.

Un posible juego es el siguiente.



PROBLEMA

Escribe un programa que dado el grafo inicial indique el orden en el que se deben llevar a cabo las jugadas para que todas las fichas puedan ser retiradas.

ENTRADA

Tu programa deberá leer del archivo de entrada INPUT.TXT los datos del grafo que servirá de tablero. En la primera línea del archivo INPUT.TXT esta el número 1 <= *N* <= 100 de nodos en el grafo. En las siguientes *N* líneas se encuentran N números que pueden ser 1 ó O, indicando un 1 que existe una arista entre el nodo i y el nodo *j* donde i es la fila y *j* es la posición en el renglón, y O indicando que no hay arista.

NOTA: Las aristas no son bidireccionales, es decir, el hecho de que exista una arista que va de i a *j* no implica que se pueda ir también de y a i.

NOTA2: Ningún nodo tiene conexión con el mismo. Es decir, una vez posicionado en un nodo, no puedes avanzar y terminar en el mismo nodo.

SALIDA

Tu programa deberá escribir en el archivo de salida OUTPUT.TXT las jugadas que se deben realizar para terminar el juego. Cada turno deberá estar escrito en una línea con el siguiente formato:
nodo_inicial nodo_final

EJEMPLO DE ENTRADA Y SALIDA

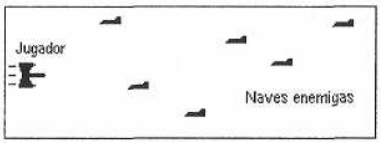
INPUT.TXT	OUTPUT.TXT
3 0 1 1 1 0 1 1 1 0	1 3 1 2

GRADO DE DIFICULTAD:
Implementación: CE Algoritmo: 2

DEFENDER

DESCRIPCIÓN

Hace ya algunos años, hubo un juego de atan llamado Defender. El juego trataba de una nave espacial cuyo objetivo era destruir a todas las naves enemigas. Para llevar a cabo esto, la nave del jugador recorría un mapa en forma de pasillo en el que iban apareciendo las naves enemigas. Una vez que se llegaba al final del mapa se comenzaba nuevamente desde el principio y así hasta destruir a todos los enemigos.



Pantalla del juego

Para destruir una nave, era necesario colocarse a la misma altura y disparar. Si una nave enemiga pasaba sin que fuera destruida era necesario dar toda la vuelta para eliminarla.

PROBLEMA

Eres el piloto de la nave DEFENDER y tu tarea, como ya se explicó es detener a tus enemigos, desgraciadamente, debido a la batalla tus motores ya no funcionan correctamente, y tus armas han perdido algo de alcance. Debido al corto alcance de tus armas solo te es posible dispararle a la nave enemiga que se encuentra más cercana a ti.

Las naves enemigas vienen con diferentes alturas entre 1 y 255. Debido a que sufriste un daño en uno de tus propulsores tu nave tarda mucho tiempo en subir, por lo que en el rango de tiempo que hay entre una nave enemiga y la otra, puedes bajar todo lo que quieras, pero sin embargo solo puedes subir S unidades de altura.

Debido a que te resultara imposible detener a todos tus enemigos, debes escribir un programa que te permita decidir a cuales debes destruir de modo que el número de naves enemigas derribadas sea el máximo posible.

La altura con la que inicia tu nave siempre es de 255.

EJEMPLO

En la pantalla que se muestra hay 6 naves enemigas, suponiendo que sus respectivas alturas fueran 230, 100, 50, 210, 190 y 220. Y que S = 30, el número máximo de naves enemigas que se pueden derribar es 4.

Una posible forma de hacerlo sería derribar la primera nave en 230, después bajar a 100 para derribar a la segunda, utilizar el tercer intervalo para subir a 130, usar el cuarto intervalo para subir a 160, derribar la quinta nave en 190, y derribar la sexta nave en 220 para un total de 4 naves derribadas.

ENTRADA

En la primera línea del archivo INPUT.TXT contiene un número entero indicando el valor de 1 <= S <= 50. En la segunda línea hay un entero que indica el número 1 <= *N* <= 200 de naves enemigas aproximándose, en las siguientes N hay un número entero en cada una indicando la altura respectiva de cada una de las naves enemigas.

SALIDA

Tu programa deberá escribir en el archivo de salida OUTPUT.TXT un número entero indicando el número máximo de naves enemigas que pueden ser derribadas.

EJEMPLO DE ENTRADA Y SALIDA

INPUT.TXT	OUTPUT.TXT
30 6 230 100 50 210 190 220	4

GRADO DE DIFICULTAD:
Implementación: SE
Algoritmo:.4

ELEVADOR

DESCRIPCIÓN DEL PROBLEMA

El COMI, se acaba de enfrentar *a* un nuevo problema, tras el desayuno de los líderes, el día de ayer en el Club Sinaloa, se ha presentado el problema de bajarlos a todos de nuevo para poder llevarlos al Centro de Ciencias. El problema nace del hecho de que solamente hay un elevador cuya capacidad de carga es mucho menor que el peso total de los líderes.

Debido a que existe la necesidad de mantenerse de acuerdo con los tiempos planeados, es necesario bajarlos a todos de la manera más rápida posible, por lo que el COMI necesita que escribas un programa, que nos indique cual es esa forma.

PROBLEMA

Escribe un programa, que dada la capacidad de carga de un elevador, así como un conjunto de líderes con sus respectivos pesos, determine cual es el numero mínimo de viajes necesarios del elevador para que todos los líderes sean transportados de nuevo a la planta baja.

NOTA: Los líderes son muy flojos, y mas después de acabar de comer, por lo que ninguno de ellos acepta bajar por las escaleras.

ENTRADA

Tu programa deberá leer del archivo de entrada INPUT.TXT los siguientes dato, en la primera línea del archivo se encuentra el numero 300 <= E <= 1000, que indica la capacidad de carga en kilos que soporta el elevador, en la segunda línea del archivo esta el número 1 <= N <= 1000 que indica el número de líderes que hay, y en la tercera línea hay una lista de *N* números enteros ordenados, cuyos valores están en el rango de 50 a 150, que indican el peso respectivo de cada líder en cuestión.

SALIDA

Tu programa deberá escribir en el archivo de texto OUTPUT.TXT un único número entero indicando el numero mínimo de viajes de elevador que se tienen que *realizar* para bajar a todos los líderes del Club Sinaloa a la planta baja.

EJEMPLO DE ENTRADA SALIDA

INPUT.TXT	OUTPUT.TXT
300 5 130 90 87 78 68	2

GRADO DE DIFICULTAD:
Implementación: CE Algoritmo:..5

TAPETES

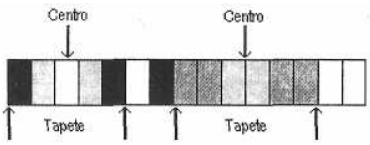
DESCRIPCIÓN

Después de tu éxito obtenido como vendedor de alfombras, has ampliado tu gama de productos y ahora también te dedicas a vender tapetes.

Los tapetes tienen más valor cuando son simétricos, por lo que en tu tienda únicamente vendes tapetes simétricos.

Las telas que usas para tus tapetes están divididas en franjas y cada franja tiene un color que puede ser cualquiera de los 256 colores que hay disponibles. En base a tu experiencia has asignado a cada color un valor entre O y 255 dependiendo de cuanto le gusta a la gente. Para calcular el precio de un tapete basta con sumar los valores de los colores que lo forman.

El centro de simetría de un tapete puede estar, entre dos franjas o en medio de alguna de ellas. Por ejemplo, los siguientes son ejemplos de tapetes simétricos válidos.



NOTA: Un tapete simétrico debe constar al menos de dos franjas. No puede haber un tapete de una sola franja.

PROBLEMA

Escribe un programa que dado un rollo de tela de hasta 20,000 franjas, calcule el precio máximo posible que se puede obtener de la suma de los tapetes simétricos que se obtengan de ella.

NOTA: Un rollo de tela puede ser cortado en tapetes utilizando muchas configuraciones diferentes. Tu programa deberá obtener la configuración cuya suma del precio de los tapetes sea la máxima posible. Por ejemplo, para la tela en la figura de arriba, una posible configuración sería

```
TELA : 1 2 3 2 1 3 1 4 4 2 2 4 4 3 3
Tapete 1 :  1 2 3 2 1
Tapete 2 :  4 4 2 2 4 4
Tapete 3 :  3 3
Precio total de los tapetes: 35
```

O, podría cortarse utilizando la configuración

```
TELA : 1 2 3 2 1 3 1 4 4 2 2 4 4 3 3
Tapete 1 :  2 3 2
Tapete 2 :  1 3 1
Tapete 3 :  4 4 2 2 4 4
Tapete 4 :  3 3
Precio total de los tapetes: 38
```

la siguiente:

ENTRADA

Tu programa deberá leer del archivo de texto 1NPUT.TXT la configuración de la tela que vas a usar. En la primera línea hay un número entero 1 <= N <= 20,000 que indica el largo de la tela. En la siguiente línea hay *N* números enteros separados por un espacio que indican el color de la tela en la posición respectiva. Los valores de los colores van del O al 255.

SALIDA

Tu programa deberá escribir en el archivo OUTPUT.TXT un único número entero indicando el mayor precio que se pueda obtener de cortar la tela en tapetes simétricos.

EJEMPLO DE ENTRADA Y SALIDA

INPUT.TXT	OUTPUT.TXT
15 1 2 3 2 1 3 1 4 4 2 2 4 4 3 3	38

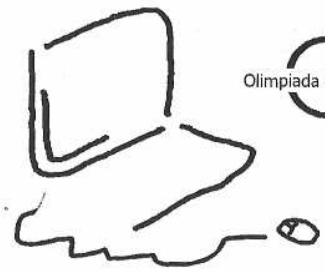
GRADO DE DIFICULTAD:
Implementación: S Algoritmo: 1

SÉPTIMA OLIMPIADA MEXICANA DE INFORMÁTICA 2002
XALAPA, VERACRUZ DEL 13 AL 17 DE MAYO DEL 2002

Examen del Concurso Nacional 2002

El examen del Concurso Nacional de la S Olimpiada Mexicana de Informática consistió de dos días de pruebas para resolver dos problemas cada día en cuatro horas. El Comité Organizador propuso los siguientes cuatro problemas al jurado:

DESCIFRANDO JEROGLÍFICOS
EN CAMINO HACÍA EL TESORO
ABRIENDO LA PUERTA EL
TESORO



Olimpiada Mexicana de Informática
7° Concurso Nacional
Xalapa, Veracruz. 13 a 17 de mayo 2002

DESCIFRANDO JEROGLÍFICOS

DESCRIPCIÓN

La situación económica mundial se encuentra en crisis, lo que te ha llevado a buscar nuevas formas de obtener ingresos económicos. En vista del éxito obtenido por personajes como Alan Quartermain, Indiana Jones, Lara Croft y algunos otros, has decidido que la mejor forma de asegurar tu futuro es la búsqueda de tesoros de culturas antiguas. Investigando sobre las viejas civilizaciones encontraste información sobre los OMIncas, esta cultura aparece mencionada en algunos códices como un pueblo de grandes riquezas y sin embargo su civilización prácticamente no ha sido explorada, lo que la hace el objetivo perfecto de tus planes.

El primer paso para buscar un tesoro es conocer la ubicación geográfica en donde se encuentra, durante tus pesquizas encontraste un mapa que indica la localización de la tumba de su más grande patriarca, desgraciadamente las instrucciones se encuentran escritas en jeroglíficos, por lo que primero tendrás que descifrarlas.

El mapa indica un punto de partida y a partir de este da instrucciones sobre como llegar a la entrada de la tumba. Después de estudiar ampliamente el mapa has descubierto que hay 14 jeroglíficos diferentes, 10 de ellos simbolizan los diez dígitos de la numeración decimal (0,1,2,3,4,5,6,7,8,9), otro mas significa girar a la derecha 90 grados, hay otro que indica girar a la izquierda 90 grados, uno mas simboliza *avanzar* y el último significa fin del mensaje. Las instrucciones del mapa son del estilo de *"avanza* 20 pasos, gira a la derecha, avanza 10 pasos ...", una característica importante es que después de un símbolo de avanzar siempre siguen dos jeroglíficos de número que indican una cantidad de dos cifras que equivale al número de pasos que hay que *avanzar* (por ejemplo, si hay que avanzar 5 pasos seguirían las cifras "05", si hubiera que *avanzar* 15 seguirían las cifras "15"). Los jeroglíficos de número irán **únicamente** después del símbolo de avanza. Para encontrar las coordenadas de entrada a la tumba debes posicionarte en el punto de partida y seguir las instrucciones.

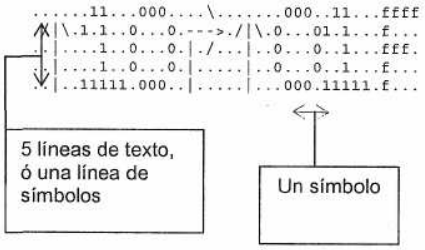
El jeroglífico "fin del mensaje" solamente aparece una vez en el mapa al final de las instrucciones, siempre para indicar que se ha llegado al final de ellas.

PROBLEMA

Debes escribir un programa que conociendo los símbolos de los 14 jeroglíficos diferentes, y el mensaje del mapa, siga las instrucciones y determine el punto exacto de la entrada de la tumba.

Los jeroglíficos están representados siempre en una cuadrícula de 5 x 5 caracteres. Por ejemplo, *los* siguientes serían jeroglíficos válidos.

El mapa contiene un mensaje con instrucciones para llegar a la entrada de la tumba, este mensaje puede contener desde 2 hasta 40 símbolos, seguidos uno del otro, todos en una sola línea de símbolos o 5 líneas de texto. Por ejemplo, suponiendo que las instrucciones fueran: "avanza 10 pasos, gira a la derecha, *avanza* 1 paso, fin del mensaje", utilizando los símbolos de arriba, el



mensaje del mapa se vería así.

Para este problema se considerará que la posición inicial siempre es la coordenada (0,0) y la dirección inicial siempre es el norte, considerando la figura

GRADO* DE *DIFICULTAD:
Implementación: CE Algoritmo:. 1

ENTRADA

Tu programa deberá leer del archivo de entrada DESCIFRA.IN los siguientes datos: los 14 jeroglíficos, cada uno en 5 líneas de texto, con 5 caracteres cada una, en el siguiente orden (O, 1, 2, 3, 4, 5, 6, *7,* 8, 9, Avanza, Gira derecha, Gira Izquierda, Fin del mensaje), los caracteres de cada línea no tienen ningún espacio entre ellos. Después de los 14 jeroglíficos vendrán 5 líneas que pueden tener desde 10 hasta 200 caracteres cada una conteniendo las instrucciones del mapa.

SALIDA

Tu programa deberá escribir al archivo de salida DESCIFRA.OUT dos números, escritos en una sola línea y separados por un espacio, los cuales indiquen la coordenada (horizontal, vertical) en la que se encuentra la entrada de la tumba.

EJEMPLO:

NOTA: Los jeroglíficos de este ejemplo y los caracteres que los representan son sólo para éste caso, los jeroglíficos de los casos de evaluación utilizarán caracteres ASCII y representaciones

Por ejemplo:

Asfui
Desfe
Ikkels
U93'3
0*o92

es un jeroglífico válido y puede representar cualquiera de los 14 símbolos.

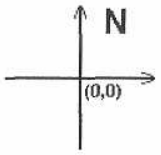
diferentes.

DESCIFRA.IN

```
.000.
0...0
0...0
0...0
.000.
.11..
1.1..
..1..
..1..
11111
.222.
2...2
..22.
22...
22222
.333.
3...3
..33.
3...3
.333.
4..4.
4..4.
44444
...4.
...4.
55555
5....
5555.
....5
5555.
.666.
6....
6666.
6...6
.666.
77777
....7
...7.
..7..
..7..
.888.
8...8
.888.
8...8
.888.
.999.
9...9
.9999
....9
.999.
.....
./|\.
..|..
..|..
..|..
...\.
.--->
.|./.
.|...
.|...
./...
<---.
.\.|.
...|.
...|.
.FFFF
.F...
.FFF.
.F...
.F...
```

DESCIFRA.OUT

```
1 10
```




```
.....11...000....\.......000..11...FFFF
./|\.1.1..0...0.--->./|\.0...01.1...F...
..|....1..0...0.|./...|..0...0..1...FFF.
..|....1..0...0.|.....|..0...0..1...F...
..|..11111.000..|.....|...000.11111.F...
```

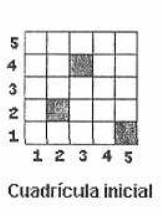

GRADO DE DIFICULTAD:
Implementación: CE Algoritmo:.1

EN CAMINO HACIA EL TESORO

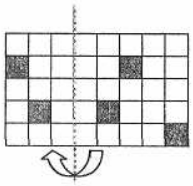
DESCRIPCIÓN

Cada vez estas más cerca de tu objetivo, sin embargo te has topado con un nuevo obstáculo. Para llegar al cuarto en donde se encuentran los tesoros del patriarca es necesario atravesar un piso cuadriculado lleno de trampas mortales. Cada cuadro en el piso puede esconder una trampa mortal, por lo que es necesario saber cuales cuadros son los que no tienen ninguna trampa.

Afortunadamente los OMIncas eran muy olvidadizos y siempre dejaban anotadas instrucciones acerca de cómo evitar sus propias trampas y esta vez no fue la excepción. Hay dibujada en el piso una cuadrícula más pequeña de M columnas por W filas, ésta cuadrícula tiene algunas posiciones marcadas, dichas posiciones indican los cuadros que son seguros para pisar, sin embargo la cuadrícula pequeña tiene menos cuadros que la del piso que mide 100 x 100 cuadros. Junto a la cuadrícula pequeña hay una serie de parejas de números.



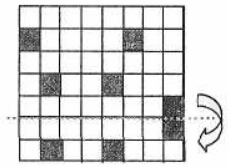
Cuadrícula inicial


Desdoble

Después de meditar un rato y hacer algunas pruebas, te das cuenta que estos números indican dobleces. El primer número indica la orientación del doblez (horizontal o vertical) y el segundo indica cuantos cuadros se doblaron. Observas que si "desdoblas" por completo la cuadrícula pequeña siguiendo las instrucciones, obtienes una cuadrícula de 100 x 100 con marcas en diferentes lugares que indican los cuadros seguros.

Cuando se hace un desdoble las marcas en la cuadrícula se reflejan utilizando como eje de simetría el extremo de la cuadrícula. Los desdobles horizontales siempre se harán por el lado izquierdo y los verticales siempre por abajo. Nunca se pueden desdoblar más cuadros de los que tiene la cuadrícula inicial.

Un "desdoble" se hace de la siguiente forma, supongamos que la cuadrícula pequeña es de 5x5 y que el desdoble dice (horizontal, 3 seguido por un vertical, 2). Después de desdoblar se tendría:


Desdoble vertical

GRADO DE DIFICULTAD:
Implementación: SE Algoritmo:..2

PROBLEMA

Debes escribir un programa que lea la cuadrícula inicial y la secuencia de desdobles que vaya "desdoblando" la cuadrícula inicial para determinar todos los puntos de la cuadrícula del piso que se pueden pisar.

ENTRADA

Tu programa deberá leer del archivo de entrada CAMINO.IN los siguientes datos, en la primera línea 2 números, 3 <= *M, N* <= 16, en las siguientes N líneas deberás leer *M* números en cada línea indicando si las posiciones de la cuadrícula inicial son seguras o no, un 'O' indica una posición marcada con trampa y un '1' indica una posición marcada como segura.

Después de leer la cuadrícula en la siguiente línea esta el número 1 <= D <= 200 que indica el número de desdobles que se van *a realizar* sobre la cuadrícula y en las últimas D lineas hay 2 números en cada línea indicando la orientación del doblez y el número de cuadros doblados. Para la orientación del doblez un 'O' indica un doblez horizontal y un '1' indica un doblez vertical.

SALIDA

Tu programa deberá escribir en el archivo de salida CAMINO.OUT todas las posiciones de la cuadrícula grande que se pueden pisar, las posiciones deberán estar dadas como coordenadas (x, y). considerando que la esquina inferior izquierda de la cuadrícula del piso es la coordenada (1,1).

NOTA: Después de hacer todos los desdobles, siempre obtendrás un cuadrado de 100 x 100 cuadros que representa el piso cuadriculado

EJEMPLO

CAMINO.IN

```
16 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
7
0 16
0 32
0 36
1 10
1 20
1 4
1 50
```

CAMINO.OUT

```
18 3
23 3
50 3
55 3
82 3
87 3
18 98
23 98
50 98
55 98
82 98
87 98
```


GRADO *DE DIFICULTAD:*
Implementación: SE Algoritmo:.2

ABRIENDO LA PUERTA

DESCRIPCIÓN

El primer paso esta dado, ya encontraste la tumba, sin embargo, tus problemas apenas comienzan, primero deberás abrir la puerta y una vez adentro deberás sortear todas la trampas en el camino al tesoro. ¡Pero comencemos por el principio, antes que cualquier cosa hay que abrir la puerta!

Al llegar a la tumba te encuentras con una puerta repleta de palancas, todas las palancas son negras, salvo una de ellas, la cual es de un color rojo sangre. En la puerta hay además una inscripción y una secuencia de números. Tras leer la inscripción descubres que para abrir la puerta es necesario bajar todas las palancas, sin embargo, las palancas se deberán ir bajando siguiendo la secuencia de números en la puerta, con la condición de que la última palanca que se baje sea la palanca roja. Hay que ser muy cuidadoso, porque la inscripción dice que si se rompe la secuencia o si se baja la palanca roja antes de bajar todas las negras "... su alma será condenada a los fuegos eternos mientras dure la era del quinto sol."

La forma de ir bajando las palancas es la siguiente: inicialmente se baja una palanca, que puede ser cualquiera. A partir de ésta se cuenta un número x, de palancas, donde *x1* es el primer número de la secuencia escrita en la puerta y se baja la palanca en donde quedaste, de ahí, cuentas x_2 palancas dónde x_2 es el segundo número de la secuencia y nuevamente bajas la palanca correspondiente. Continuando con la secuencia hasta que todas las palancas estén abajo. Las palancas siempre se cuentan en el mismo sentido, es decir, siempre hacia la derecha y si llegas al final debes de seguir contando por el principio. Al contar las palancas no debes tomar en cuenta las que ya estén abajo.

PROBLEMA

Debes escribir un programa que conociendo el número de palancas, la posición de la palanca roja y la secuencia de números, te diga cual es la primera palanca que debes bajar para que siguiendo la secuencia de números, la última palanca que bajes sea la palanca roja.

ENTRADA

Tu programa deberá leer del archivo de entrada **ABRE.IN** los siguientes datos: en la primera línea el número 9 <= *P* <= 20000 de palancas en la puerta, en la segunda línea el número *R,* dónde 1<= *R* <=*P* que indica la posición de la palanca roja contando a partir de 1 y por último en la tercera línea ***P-1*** números separados por un espacio que indican la secuencia que se debe llevar en el conteo de palancas. Los números de la secuencia de saltos son todos entre 1 y 16.

SALIDA

Tu programa deberá escribir en el archivo de salida **ABRE.OUT** un único número indicando la posición de la primera palanca que se debe bajar.

EJEMPLO

```
ABRE.IN
9
7
1 2 4 5 3 1 7 8

ABRE.OUT
5
```


GRADO DE DIFICULTAD:
Implementación: S Algoritmo:...3

EL TESORO

DESCRIPCIÓN

Ha sido un camino difícil, pero al fin llegaste al tesoro del gran patriarca, exaltado por el brillo de los tesoros, estas a punto de tomar cierta figura grande que parece hecha de oro sólido e incrustada con un sin número de piedras preciosas, cuando te das cuenta que debajo de la figura hay un mecanismo sensible al peso que activa una trampa si se retira la joya.

De cualquier forma, no todo esta perdido, debajo de la figura está escrito el nombre de un dios OMInca y junto a la mesa donde está la figura hay una serie de cubos de plomo de diferentes pesos, los cuales tienen una letra diferente en cada una de sus seis caras. Deduces que los cubos están ahí para ser utilizados como medidas de peso, tras un largo análisis te das cuenta de que si tomas cubos tales que puedas formar el nombre del dios OMInca escrito debajo de la figura, apilando cubos uno sobre otro, siempre con la condición de que las caras de dos cubos apilados que se tocan deben tener la misma letra, el peso obtenido es exactamente igual al de la figura de oro. Cuando se forma el nombre del dios, este debe quedar escrito en las caras de los cubos que quedan de frente a ti. En la siguiente figura se muestra el nombre del dios OMITL.

Lo más importante de tomar en cuenta a la hora de formar el nombre de un dios *es* que las caras que se tocan de los cubos deben tener la misma letra. Por ejemplo si el nombre del dios fuera OMITLA en lugar de OMITL, el cubo que va arriba de la 'L' debe tener una letra 'U' en su parte inferior, es posible también poner las letras volteadas, por ejemplo, la siguiente manera de formar OMITL también es válida.

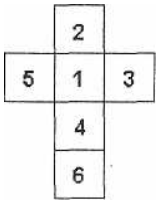
Cada cubo tiene siempre 6 letras diferentes y para todos los casos de prueba que se utilizan en la evaluación, existe al menos una forma de escribir el nombre del dios cumpliendo con las condiciones del problema.

PROBLEMA

Debes escribir un programa que encuentre una manera de formar el nombre del dios utilizando los cubos que se proporcionan en el archivo de entrada, cumpliendo con las limitantes del problema.

ENTRADA

Tu programa deberá leer del archivo de entrada TESORO.IN los siguientes datos, en la primera línea el numero 4 <= *N* <= 40 de cubos. En las siguientes N debe leer 6 caracteres, los cuales no están separados por ningún espacio, y que indican las letras en las caras de los cubos. Las letras vienen en orden, del 1 al 6 y las caras de los cubos se numeran de la siguiente forma:



En la última línea del archivo se encuentra el nombre del dios el cual nunca será menor de 4 letras ni mayor de 10 letras.

GRADO DE DIFICULTAD:
Implementación: CE Algoritmo: 2

NOTA: Para los cubos y el nombre del dios se utilizaran únicamente las letras mayúsculas de la 'A' a la 'Z'

SALIDA

Tu programa deberá escribir en el archivo de salida TESORO.OUT los cubos utilizados para formar el nombre del dios. Los números de los cubos utilizados deberán ir en una sola línea separados por un espacio y escritos en el orden en el que se fueron utilizando para formar el nombre del dios, es decir, siempre se irá primero el cubo que tenga la primera letra del nombre del dios. Los números de los cubos se asignan según su orden en el archivo de entrada contando a partir del 1.

En el caso de que exista más de una solución, solo deberás escribir una de ellas, cualquier solución será considerada correcta.

EJEMPLO

TESORO.IN

```
6
CVKSMZ
PXLAHU
HTINCE
ALQIOB
LCTZAV
CNEZMU
ZEUS
```

TESORO.OUT

```
5 3 6 1
```


GRADO DE DIFICULTAD:
Implementación: CE Algoritmo:..2

TIPS PARA

TAXIS

El truco para resolver este problema fácilmente, es ver a Taxtlán como un grafo en el que cada región es un nodo y las fronteras entre dos regiones son las aristas. Una vez teniendo ei mapa representado por un grafo la solución de ambos incisos resulta evidente. Para el inciso (1) hay que hacer una búsqueda a lo ancho, empezando ya sea por la región que contiene al hotel o la región que contiene a la termina!, hasta que se encuentre el otro punto. Para resolver el inciso (2) se puede utilizar la misma técnica (búsqueda a lo ancho), solamente que se tiene que hacer partiendo desde cada una de las regiones para poder escoger la que tenga un mínimo costo.

Quizás ia parte más difícil del problema está entonces en representar el mapa como un grafo. Se puede lograr ésto revisando los vértices de cada una de las regiones rectangulares y buscando con cuales coincide, pero éste es un trabajo de implementación un poco largo y propenso a errores. Una forma más fácil de implementarlo, aunque tal vez un poco más lenta es ia siguiente: Haz un arreglo del tamaño de Taxtlán y conforme vayan llegando las regiones, ve llenando el mapa con números, de acuerdo al número y localización de la región, una vez lleno el mapa basta con irlo recorriendo recursivamente (como un fill), claro que cuando estés llenando una región (digamos la 0), lo haces hasta encontrarte un número diferente a O, si encuentras este número entonces en una matriz de adyacencia marcas que hay frontera entre la región O y la región número x, cuando todos los cuadros del mapa hayan sido recorridos tu grafo estará completo. E! número de cuadros máximo en Taxílán es un poco mayor a 10,000 y si marcas correctamente, cada cuadro se recorre solo una vez, por lo que el tiempo no es gran problema y sin embargo la implementación se facilita mucho y hay menor posibilidad de errores.

CIRCUITO

Este no es un problema particularmente bueno, su solución es bastante sencilla pero requiere de una implementación un poco larga.

Primero hay que localizar las compuertas, ésto es fácil ya que puedes buscar las E's o las S's en la matriz del circuito, asi tendrás la primera parte de la solución del problema.

Para resolver ia segunda parte, hay que simular el circuito, pero primero será necesario buscar una estructura de datos que nos facilite la simulación. Todas las compuertas tienen dos entradas y una salida lo que nos lleva a pensar en un árbol binario, en que los nodos representen el valor de salida de la compuerta y los hijos estén conectados a las entradas de la misma. Una vez teniendo éste árbol basta recorrerlo en post-orden (primero los hijos y luego el nodo) para obtener las salidas en cada una de las compuertas.

Una vez localizadas las compuertas hay que ver a donde están conectadas para poder crear el árbol de simulación, hay que buscar las letras C's y haciendo una búsqueda recursiva encontrar los puntos que unen, una vez conociendo los puntos hay que ver si pertenecen a la entrada o salida de alguna compuerta. Esto nos permitirá ir creando un árbol. Aunque en general no se recomienda el uso de memoria dinámica para estructuras de datos, en este caso resulta mucho más fácil armar el árbol con una estructura de apuntadores que con un árbol en memoria estática.

Una cosa importante es tomar en cuenta que el circuito puede quedar dividido en más de un árbol de simulación, de hecho para cada compuerta puede estar en su propio árbol, por lo que resulta importante verificar que todos los árboles hayan sido simulados antes de terminar.

SUBMARINO

La solución de este problema puede encontrarse en cualquier libro de algoritmos de grafos. Para la primera parte hay que utilizar I algoritmo de *Dijkstra* y para la segunda parte hay que utilizar el algoritmo de puntos de articulación.

ÁTOMOS

Este es uno de los problemas más difíciles que se ha aplicado en un examen nacional, es copia exacta de un problema utilizado en la Olimpiada Internacional de Holanda, con la dificultad añadida de que el número de compuestos es limitado, por lo tanto se hace bastante más difícil que el usado en la Olimpiada Internacional.

Este problema se resuelve usando Jos métodos de programación dinámica con elementos limitados, por lo que es necesario que sepas usarlos o al menos que conozcas y entiendas la solución de! *problema de la mochila o knapsack.*

La dificultad real del problema, radica en el hecho de que la cantidad de compuestos de que dispone cada proveedor es limitada. Si la cantidad de compuestos no fuera limitada, entonces el problema se resolvería utilizando la técnica del *knapsack* pero ampliada a tres dimensiones, en vez de llenar un arreglo unidimensional se iría llenando un arreglo tridimensional.

Primero que nada, llena el arreglo con los precios correspondientes para comprar solo átomos por separado, con esto tendrás al inicio un precio máximo para comprar. Uno a uno ve tomando los compuestos y empezando desde el punto al que quieres llegar (6.3.4), cuyo valor inicial para el ejemplo es 6*8 + 3*9 + 4*7 = 103, escoge la mejor opción entre las que hay (103) en este caso y la opción de tomar el compuesto que se esta utilizando una, dos, tres..... hasta las veces que ese compuesto este disponible. Por ejemplo, si tomarnos el primer compuesto del ejemplo y estamos en (6,3,4} podemos escoger entre dejar lo que esta (103) y utilizar e! compuesto una vez, que sería lo que esta en (4,2,4), más lo que vale el compuesto (96) , o utilizar el compuesto dos veces, que seria lo que esta en (2,1,4), más dos veces el valor del compuesto (89). Obviamente el mejor valor es el (89), este valor nos indica la solución óptima para llegar a (6,3,4), si solo dispusiéramos de átomos separados y del compuesto uno. Ahora, no basta con calcular el óptimo para (6,3,4), es necesario rellenar la tabla con los óptimos de usar el compuesto 1, de modo que cuando utilicemos el compuesto 2, el compuesto 1 ya este tornado en cuenta. Una vez que se haya rellenado la tabla con todos los compuestos existentes se tienen los valores óptimos para cada combinación de átomos.

Otra interesante tarea para el lector, es ver porque esta solución funciona ¡¡suerte¡¡ .

CINTAS

Para resolver este problema hay que observar un poco como pueden ir los discos en una cinta. Cada disco solo puede ser grabado de dos formas, solo ocupando de preferencia ambos

lados de la cinta (obviamente porque se aprovecha mejor el espacio), o compartiendo la cinta con otro disco.

Aquí es donde entra el axioma que resuelve el problema ©.

Si los discos están ordenados según su tamaño en una lista, entonces, para cualquier solución que encuentres en la que dos discos no consecutivos en la lista compartan una cinta, existe una solución igua! o mejor en la que sólo discos consecutivos en la lista comparten cintas.

El porque lo anterior es cierto debe ser fácil de comprobar y queda como tarea para el lector.

Para encontrar la solución óptima, basta que para cada disco escojamos la mejor opción, entre ponerlo solo en la cinta más chica en que pueda caber, o ponerlo en la misma cinta que con el disco que esta junto a él en la lista.

SUPERCONEJO

Este es un problema clásico de búsqueda a lo ancho. Muy parecido al problema del caballo de ajedrez. La primera parte es trivial, haciendo una búsqueda por nivel. Para la segunda parte, basta con ir marcando los cuadros que has visitado y abstenerte de volverlos a introducir en la cola del proceso, cuando la cola quede vacia, entonces habrás visitado todos los cuadros que podías visitar.

SUMAS

Este problema se puede resolver por una búsqueda en profundidad, pero como el espacio se soluciones es de 1,000,000,000,000, será necesario podarlo, pero por suerte puedes usar la siguiente poda: una vez que te pases del número al que quieres llegar, corta la rama en la que te encuentres, esa poda debe ser suficiente para resolver el problema en tiempo.

OMICEL

Este es otro problema más de búsqueda a ío ancho, con la leve variante de que en este problema deben efectuarse cuatro búsquedas, una por cada canal y seleccionar la mejor de ellas.

PRÁCTICA

Los problemas como este, se conocen como problemas heurísticos, ya que no es posible encontrar un algoritmo que asegure la solución óptima en el tiempo que se da. Por lo tanto debes hacer un algoritmo que se acerque lo más posible a la solución óptima en el tiempo dado.

Para este problema se implementaron cuatro soluciones oficiales, cada una con una heurística diferente, dos de las heurísticas eran algoritmos glotones, es decir, toman lo mejor que hay en ese momento y las otras dos eran algoritmos genéticos. Para hacer un algoritmo genético, es necesario partir de un algoritmo rápido que te acerque a la solución, mientras más se acerque a la solución tu algoritmo inicial, mejor será el desempeño de tu algoritmo (para los algoritmos genéticos que se propusieron, se tomo como punto de partida los algoritmos glotones de las dos primeras soluciones), una vez que se tiene una solución inicial, el algoritmo genético, mientras cuente con tiempo, debe hacer cambios aleatorios en su solución y verificar si la solución mejora o no. No da el mismo resultado utilizar cualquier tipo de cambio, mientras más rápido sea el proceso de cambio y verificación del resultado, más cambios podrás realizar en íiempo y tu algoritmo tendrá mejores oportunidades, también hay cambios que tienen una alta probabilidad de mejorar el resultado, mientras que otros por más que se hagan, difícilmente van a brindar alguna mejora. Experimenta con diferentes cambios para ver cual da mejores resultados.

TORNEO

Este problema es enteramente matemático, aunque se puede buscar la solución por medio de búsqueda, la verdad es que no vale fa pena, ya que para la mayoría de los casos no resolverla el problema en tiempo.

Para resolver el problema hay que dividirlo en casos. El primer caso va a ser cuando los dos jugadores se encuentren en la misma mitad, es decir, el número de ambos es mayor que la mitad de los jugadores o menor que la mitad de los jugadores. Cuando dos jugadores de encuentran en la misma mitad, siempre juegan la final. Para que dos jugadores se enfrenten es necesario que se encuentren en mitades opuestas, pero si se encuentran en la misma mitad basta con que en esa ronda pierdan todos los jugadores que se encuentran entre los jugadores A y B para que estos queden juntos al final o al principio de la tista y no sea posible que se enfrenten antes de la final.

El segundo caso, es cuando A y B se encuentran en mitades opuestas. Si es posible llevarlos a una mitad de la lista, entonces el problema se convierte en el caso anterior y queda resuelto. Si no es posible llevar a ambos jugadores a un solo lado de la lista, es necesario evitar lo más posible que se encuentren, resulta evidente que si no se encuentran en posiciones completamente opuestas tienen X jugadores en medio, pero cada ronda que pasa se acercan al menos un jugador, ya que aunque todos los jugadores entre ellos permanezcan, A y B forzosamente tienen que ganar, para que sigan en el juego. Debido a que ei torneo completo dura log2(N) rondas, entonces los dos jugadores se enfrentan antes de la final, si ambos se encuentran en mitades opuestas y los dos están a una distancia menor ai número total de rondas de sus extremos, en cuyo caso la ronda en la que se enfrenten es igual a la distancia del extremo a la que se encuentre el jugador que esté más lejos de su respectivo extremo.

BIBLIOGRAFIA

Algunos de los libros que te recomendamos para aprender el lenguaje y los algoritmos de programación son:

Estructuras de datos y algoritmos
Aho, Hopcroft y Ullman
Addison Wesley

Programación en Pascal
Grogono
Addison Wesley

El lenguaje de programación C
Kernighan y Ritchie
Prentice Hall

Algoritmos en C++
Sedgewick
Addison Wesley

Estructura de datos en Pascal
Tenenbaum y Augenstein
Prentice Hall

Algoritmos y Estructuras de Datos
Wirth
Prentice Hall

www.olimpiadadeinformatica.org.mx