"First, solve the problem. Then, write the code" John Johnson.

ProgramaMe 2024
Final Nacional
Problemas

Patrocinado por

KPMG

MÁSTER EN DESARROLLO DE VIDEOJUEGOS

CÁTEDRA BOSCH-UCM en Inteligencia Artificial aplicada a Internet de las Cosas

Ejercicios realizados por

Facultad de Informática
Universidad Complutense de Madrid

Realizado en la **Facultad de Informática (UCM)**
7 de junio de 2024

Universidad Complutense Madrid

7 de junio de 2024
https://www.programame.com

Listado de problemas

Autores de los problemas:

- Marco Antonio Gómez Martín (Universidad Complutense de Madrid)
- Pedro Pablo Gómez Martín (Universidad Complutense de Madrid)

Revisores:

- Alberto Verdejo (Universidad Complutense de Madrid)

Imágenes y fotografías:

- Problema D, "*Se buscan programadores de COBOL*": ilustración de `https://www.cbtnuggets.com/`
- Problema E, "*La mezcla de Juancar*": ilustración de los pasos realizada a partir de imágenes con licencia Pixabay.
- Problema G, "*La división perdida*": retrato de Newton hecho por Godfrey Kneller en 1689 (óleo sobre tela); las divisiones son creación propia.
- Resto de fotos e ilustraciones con licencia Pixabay.

A
Ganando el concurso

Luisa Bionda es una fantástica programadora que es capaz de resolver los problemas más difíciles de los concursos. Por desgracia, cuando ha participado en alguno de ellos su resultado ha sido mediocre. El problema es que como está convencida de que sabe resolverlos todos, empieza por el primero que aparece en el cuadernillo y va por orden hasta que termina el tiempo.

El punto débil de su estrategia, claro, es que el concurso lo gana el participante que más problemas resuelve, sin importar cómo de difíciles sean. Si en un determinado concurso el primer problema resulta ser el más difícil y lleva mucho tiempo resolverlo, Luisa al final lo conseguirá pero para cuando lo haga sus competidores puede que hayan conseguido resolver varios más sencillos.

Si hay dos contrincantes que, al acabar el concurso, tienen el mismo número de problemas resueltos entonces gana el que *menos tiempo acumulado* haya necesitado. Este se calcula sumando el *tiempo de "penalización"* ocasionado por cada problema resuelto, que consiste en el tiempo transcurrido *desde que comenzó el concurso* hasta que el problema se resolvió.

Gracias a su experiencia resolviendo problemas, Luisa es capaz de leer el título de un problema y decir el tiempo que necesitará para resolverlo. ¡Y nunca se equivoca! Ahora tiene que convencerse de que, con esa información, es posible ganar muchos más concursos si la utiliza con inteligencia.

Entrada

Cada caso de prueba comienza con un número $1 \leq n \leq 200$ que indica la cantidad de problemas planteados en un determinado concurso de programación de participación individual. A continuación viene un número $1 \leq t \leq 10.000$ con la duración, en minutos, del concurso.

Tras esos dos valores aparecen, en otra línea, n números entre 1 y 10.000 indicando cuánto tarda Luisa en resolver cada problema desde que se lo lee hasta que consigue su veredicto de envío correcto al enviarlo al juez del concurso. Luisa nunca se equivoca, y cuando envía una solución siempre está bien.

La entrada termina con dos ceros.

Salida

Por cada caso de prueba el programa escribirá el número máximo de problemas que Luisa puede resolver durante el concurso, seguido de la menor penalización que puede conseguir con ellos.

Entrada de ejemplo

```
3 2
1 1 1
3 20
10 15 5
4 100
120 130 110 150
0 0
```

Salida de ejemplo

```
2 3
2 20
0 0
```

B
Abreviaturas

Una abreviatura es la representación gráfica reducida de una palabra o grupo de palabras, obtenida por eliminación de algunas de sus letras o sílabas. Algunos ejemplos son *dcha.* ("Vive en el 3.º dcha."), *sra.* ("Saluda a la sra. Luisa") o *cap.* ("Para más información consulte el cap. VII").

Las abreviaturas *por truncamiento extremo* se forman cuando únicamente se mantiene la primera letra de la palabra abreviada. Suelen aparecer cuando lo que se abrevia está compuesto por varias palabras, como en *r. p. m.* (revoluciones por minuto) o *b. d.* (base de datos). Estas construcciones no deben confundirse con las siglas. Las abreviaturas incorporan puntos y, cuando se leen, *se sustituyen por la palabra completa*, justo al contrario de lo que ocurre con las siglas.

En las abreviaturas por truncamiento extremo, el plural se crea duplicando las letras conservadas, poniendo un único punto para cada pareja y separándolas por un espacio, como en *FF. AA.* (Fuerzas Armadas) o *EE. UU.* (Estados Unidos).

Entrada

La entrada comienza con un número indicando cuántos casos de prueba deberán procesarse. Cada uno, en su propia línea, se compone de una abreviatura por truncamiento extremo formada por una o más letras. Cada letra, del alfabeto inglés, estará siempre seguida de un punto y, si no es la última letra de la abreviatura, de un espacio.

Ninguna abreviatura tendrá más de 10 letras.

Salida

Por cada caso de prueba el programa escribirá la versión en plural de la abreviatura siguiendo las reglas de la ortografía descritas.

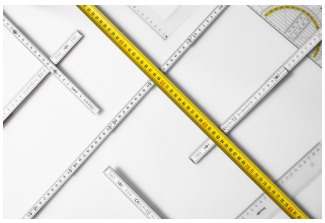
Entrada de ejemplo

```
3
B. D.
S. O.
f.
```

Salida de ejemplo

```
BB. DD.
SS. OO.
ff.
```

C
Distancia de Hamming



La *distancia de Hamming* entre dos vectores de la misma longitud es el número de elementos *distintos* entre ellos. Indica el mínimo número de cambios que hay que realizar para convertir uno en otro.

En teoría de códigos se utiliza para definir conceptos útiles para categorizar códigos de detección y corrección de errores. En este contexto, los "vectores" son secuencias de bits. La distancia de Hamming cuenta el número de bits que son diferentes. En redes de comunicación esto indica el número de errores de transmisión.

Entrada

El programa deberá leer de la entrada un primer número con la cantidad de casos de prueba que deberán ser procesados.

Un caso de prueba está compuesto por dos números entre 0 y 2^{63}–1 separados por un espacio.

Salida

El programa deberá escribir, por cada caso de prueba, la distancia de Hamming de los dos números interpretados como secuencias de bits.

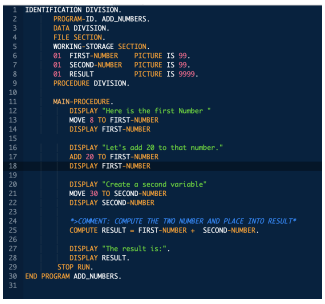
Entrada de ejemplo

```
3
0 1
3 0
255 255
```

Salida de ejemplo

```
1
2
0
```

D
Se buscan programadores de COBOL



```
IDENTIFICATION DIVISION.
       PROGRAM-ID. ADD_NUMBERS.
       DATA DIVISION.
       FILE SECTION.
       WORKING-STORAGE SECTION.
       01  FIRST-NUMBER    PICTURE IS 99.
       01  SECOND-NUMBER   PICTURE IS 99.
       01  RESULT          PICTURE IS 9999.
       PROCEDURE DIVISION.

       MAIN-PROCEDURE.
           DISPLAY "Here is the first Number "
           MOVE 8 TO FIRST-NUMBER
           DISPLAY FIRST-NUMBER

           DISPLAY "Let's add 20 to that number."
           ADD 20 TO FIRST-NUMBER
           DISPLAY FIRST-NUMBER

           DISPLAY "Create a second variable"
           MOVE 30 TO SECOND-NUMBER
           DISPLAY SECOND-NUMBER

           *>COMMENT: COMPUTE THE TWO NUMBER AND PLACE INTO RESULT*
           COMPUTE RESULT = FIRST-NUMBER +  SECOND-NUMBER.

           DISPLAY "The result is:".
           DISPLAY RESULT.
           STOP RUN.
END PROGRAM ADD_NUMBERS.
```

Alba Cil Illa ha sido desde siempre una amante de los lenguajes de programación esotéricos más enrevesados. Ha pasado innumerables horas "jugando" con *Brainfuck*, *Befunge* o *Whitespace*.

Aunque nunca lo ha hecho con ánimo de sacar partido a esos conocimientos más allá de ir vacilando a sus conocidos (al fin y al cabo esos lenguajes no tienen ninguna utilidad práctica) es posible que eso esté a punto de cambiar. Ha visto una oferta de trabajo que busca programadores en COBOL y ha pensado que, por muy complicado que sea, seguro que lo podrá aprender en poco tiempo y ser contratada. Su habilidad con *Brainfuck* seguro que le sirve para interiorizar rápidamente cómo funciona COBOL. Pero la realidad es que ese antiguo lenguaje de medidados del siglo XX parece más esotérico que los lenguajes que ha utilizado en los últimos tiempos. ¡Lo de usar lenguaje natural para sumar es de lo más desconcertante!

Para ayudarla a aprenderlo crearemos un intérprete de un subconjunto muy reducido de COBOL en el que tendremos asignación de valores a variables, sumas y escritura de variables por pantalla. Para hacerlo más sencillo nos tomaremos, además, ciertas "licencias" que no se corresponden con el lenguaje original.

Nuestros programas estarán compuestos de órdenes, cada una en una línea. Todas las órdenes terminan con un carácter punto ('`.`'), y todos los programas terminan con la orden `STOP`.

Las tres órdenes que admitiremos (además de la de final de programa) son `MOVE` para dar un valor numérico a una variable, `ADD` para hacer operaciones de suma y `DISPLAY` para escribir el valor de las variables.

En nuestro caso las variables serán palabras compuestas por como mucho 5 letras minúsculas del alfabeto inglés. Aunque COBOL necesitaba declarar esas variables en una sección de código independiente, en nuestro caso se declararán automáticamente en su primera aparición en una orden `MOVE`.

La orden `MOVE` tiene la forma `MOVE [valor] TO [variable]` donde `[valor]` es un número no negativo menor que 2×10^9 y `[variable]` es el nombre de una variable. La operación crea la variable si no existía ya y le da el valor indicado.

La orden `DISPLAY` tiene, tras la palabra clave, una lista con los nombres de las variables que hay que escribir. Aparecerán separadas por espacios y asumiremos que todas ellas han sido declaradas para almacenar como mucho 5 dígitos decimales por lo que si tienen que guardar un valor que supere ese número de dígitos, los más significativos se perderán.

Por último, la orden `ADD` tiene dos variantes:

- `ADD var1 ... varN TO destVar1 ... destVarM`: hace la suma de todas las variables iniciales `var1`...`varN` (serán como mucho 10) y suma el resultado a las variables `destVar1`...`destVarM`. Podría ocurrir que la misma variable apareciera varias veces en ambas listas.

- `ADD var1 ... varN TO otherVar GIVING destVar1 ... destVarM`: hace la suma de todas las variables `var1`...`varN` (también 10 como mucho) y `otherVar` y el resultado lo *asigna* a las `destVar1`...`destVarM` (en este caso, `otherVar` *no* se modifica). Igual que con la primera variante, la misma variable puede aparecer varias veces.

Entrada

La entrada comienza con una línea con el número de programas que vendrán a continuación.

Cada programa es una secuencia de líneas con las órdenes a ejecutar, que terminan con `STOP`. Cada orden sigue el formato explicado más arriba, donde todas las palabras y números están separados únicamente por un espacio. Todas las líneas terminan con un punto ('`.`') pegado a la última palabra de la orden.

Salida

Cada orden `DISPLAY` de los programas generará una línea con el valor de las variables indicadas separadas por espacios. Recuerda que las variables se asumen declaradas como `9(5)` que significa que son capaces de guardar 5 caracteres numéricos.

Tras cada programa se escribirá una línea con tres guiones (`---`).

Entrada de ejemplo

```
1
MOVE 10 TO a.
MOVE 20 TO b.
ADD a TO b.
DISPLAY a b.
MOVE 0 TO c.
ADD a b TO c.
DISPLAY a b c.
MOVE 100100 TO d.
DISPLAY d.
ADD a b TO c d.
DISPLAY a b c d.
ADD c d TO a GIVING b d.
DISPLAY a b c d.
ADD a TO c c.
DISPLAY c.
STOP.
```

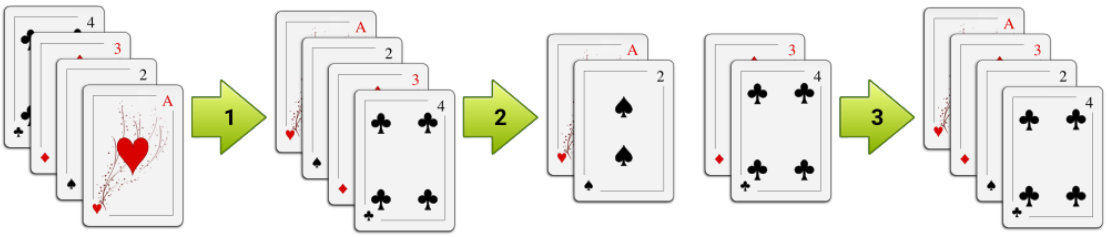
Salida de ejemplo

```
10 30
10 30 40
100
10 30 80 140
10 230 80 230
100
---
```

E
La mezcla de Juancar

Juancar Nariz es un experto mago capaz de hacer trucos de cartas increíbles, que le han llevado a actuar a lo largo y ancho de todo el mundo. Tiene una soltura envidiable manejando los naipes y es capaz de hablar mirando al público mientras mezcla un mazo de cartas de forma que queden colocadas tal y como las necesita para su próximo efecto.

Uno de sus métodos de mezcla, que ejecuta a velocidad pasmosa, comienza invirtiendo el orden de todo el mazo, de modo que las cartas que estaban abajo pasen a estar arriba en sentido opuesto (paso 1 en el siguiente dibujo). Si tiene al menos cuatro cartas, a continuación divide la pila en dos grupos exactamente iguales (paso 2) y los entrelaza al estilo americano, poniendo primero la carta inferior del lado izquierdo, luego la inferior del lado derecho, luego la segunda del izquierdo, y así sucesivamente (paso 3).



Tiene tan entrenado el proceso que nunca comete el más mínimo error y ante una configuración de partida sabe el orden final exacto que conseguirá. Cuando tiene la sensación de que el público también lo conoce, complica más el proceso para despistar. Tras dividir el mazo en dos mitades cuando tiene al menos 4 cartas, antes de entrelazar las cartas aplica el mismo procedimiento de mezcla de manera separada y completamente a cada una de las dos mitades, lo que puede provocar que también se aplique a las dos mitades de cada mitad, etcétera.

Entrada

Cada caso de prueba comienza con un número n potencia de 2 entre 2 y 16.384. A continuación, en otra línea, vienen n números indicando la secuencia de cartas, de abajo hacia arriba, que Juancar va a mezclar. Las cartas se representan con números del 1 al 100.000 y podría haber algunas repetidas.

La entrada termina con un 0, que no debe procesarse.

Salida

Por cada caso de prueba el programa escribirá, en la salida estándar, el orden de las cartas tras la mezcla cuando sospecha del público y complica el proceso, separando cada número por un espacio.

Entrada de ejemplo

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

Salida de ejemplo

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

F
Higos robados

En la comarca donde vive la pequeña Manola Drona hay multitud de fincas con higueras que, cuando llega la época, dan unos frutos dulces y jugosos. Es habitual que, después de clase, ella y su panda de amigos vayan a colarse aprovechando cualquier hueco entre las vallas para "liberar de peso" las ramas de los árboles. "¡Mejor para nosotros que para los pájaros!" — se justifican, ignorando la posibilidad de que los dueños legítimos de esos higos entren también en la ecuación.

Cuando, a salvo de miradas indiscretas y lejos ya de los árboles que han esquilmado, llega el momento de repartirse el botín, lo hacen de manera equitativa entre todo el grupo. Eso sí, Manola, como jefa de la pandilla, es la que se lleva los higos sobrantes si el reparto no es exacto.

Entrada

La entrada comienza con un primer número que indica cuántos casos de prueba deberán procesarse. A continuación aparecen los casos de prueba, cada uno compuesto de dos números, entre 1 y 10^9, indicando el número de personas que forman la pandilla, contando a Manola, y el número total de higos que han conseguido recolectar.

Salida

Por cada caso de prueba el programa escribirá cuántos higos le corresponden a Manola.

Entrada de ejemplo

```
3
4 4
4 5
5 13
```

Salida de ejemplo

```
1
2
5
```

G
La división perdida

Recientemente se han encontrado en la Royal Society de Londres unos legajos que, se cree, pertenecieron a Sir Isaac Newton. Los especialistas en su obra lo sospechan porque aparecen muchos cálculos matemáticos, entre otras cosas divisiones. Newton, con su agilidad mental, hacía las divisiones rápidamente calculando, para cada dígito del cociente, la multiplicación y la resta de cabeza, y apuntaba únicamente el resto al que luego "*bajaba*" el siguiente dígito del dividendo. Lo que cualquier persona de su tiempo hubiera escrito así:

```
 161803 | 42
-126    ----
 358    3852
-336
 220
-210
 103
 -84
  19
```

Newton lo reducía a:

```
161803 | 42
 358   ----
  220  3852
   103
    19
```

Desgraciadamente, el paso del tiempo ha hecho mella en los legajos y muchas divisiones aparecen dañadas, bien por la humedad, bien por roedores o bien por, directamente, manchas de tinta que han destruído gran parte de la información. En algunas ocasiones ha sido posible recuperar el lado izquierdo de la división, con el dividendo y los restos parciales, pero se han perdido el divisor y el cociente, es decir el número por el que se estaba dividiendo, y el resultado. Los especialistas creen que algunas divisiones podrían dejar entrever que Newton intuyó la Teoría de la Relatividad, pero para asegurarlo es necesario tener certeza de los números por los que dividía.

Entrada

Cada caso de prueba comienza con un número de hasta 1000 dígitos decimales indicando el dividendo de una división. A continuación aparece un número indicando cuántos restos parciales tiene la división, seguido de sus valores. Ten en cuenta que *no* se incluye en esos restos parciales el dígito "*bajado*" del divisor en cada paso.

Aunque Newton tenía una agilidad increíble para dividir números enormes, el divisor perdido siempre tenía un tamaño moderado que no superaba 10^8. Además, ninguna división tiene como único resto al propio divisor, lo que significa que ninguna tiene 0 como cociente.

La entrada termina con un `0`.

Salida

Por cada caso de prueba el programa escribirá el divisor y el cociente que se ha perdido de la división original. Si hay varias posibilidades se escribirán todas de mayor a menor divisor separadas por un guión.

Newton nunca se equivocaba, por lo que se garantiza que siempre hay solución.

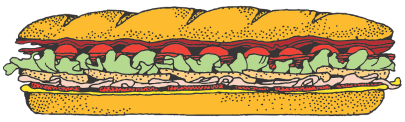
Entrada de ejemplo

```
161803
4 35 22 10 19
258728
3 379 480 392
0
```

Salida de ejemplo

```
42 3852
1104 234 - 552 468
```

H
El bocadillo más rápido



Borja Monse Rano siempre va con prisa. Tiene que comer cualquier cosa entre reunión y reunión porque la vida no le da para más. El problema es que muchas veces se le olvida llevarse de casa algo que echarse al buche y tiene que salir corriendo a conseguir algo con lo que matar el hambre.

Afortunadamente, trabaja cerca de la calle *Manjar*, donde hay infinidad de panaderías y charcuterías de renombre. De modo que el olvido no es para él un problema, sino una oportunidad. Llama a un taxi y pide que le lleve a alguna de las panaderías y, una vez conseguido el primer ingrediente, va a pie hasta la charcutería más cercana para comprar un buen jamón con el que rellenarlo.

Se ha dado cuenta de que, con la gran oferta de establecimientos que hay en esa calle, dependiendo de la panadería donde le deje el taxista tiene que andar una distancia diferente para poder comprar el jamón. Como siempre va con prisa, quiere minimizar ese recorrido, aunque eso suponga que tenga que pagar más al taxista para llevarle a una panadería más alejada.

Entrada

El programa deberá procesar múltiples casos de prueba, cada uno compuesto por dos líneas. La primera línea indica la posición de cada una de las panaderías en la calle *Manjar* y la segunda la de cada una de las charcuterías.

Ambas líneas utilizan el mismo formato. Comienzan con un número entre 1 y 200.000 indicando cuántos establecimientos de ese tipo (panadería o charcutería) hay en la calle. A continuación aparece la distancia de cada una de ellas desde el inicio de la calle, ordenadas de menor a mayor. Ninguna distancia es mayor que 1.000.000 y nunca hay dos establecimientos del mismo tipo a la misma distancia.

La entrada termina con un `0` que no debe procesarse.

Salida

Por cada caso de prueba el programa escribirá la menor distancia entre una panadería y una charcutería.

Entrada de ejemplo

```
1 3
2 4 6
3 1 7 11
3 3 5 9
1 100
1 100
0
```

Salida de ejemplo

```
1
2
0
```

Guadian

En un pequeño pueblo del sur del país se está gestando la siguiente gran multinacional llamada a revolucionar el comercio electrónico y la forma en la que la gente adquirirá bienes de consumo. En concreto Yef Besos está a punto de lanzar su tienda por internet a la que ha llamado *Guadian*. Ya tiene acuerdos con proveedores y una flota de furgonetas para distribuir por toda la zona lo que vayan comprando sus futuros clientes. En un par de décadas formará parte de la historia.

Siendo fiel a su apellido, en el diseño de la tienda y de la gestión logística ha mantenido el principio KISS (*keep it simple, stupid*) por lo que, de momento, solo tiene un tipo de caja para hacer los envíos. Da igual que el pedido consista en un par de sacapuntas o en una máquina de coser antigua; todos los paquetes irán en cajas del mismo tamaño. Eso facilita muchísimo el empaquetado pues al fin y al cabo no hay que decidir qué embalaje se ajusta mejor al tamaño del pedido. Además pone las cosas fáciles para apilar las cajas en el almacén pues se pueden poner siempre unas encima de otras sin miedo a que la torre se desestabilice. Lo único que hay que tener en cuenta, eso sí, es la carga máxima que soporta cada una, que dependerá de lo frágil que sea el contenido interior.

Entrada

La entrada está compuesta de varios casos de prueba, cada uno ocupando dos líneas. La primera línea contiene el número n de cajas del almacén (hasta 500.000). A continuación viene una línea con n números, uno por caja, indicando para cada una la cantidad de cajas que pueden ponerse encima de ella sin que el contenido del paquete sufra desperfectos (como mucho 100.000).

Salida

Por cada caso de prueba se escribirá, en una línea independiente, el número mínimo de pilas que se necesitan para organizar todas las cajas.

Entrada de ejemplo

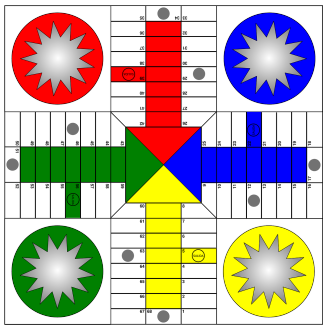
```
3
0 10 4
4
1 10 1 1
```

Salida de ejemplo

```
1
2
```

J

Tiradas del parchís



El parchís es un conocido juego de mesa de fichas y dados donde los jugadores deben conseguir llevar las fichas de su color desde su "casa" hasta la meta. La parte principal del tablero crea un circuito de 68 casillas, y dispone además de un pasillo de cada color de 8 posiciones más. Las fichas comienzan en una posición del tablero diferente según su color, y no recorren el circuito entero, sino que, cuando están a punto de hacerlo, se desvían por el pasillo de su color que las lleva hasta la meta.

Los jugadores lanzan, de forma alterna, un dado que indica el número de casillas que deben hacer avanzar alguna de sus fichas. Estas deben alcanzar la meta *de forma exacta* cuando llegan al final. De otro modo *rebotarán* volviendo hacia atrás a lo largo del pasillo de su color.

Existen varias reglas adicionales e infinidad de variantes (comer fichas, casillas seguras, bloqueos, jugar con dos dados...) pero ignorando todas ellas, ¿de cuántas formas distintas se puede conseguir hacer llegar una ficha a su destino?

Entrada

Cada caso de prueba son dos números. El primero, entre 1 y 25.000, indica el número de casillas que una determinada ficha de parchís debe recorrer desde su posición actual hasta alcanzar la meta. El segundo, entre 1 y 24, indica el número de caras del dado con el que se juega.

Como mucho habrá 5.000 casos de prueba y la mayoría serán con números de casillas moderados.

Salida

Por cada caso de prueba el programa escribirá cuántas secuencias distintas de tiradas de dados existen que hacen que la ficha termine, de forma exacta, en la meta. Como el número es grande, se dará módulo 1.000.000.007.

Entrada de ejemplo

```
4 1
4 2
2 5
71 6
```

Salida de ejemplo

```
1
5
2
861259375
```

K
Cinco pasos al norte

En el ocaso de su vida, para el pirata John Oper Dono ha llegado el momento de retirarse. Está desempolvando los viejos mapas de su camarote, donde apuntaba la forma de alcanzar los tesoros enterrados en las islas del caribe que arrasó en su juventud.

Todos son iguales. Comienzan indicando un punto de referencia, como un árbol característico, una roca con forma llamativa o el inicio de un desfiladero, y a partir de ahí indican la cantidad de pasos que hay que ir dando, en distintas direcciones, para llegar al punto exacto donde está escondido el botín. Por ejemplo "Comenzando en la gran roca encima de la colina, cinco pasos al norte, tres hacia el oeste...".

En las instrucciones, solía crear caminos que hicieran a quien los siguiera volver sobre sus pasos y hasta pasar varias veces por el mismo sitio. En ocasiones ¡el tesoro estaba enterrado al lado de donde se empezaba el recorrido! La imprecisión en la medida de los pasos y los caminos largos, se decía, podían mantener a salvo sus tesoros si los mapas caían en malas manos.

El problema es que, con el tiempo, ha perdido una pierna, un ojo y varios dedos de una mano, y no está para seguir caminos largos. Querría tener el recorrido pintado en un mapa para saber la verdadera distancia desde el origen al destino e intentar así ir en línea recta.

Entrada

Cada caso empieza con un número que indica cuántas instrucciones se dan para alcanzar el tesoro. La siguiente línea tiene la lista con las instrucciones, cada una consistiendo en un número seguido de un punto cardinal (`N`, `S`, `E`, `O`).

Se garantiza que el número de pasos que habrá que dar (pasos, no instrucciones, es decir la suma de todos los números) será como mucho 100.

La entrada termina con un caso sin instrucciones que no debe procesarse.

Salida

Por cada caso de prueba el programa escribirá el mapa del tesoro indicando el camino completo que hay que recorrer.

El origen del camino estará identificado con una "`O`", el final con una "`F`" y el camino en sí mismo se marcará con "." (punto). Es posible que el camino pase varias veces por el mismo sitio, incluído el inicio y el final, que siempre deberán quedar identificados en el mapa. Se garantiza que el inicio y el final nunca coincidirán.

El mapa se rodeará de un rectángulo (o un cuadrado) de caracteres "`#`" de la forma más ajustada posible. Es decir junto a cada lado habrá al menos una celda que forme parte del camino.

Tras cada caso de prueba se escribirá una línea en blanco.

Entrada de ejemplo

```
2
4 N 3 E
2
4 N 3 O
7
1 N 1 O 1 S 2 E 2 S 1 O 1 N
0
```

Salida de ejemplo

```
######
#...F#
#.   #
#.   #
#.   #
#O   #
######

######
#F...#
#   .#
#   .#
#   .#
#   O#
######

#####
#.. #
#.O.#
# F.#
# ..#
#####
```

L
Oferta de Sushi

En el restaurante japonés "El Buen Rollo" tienen un montón de tipos de *sushi*: maki, hosomaki, futomaki... Funcionan a modo de *buffet*. Tienen las piezas de *sushi* en grandes bandejas y los clientes pasan por la barra, cogen los que quieren con unas pinzas o, los más expertos, unos *toribashi*, y pagan en función del precio de cada tipo.

A veces en el restaurante ponen en oferta un *lote*. Si se cogen juntas una pieza de tres tipos concretos de *sushi*, se aplica un precio más barato que si pagara individualmente cada pieza por separado. De hecho a veces la oferta es tan ventajosa que merece la pena coger algún *sushi* de más para completar el lote y poder aplicarla.

Jesús Hisas Himi ha ido con un grupo de amigos al restaurante y, como experto en comida japonesa, se ha encargado él de coger *sushi* para todos. Ahora que se acerca a la caja tiene que pensar rápido si coge o no algunas piezas más de los tipos en oferta para completar lotes y poder aplicarse la oferta más veces.

Entrada

Cada caso de prueba comienza con dos números. El primero indica el número n de tipos de *sushi* que hay en la carta del restaurante (entre 3 y 20). El segundo indica el precio del lote en la oferta.

A continuación aparecen, en otra línea, el precio de los n tipos de sushi disponibles. Los 3 primeros son los precios de los tipos que se incluyen en la oferta. Es decir si se coge una pieza de cada uno de los tres tipos, en lugar de pagar el precio individual de cada uno se pagará el de la oferta, que siempre es más ventajoso. Ningún precio es mayor que 1.000.

La tercera línea indica cuántas piezas de cada tipo de *sushi* había cogido Jesús antes de plantearse coger algunas más para beneficiarse más veces de la oferta. Son muchos amigos pero se garantiza que no ha cogido más de 20 de cada tipo.

La entrada termina con dos ceros.

Salida

Por cada caso de prueba se escribirá el precio total de lo realmente comprado, si Jesús elige las cantidades inteligentemente para completar lotes en oferta y así minimizar el coste total.

Entrada de ejemplo

```
3 7
3 3 3
1 1 2
3 7
4 4 4
2 2 1
5 6
2 2 4 4 5
2 3 2 0 1
0 0
```

Salida de ejemplo

```
10
14
19
```