Concurso de Programación

X AdaByron - Madrid 2024

http://www.ada-byron.es

Cuadernillo de problemas

uc3m | Universidad Carlos III de Madrid

inetum
Positive digital flow

next digital

rtve VALCU

Cátedra RTVE-UC3M

uc3m | Universidad Carlos III de Madrid
Departamento de Informática

Realizado en la **Universidad Carlos III de Madrid (UC3M)**
9 de marzo de 2024

In almost every computation a great variety of arrangements for the succession of the processes is possible, and various considerations must influence the selections amongst them for the purposes of a calculating engine. One essential object is to choose that arrangement which shall tend to reduce to a minimum the time necessary for completing the calculation.
Ada Byron

9 de marzo de 2024

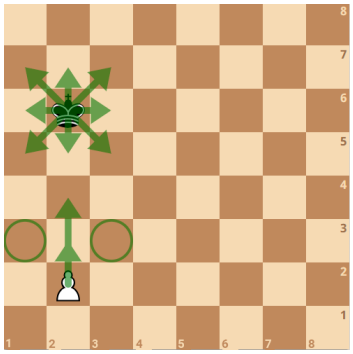
Índice

Autores de los problemas:

- Isaac Lozano Osorio (Universidad Rey Juan Carlos)
- Margarita Capretto (Instituto IMDEA Software)
- Javier del Río López (Walcu)
- Jesús Javier Doménech Arellano (Next Digital)
- Carlos Linares López (Universidad Carlos III de Madrid)
- Sergio Salazar Cárdenas (Universidad Rey Juan Carlos)
- Catalin Sorin Covaci (Universidad Politécnica de Madrid)
- Santiago Tapia Fernández (Universidad Politécnica de Madrid)

Tiempo: 1 segundo

A
Final de Peón vs Rey



El ajedrez es un juego estratégico que involucra diferentes fases: apertura, medio juego y final. Cada una de las fases requiere de una preparación específica. Muchos de los jugadores priorizan la apertura: "Si pierdo en la apertura nunca llegaré a un final". Otros, en cambio, juegan aperturas con planes muy sencillos y sus esfuerzos se dedican a estudiar patrones de finales de ajedrez. Existen muchas maniobras de finales, cada una con conceptos específicos, ya sea para ganar la partida o hacer tablas. Algunos de los finales más complicados son los finales de peones. Es cierto que no hay muchas piezas, pero hay demasiadas posibles combinaciones para los humanos. Este problema se centra en el final de un peón y reyes (para simplificar, asumiremos que el peón siempre será blanco y no existirá rey blanco).

Dada una posición específica en un tablero de ajedrez, tu tarea es determinar si el rey negro puede capturar al peón blanco. El blanco, intentará coronar antes de que sea capturado (puedes ver más abajo algunas reglas simplificadas del ajedrez).

Se garantiza que todas las posiciones son legales, el peón blanco juega de forma óptima y los movimientos deben ser legales según las reglas del ajedrez.

Entrada

Existirán varios casos de prueba, representados por un número N dado en la primera línea. Para cada caso de prueba se muestra una línea que contiene un número entero positivo, representando el tamaño del tablero de ajedrez (T), e inmediatamente a continuación el carácter, "B" o "N", indicando el jugador que realiza el próximo movimiento ("B" representa que es el turno del peón blanco mientras que "N" representa que es el turno del rey negro). La siguiente y última línea de cada caso, contiene cuatro números enteros separados por espacios: la columna C_b y fila F_b del peón blanco, seguido por la columna C_n y fila F_n del rey negro.

Salida

Para cada caso de prueba, imprime "SI", si el rey negro puede capturar al peón blanco, y "NO" en caso contrario.

Entrada de ejemplo

```
2
8 N
1 3 6 2
8 B
1 3 6 2
```

Salida de ejemplo

```
SI
NO
```

Límites

- $1 \leq N \leq 6{,}000$
- $8 \leq T \leq 10{,}000{,}000$
- $1 \leq C_b, C_n, F_n \leq T$
- $2 \leq F_b < T$
- $(C_b, F_b) \neq (C_n, F_n)$
- No existen posiciones donde el peón no pueda mover, se encuentre en última fila o el rey esté en jaque y no le toque mover.

Notas

Algunas reglas de ajedrez relevantes para este problema:

- Una pieza es capturada cuando se ocupa la casilla en la que esta se encuentra por una pieza del rival.
- El rey sólo puede avanzar una casilla en cualquier dirección: hacia arriba, hacia abajo, hacia los lados o en diagonal. En ningún caso el rey puede moverse a una casilla en la que estaría en jaque, es decir, en la que pudiera ser capturado.
- Los peones tienen la particularidad de que se mueven y capturan de diferentes maneras: se mueven hacia adelante, pero capturan en diagonal hacia adelante. Los peones sólo pueden avanzar una casilla en cada jugada, a excepción de cuando se encuentran en la fila 2 (posición inicial en las partidas de ajedrez) en el que puedan avanzar dos casillas. Los peones sólo pueden capturar una casilla situada en diagonal. Nunca pueden retroceder, ni siquiera para capturar una pieza. Los peones no pueden moverse si en su camino encuentran una pieza que ocupe la casilla situada directamente frente a ellos y tampoco pueden capturarla. Es decir, si el peón se encuentra en la casilla (F, C) los únicos movimientos posibles son:
  1. mover a la casilla $(F + 1, C)$ si la casilla esta libre,
  2. mover a la casilla $(F + 1, C - 1)$ o $(F + 1, C + 1)$ si esta ocupadas por una pieza del rival,
  3. si $F = 2$ también pueden moverse a la casilla $(F + 2, C)$ si las casillas $(F + 1, C)$ y $(F + 2, C)$ están libres.
- Se dice que un peón corona cuando llega a la última fila del tablero. En este punto se puede convertir en otra pieza a su elección (a excepción de rey o peón). Ten en cuenta que si llega a la última fila y corona, el siguiente turno es del rival, que si está al lado, puede capturar a la pieza.

Tiempo: 1 segundo

B
Huelga de Agricultores

En medio de una protesta por las difíciles condiciones que enfrentan los agricultores, se ha decidido bloquear algunas carreteras como medida de presión. Como hijo de un agricultor, te han encomendado la tarea de desarrollar un algoritmo para determinar, dadas una ciudad origen y una ciudad destino, el mínimo número de carreteras a bloquear para que sea imposible llegar desde una ciudad hasta la otra.

Un agricultor es capaz de bloquear una carretera, así que el objetivo es requerir del mínimo número de agricultores, permitiendo que el resto continúen trabajando y proveyendo de recursos básicos a la población.

Entrada

La entrada consta de un único caso de prueba. Cada caso comienza con dos enteros N y M, representando el número de ciudades y carreteras en la ciudad, respectivamente. A continuación, siguen M líneas, cada una con dos enteros A_i y B_i entre 0 y $N-1$, indicando que hay una carretera que conecta la ciudad A_i con la B_i (y no viceversa). Se garantiza que como mucho habrá una sola carretera que conecte dos ciudades en la misma dirección. La ciudad origen es la ciudad 0 y la ciudad destino es la ciudad $N-1$.

Salida

En la primera línea debes imprimir un solo entero, el mínimo número de agricultores X necesario para bloquear la circulación entre la ciudad 0 y la ciudad $N-1$. A continuación, debes imprimir X líneas, con dos enteros separados por un espacio en cada una, las dos ciudades A_i y B_i que une la carretera que ocupará un agricultor, en el sentido de A_i a B_i. Si existen varias respuestas posibles, cualquiera será aceptada. Además, el orden en el que se imprimen las carreteras no importa.

Entrada de ejemplo

```
6 8
0 1
0 2
1 2
2 3
3 4
4 1
3 5
4 5
```

Salida de ejemplo

```
1
2 3
```

Límites

- $1 \leq N \leq 160$ $\qquad$ $1 \leq 1,000 \leq M$
- $0 \leq A, B \leq N-1$
- $A_i \neq B_i$ $\qquad$ $(A_i, B_i) \neq (A_j, B_j)$

Tiempo: 4 segundos

C
Y sin embargo se mueve

A Isidra le gusta viajar entre galaxias en su tiempo libre. Tras años de estudio, la Facultad de Fomento del Turismo proporciona un mapa que incluye las distancias entre galaxias. Quizá porque los fondos dedicados a la investigación no alcanzaban para obtener más datos, o quizá por miedo a encontrar incongruencias en los resultados y porque nadie entiende las leyes de la física, el mapa proporcionado es un poco especial, de tal forma que entre cualesquiera dos galaxias, existe una única forma de medir la distancia entre ellas.

Isidra es de esas personas que dicen que el viaje es el camino, así que cuanto más largo el camino, más lo disfruta. También es consciente de que a medida que pasa el tiempo, las galaxias siempre se van alejando entre ellas. Para hacer las cosas más fáciles, el mapa proporciona por una parte la distancia inicial entre galaxias, y por otra parte en cuánto va a aumentar esa distancia cada año que pasa. Isidra es muy preguntona, así que le gustaría saber para dos galaxias distintas, cuál es el mínimo número entero de años que deben transcurrir para que estas dos galaxias estén a una distancia mayor o igual que lo que ella quiera.

Más formalmente, se dan n galaxias y $n-1$ datos de distancias entre pares de galaxias. El dato i entre las galaxias u_i y v_i consiste en dos enteros a_i, b_i, tales que tras t años, la distancia entre u_i y v_i es $d_i = a_i + t \cdot b_i$. Una pregunta de Isidra es de la forma (u, v, w), que significa "¿cuál es el mínimo número entero de años que deben pasar para que la distancia entre u y v sea mayor o igual que w?"

Por supuesto, Isidra no se quedará satisfecha con una sola pregunta, así que prepárate para responder muchas.

Entrada

La primera línea contiene un único entero n — el número de galaxias.

Las siguientes $n-1$ líneas incluyen cada una cuatro enteros (u_i, v_i, a_i, b_i) — las dos galaxias u_i, v_i que conforman el dato i-ésimo, la distancia inicial a_i entre ellas y la distancia extra b_i que se separan cada año. Las distancias son independientes del sentido en el que se viaje entre las dos galaxias.

La siguiente línea contiene un único entero q — el número de preguntas.

A continuación vendrán q líneas, cada una con tres enteros (u_i, v_i, w_i) — las dos galaxias (u_i, v_i) $(u_i \neq v_i)$ para las que calcular el tiempo mínimo para que la distancia sea mayor o igual que w_i.

Salida

Imprime q enteros separados por espacios, las respuestas a cada pregunta, en el orden dado. Si ya están a distancia mayor o igual que la pedida en el instante inicial, se responderá con 0.

Entrada de ejemplo

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

Salida de ejemplo

```
2 1 0
```

Límites

- $2 \leq n \leq 10^5$
- $1 \leq u_i, v_i \leq n$
- $1 \leq a_i, b_i \leq 10^4$
- $1 \leq q \leq 10^5$
- $1 \leq w \leq 10^9$

Tiempo: 1 segundo

D
El problema de las 3 fuentes

Tras tantos años de competición tu amigo Marcos ha decidido dejar la programación competitiva, está harto de resolver problemas. En este radical cambio de vida ha decidido abrir un puestecito de prensa en el parque donde poder disfrutar de una vida más tranquila.

El problema es que el parque donde quiere trabajar tiene 3 fuentes que están activas durante toda la tarde. Marcos no quiere que sus clientes se mojen mientras charla con ellos cuando eligen qué periódico llevarse, por lo que está buscando el sitio perfecto para colocar su tienda.

La cantidad de agua que llega a cada punto del parque viene definida por la fuente más cercana a él. Es cierto que el resto de fuentes también mojan, pero es tan poco que puede despreciarse. Por tanto, el objetivo de Marcos es colocar su tienda de tal manera que se maximice la distancia a la fuente más cercana, es decir, maximizar el valor de `SEQUEDAD` descrito por:

$$\texttt{SEQUEDAD}(p) = \text{mín}\{d(p, F_1), d(p, F_2), d(p, F_3)\}$$

donde F_1,F_2 y F_3 representan las posiciones de las fuentes y la función $d(p, q)$ representa la distancia euclídea en el plano entre dos puntos p y q, es decir, la distancia usual en línea recta.

Sabiendo que el recinto del parque es un cuadrado perfecto de lado 100 y su esquina inferior izquierda se encuentra en el punto $(0, 0)$ ¿sabrías indicar cuál es el mayor valor de `SEQUEDAD` que podrá lograr Marcos al colocar su nueva tienda?

Entrada

La entrada comienza con un número N, la cantidad de casos de prueba que deben procesarse. Cada caso de prueba está compuesto por 3 líneas, cada una indicando un par de números (F_x, F_y), la posición de una de las fuentes. Todos estos valores se indicarán con 3 decimales, usando el punto como separador decimal.

Salida

En la salida por cada caso se indicará un número **con 3 decimales** redondeados (usando el punto como separador decimal), el valor máximo de `SEQUEDAD` que Marcos puede lograr al colocar su tienda en el parque.

Entrada de ejemplo

```
1
19.000 13.000
10.000 81.000
73.000 44.000
```

Salida de ejemplo

```
62.169
```

Límites

- $1 \leq N \leq 10^4$
- $0 \leq F_{1x}, F_{2x}, F_{3x}, F_{1y}, F_{2y}, F_{3y} \leq 100$
- $(F_{1x}, F_{1y}) \neq (F_{2x}, F_{2y})$

- $(F_{1x}, F_{1y}) \neq (F_{3x}, F_{3y})$
- $(F_{2x}, F_{2y}) \neq (F_{3x}, F_{3y})$

Notas

Para imprimir un número x con 3 cifras decimales exactas y redondeadas puedes usar:

- C++

```
std::cout << std::fixed << std::setprecision(3) << x << std::endl;
```

- Java

```
double redondeo = Math.round(x * 1000.0) / 1000.0;
String numeroFormateado = String.format("%.3f", redondeo);
System.out.println(numeroFormateado);
```

- Python

```
print(f'{x:.3f}')
```

Tiempo: 1 segundo

E
La cola del supermercado

Un supermercado tiene dos cajas para cobrar los productos que compran sus clientes y, por ese motivo, hay siempre dos colas que, sin embargo, progresan a velocidad diferente, con lo que se producen enfados entre algunos clientes. Hartos de la situación, el supermercado ha decidido implementar un algoritmo que determine en qué cola debe situarse cada cliente, habida cuenta del tiempo estimado en atenderle en caja, de modo que el tiempo total en atender a todos los clientes sea el mínimo posible.

Entrada

Existirán varios casos de prueba, representados por un número C en la primera línea. Para cada caso de prueba se indica, en una sola línea, primero el número de clientes que hay en total, N, y a continuación N números indicando el tiempo T que tarda cada uno en ser atendido.

Salida

Para cada caso de prueba hay que indicar el tiempo que tardan la primera y segunda caja, respectivamente, en atender a los clientes asignados a cada una de ellas, de tal modo que la diferencia entre ambos tiempos sea la mínima posible. En caso de no tardar lo mismo, la caja que más tiempo atenderá será la segunda.

Entrada de ejemplo

```
3
3 1 3 1
4 1 3 6 2
6 2 2 3 4 8 11
```

Salida de ejemplo

```
2 3
6 6
15 15
```

Límites

- $1 \leq C \leq 3000$
- $1 \leq T \leq 8000$
- $1 \leq N \leq 3000$
- La suma de todos los tiempos T sobre todos los casos no excederá 8000
- La suma de N sobre todos los casos no excederá 3000

Tiempo: 1 segundo

F
Follow the LIDAR

La empresa "Bailes S.A." está interesada en hacer un nuevo robot de limpieza doméstica. Como hay una competencia muy fuerte en este sector, ha decidido hacer un robot que tenga algo novedoso y se ha decidido por incorporar un Lidar (un sensor que mide distancias por láser). Aunque, como los sensores Lidar son bastante costosos, ha optado por un Lidar de haz único que le permite medir la distancia libre delante del robot. Le ha ido muy bien, ya lo ha construido y está deseando probarlo, así que se ha puesto en contacto con vosotros para hacer un prototipo de software que le permita moverlo.

Vuestro programa debe controlar el movimiento del robot en un mapa definido por una cuadrícula de celdas utilizando 3 órdenes básicas: avanzar, girar a la izquierda y girar a la derecha. Cuando se avanza sólo se mueve una única casilla. Los giros son de 90^{o}. Cada vez que se ejecute una de estas órdenes se recibirá el número de celdas libres que hay delante del robot. Cada celda de la cuadrícula solo puede estar libre u ocupada. El robot no se puede salir de la cuadrícula y, de hecho, la medición del número de celdas libres será cero si se mide desde el borde y hacia el exterior.

El robot comienza siempre en la celda situada en el límite inferior izquierdo de la cuadrícula y debe llegar a la celda en el extremo opuesto, es decir, si la cuadrícula tiene un tamaño $N \times N$ y tomamos la celda inferior izquierda como la celda cuyas coordenadas son $(1, 1)$, el destino sería la celda cuyas coordenadas son (N, N). El robot siempre empieza **orientado hacia arriba**, es decir, si su primer movimiento es un *avance* entonces se movería a la celda $(1, 2)$. Se garantiza que $N \leq 20$.

No es necesario encontrar la solución **óptima**. Es decir, se admiten soluciones donde el camino puede ser más largo del óptimo, pero eso sí, debido a las restricciones de batería, los movimientos de avance están limitados a $4 \cdot N^2$. Por supuesto, la solución no es única, el robot puede ir por muchos caminos distintos, en el fondo da igual, lo importante es que limpie.

X	X	X	X	X	X	X
X				X	(5,5)	X
X		X		X		X
X				X		X
X	(1,2)	X	X	X		X
X	(1,1)	(2,1)				X
X	X	X	X	X	X	X

Ejemplo de cuadrícula de celdas

En la figura se muestra un posible mapa de celdas. Se muestran las coordenadas de la casilla inicial y final, así como las coordenadas de un par de casillas adicionales. Las celdas con una "X" no están libres. Se muestra también una "capa" de celdas fuera de la cuadrícula, marcadas también con una "X".

Entrada

La entrada consiste en un único caso de prueba.

En la primera línea se encuentra un número N, que indica el tamaño en celdas de la cuadrícula del problema. A continuación, se podrá leer, en una línea, el número de celdas libres inicialmente delante del robot, un número entero positivo entre 0 y $N-1$.

A continuación, cada vez que el programa produzca una salida (una orden de movimiento), se escribirá en una nueva línea el número de celdas libres enfrente del robot o *END* si el robot ha llegado a su destino, es decir, si ha llegado a la celda de coordenadas (N, N).

Si el robot está colocado en el borde de la cuadrícula y mirando hacia el exterior de la misma el número de celdas libres será 0.

Salida

Las órdenes se codifican con tres letras, "A" para avanzar, "D" para girar a la derecha e "I" para girar a la izquierda. Se deben escribir sin las comillas y se debe escribir un salto de línea después de la orden. Inmediatamente después de escribir la orden se podrá leer el número de casillas libres.

Si se indica la orden de avanzar y el robot no puede hacerlo porque no hay ninguna casilla libre, se producirá automáticamente el resultado incorrecto (*Wrong Answer*).

Se recuerda que es conveniente hacer *flush* del canal de salida. En C/C++ se hace mediante: `fflush(stdout);`, en Java con `System.out.fflush();` y en Python con `print([...], flush=True);`

Ejemplo de entrada y salida

La secuencia en que se produce la entrada/salida para los primeros pasos es: se escribe en la entrada el 5 (N) y el 4 (celdas libres), el programa entonces escribe A (avance), y justo a continuación se tendrá el 3 en la entrada.

```
5
4
3
2
1
0
2
1
0
2
1
0
2
1
0
2
1
0
4
3
2
1
0
4
3
2
1
END
```

```
A
A
A
A
D
A
A
D
A
A
D
A
A
I
A
A
I
A
A
A
A
I
A
A
A
A
```

Notas

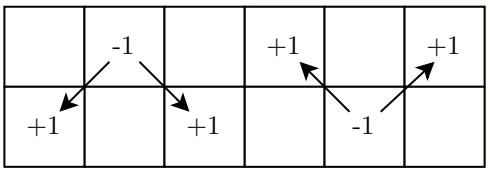
- La salida de ejemplo se corresponde con el diagrama de la figura.
- **Importante**: La solución que se muestra como salida se ha hecho manualmente conociendo el mapa de celdas. No se puede utilizar como base para desarrollar el algoritmo.
- El algoritmo tiene un número límite de movimientos de avances, pero no hay límite para los giros.
- Naturalmente, sí existe el límite de tiempo habitual y se superpone al límite de avances.
- Un ejemplo sobre giros: si el robot está apuntando hacia arriba (coordenadas "y" crecientes) y gira a la derecha se queda apuntando a la derecha (coordenadas "x" crecientes) y respectivamente si gira a la izquierda se queda apuntando a la izquierda (coordenadas "x" decrecientes).

Tiempo: 1 segundo

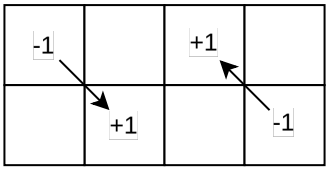
G
El pensador cuadriculado

Furtis está cansado de programar. Ahora dedica su tiempo libre a pensar problemas, a poder ser sin tener que programar sus soluciones, porque le da mucha pereza. Se pasa el día inventándose operaciones, haciéndose preguntas e intentando sacar una solución satisfactoria. Parece que es la única forma de explicar que se le haya ocurrido este problema...

Furtis define un juego en una cuadrícula $2 \times M$ (2 filas y M columnas). En cada casilla se empieza con el valor 0. En una operación se puede elegir una casilla, restarle 1 a ella y sumar 1 a sus dos vecinas diagonales. Solo se puede hacer si las dos vecinas quedan dentro de la cuadrícula.



Ejemplo de dos tipos de operaciones permitidas



Ejemplo de dos tipos de operaciones **no** permitidas

Más formalmente:

- Si se elige la casilla $a_{1,x}$ y las casillas $a_{2,x-1}$ y $a_{2,x+1}$ están dentro de la cuadrícula, se asignan:
  - $a_{1,x} \leftarrow (a_{1,x} - 1)$
  - $a_{2,x-1} \leftarrow (a_{2,x-1} + 1)$
  - $a_{2,x+1} \leftarrow (a_{2,x+1} + 1)$
- Si se elige la casilla $a_{2,x}$ y las casillas $a_{1,x-1}$ y $a_{1,x+1}$ están dentro de la cuadrícula, se asignan:
  - $a_{2,x} \leftarrow (a_{2,x} - 1)$
  - $a_{1,x-1} \leftarrow (a_{1,x-1} + 1)$
  - $a_{1,x+1} \leftarrow (a_{1,x+1} + 1)$

Furtis quiere saber si es posible hacer que todas las casillas acaben a la vez con un valor entero V dado y si lo es, el mínimo número de operaciones necesarias para ello.

Naturalmente, no eres Furtis y estás en un concurso de programación competitiva, así que te va a tocar programar la solución...

Entrada

La primera línea contiene un único entero t — el número de casos de prueba.

Las siguientes t líneas incluyen cada una dos enteros (M, V) — el número de columnas para una cuadrícula $2 \times M$ y el valor V que se desea obtener en todas las casillas a la vez.

Salida

Para cada caso de prueba, imprime en una nueva línea un único entero – el número mínimo de operaciones necesarias para que todas las casillas tengan el valor V a la vez. Si es imposible para ese caso, imprime -1.

Entrada de ejemplo

```
2
3 1
6 3
```

Salida de ejemplo

```
-1
36
```

Límites

- $1 \leq t \leq 2 \cdot 10^5$
- $1 \leq M \leq 10^9$
- $1 \leq V \leq 10^9$

Tiempo: 1 segundo

H
Mini Tetris

El Tetris es probablemente el videojuego más famoso del mundo. Hace unos meses se consiguió un logro que parecía imposible, un chaval de 13 años apodado "BlueScuti" consiguió llegar al *kill screen* del juego, un momento donde la memoria del juego original falla y se asume que has llegado al "final". Realmente esto pasa en la versión de NES del juego que se considera la más real del Tetris, los ordenadores actuales pueden sobradamente con el juego.

Tú vas a jugar a un juego parecido, debes crear una torre de bloques 2×1 muy parecidos a los del Tetris, que se irán colocando en la típica cuadrícula del juego. Al igual que pasa con el Tetris, estos irán cayendo desde arriba y se posarán al contactar (en su base) con el suelo o con otro bloque.

En algunos puntos del mismo habrá monedas de distintos valores que conformarán la puntuación del juego: en el caso de que un bloque esté encima de una moneda en el momento de acabar el juego, el jugador obtendrá una puntuación igual a su valor.

Los bloques se pueden colocar en vertical u horizontal. Pero, a excepción del primer bloque, es necesario que estos siempre se coloquen encima del último colocado, es decir, al menos una casilla del nuevo bloque debe estar encima de una casilla ocupada por el bloque anterior. En otras palabras, deben conformar una "torre". Además, la anchura de la zona de juego estará delimitada, no se podrán colocar bloques que sobresalgan este límite. La partida terminará cuando se supere la altura máxima de la zona de juego.

¿Eres capaz de determinar la mayor puntuación posible en el Mini Tetris?

Entrada

La entrada conformará un solo caso de prueba. En la primera línea aparecerán dos valores W, L, que serán el ancho y el alto del tablero respectivamente.

Posteriormente aparecerá un número N, el número de monedas que hay en el terreno de juego. Finalmente N líneas con tres valores X_i, Y_i, V_i, la posición en la que estará la moneda descrita por sus coordenadas (X_i, Y_i), y su valor V_i. La posición se calcula asumiendo que el cuadro inferior izquierdo es el $(0, 0)$ y los valores crecen hacia arriba y hacia la derecha.

Salida

Por cada caso se debe imprimir un único valor D, el máximo de puntos que se pueden obtener en la partida, hasta alcanzar el límite superior de la zona de juego.

Entrada de ejemplo

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

Salida de ejemplo

```
30
```

Entrada de ejemplo

```
5 6
6
0 0 5
0 1 5
0 2 30
0 3 5
0 4 5
4 2 60
```

Salida de ejemplo

```
60
```

Figura 1: Entrada de ejemplo 1

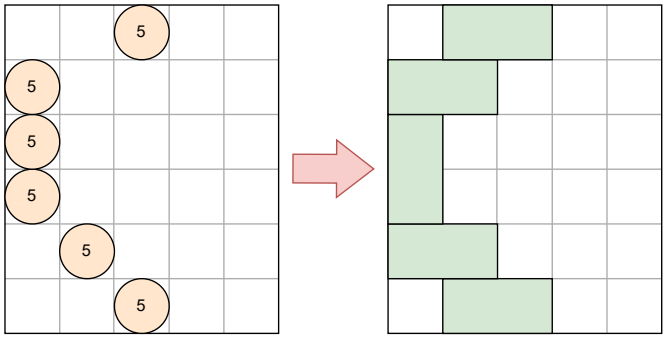
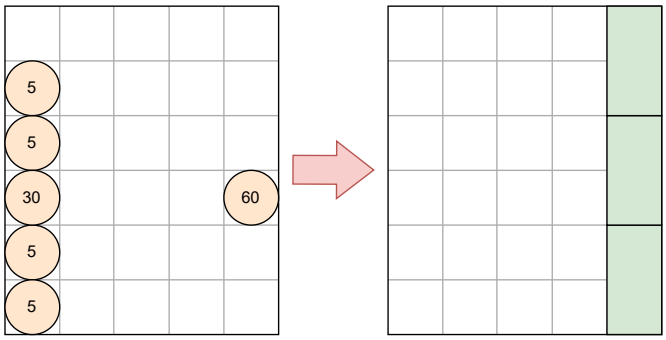


Figura 2: Entrada de ejemplo 2



Límites

- $2 \leq W, L \leq 1,000$
- $0 \leq N \leq W \cdot L$
- $0 \leq V_i \leq 100$
- $0 \leq X_i < W$
- $0 \leq Y_i < L$
- Dos monedas nunca aparecerán en el mismo sitio.

Tiempo: 1 segundo

I
El calcetín mágico

En un mundo fantástico existe un calcetín mágico especial, y en este mundo viven dos traviesos elfos, Dobby y Winky, que adoran crear y jugar con objetos mágicos.

Llenos de creatividad, Dobby y Winky trabajan juntos para dar vida a nuevos objetos mágicos. Inspirados por su fascinación, crean un reluciente puente de luz, conectando cada nuevo objeto a otro. Esta conexión puede hacerse con el calcetín mágico especial o con cualquier otro objeto mágico que ya esté conectado a través de los puentes de luz existentes al calcetín mágico. ¡Es como si tejieran una red mágica con sus propias manos! De esta manera, dan forma a su "árbol mágico", un reflejo de su imaginación y espíritu.

Luego de crear su obra maestra, Dobby y Winky se embarcan en la parte competitiva del juego. En lugar de agregar más objetos, se turnan para eliminar puentes de luz. En cada turno, un jugador elimina un único puente de luz, haciendo que los objetos mágicos, y los puentes de luz que los conectan, que ya no estén vinculados al calcetín mágico se desvanezcan en el aire. A quien le toque jugar cuando solo quede el calcetín mágico pierde, al no poder eliminar ningún puente.

Dobby y Winky han repetido este juego numerosas veces, creando árboles mágicos y luego desarmándolos, por lo que se han vuelto expertos pero ya están un poco cansados de jugar. Para los siguientes árboles mágicos que tenían pensados, quieren que les digas si ganará el primer o el segundo jugador, teniendo en cuenta que los dos juegan de forma óptima.

Entrada

La entrada consiste en varios casos de prueba. La primera línea contiene un único entero t ($1 \leq t \leq 10^5$) — el número de casos de prueba. Cada caso de prueba está formado por dos líneas.

La primera línea de cada caso contiene un único entero n ($2 \leq n \leq 10^6$) — el número de objetos en el árbol mágico.

Cada objeto es representado por un número de 1 a n, siendo el calcetín mágico el representado por el número 1.

La segunda línea de cada caso contiene $n-1$ enteros $o_2, o_3, \ldots, o_n$ ($1 \leq o_i < i$) — el objeto al que fue conectado cada objeto no especial en el árbol.

Se garantiza que la suma de n sobre todos los casos de prueba no excederá 10^6.

Salida

Para cada caso de prueba, imprime 1 si el primer jugador tiene estrategia ganadora y 2 si el segundo jugador tiene estrategia ganadora.

Entrada de ejemplo

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

Salida de ejemplo

```
1
2
1
2
```

Ejemplo 1

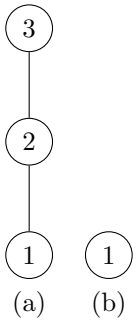


Figura 3: (a) Árbol mágico creado por Dobby y Winky. (b) Árbol mágico después de que el primer jugador elimine el puente entre el objeto 1 y el objeto 2.

El primer jugador tiene una estrategia ganadora. En su primer turno, elimina el puente que conecta el objeto 2 con el objeto 1. Esto hace que los objetos 2 y 3, y el puente que los conecta desaparezcan (ver Fig. 3 (b)). En el turno del segundo jugador, solo queda el objeto 1, que no está conectado a ningún otro objeto. El segundo jugador no tiene jugadas válidas y pierde la partida.

Ejemplo 2

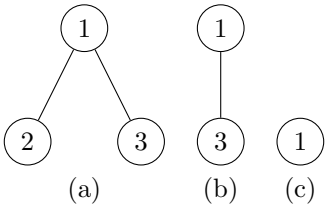


Figura 4: (a) Árbol mágico creado por Dobby y Winky. (b) Árbol mágico después de que el primer jugador elimine el puente entre el objeto 1 y el objeto 2 (c) Árbol mágico después de que el segundo jugador elimine el puente entre el objeto 1 y el objeto 3.

Independientemente de la jugada inicial del primer jugador, el segundo jugador tiene una estrategia ganadora. Si el primer jugador elimina el puente conectando el objeto 2 con el objeto 1, el objeto 2 desaparece. En el turno del segundo jugador, solo quedan los objetos 1 y 3 conectados por un puente (ver Fig. 4(b)). El segundo jugador puede eliminar el único puente produciendo que el objeto 3 desaparezca. Luego, cuando vuelva a ser el turno del primer jugador solo quedará el objeto 1 (ver Fig. 4 (c)), no podrá jugar y, por lo tanto, perderá la partida. Con un análisis análogo se puede mostrar que si el primer jugador elimina en su primer turno el puente conectando el objeto 3 con el 1, en el segundo turno del primer jugador nuevamente solo estará el objeto 1 y, por lo tanto, también el primer jugador perderá la partida en este caso.

Límites

- La suma de n sobre todos los casos de prueba no excederá 1,000,000.

Tiempo: 1 segundo

J
Walcu: Explotando globos

En preparación para la fiesta, te has puesto a llenar globos de agua con tanto entusiasmo que no te has dado cuenta de que casi ninguno tiene la misma cantidad de agua. En vez de empezar desde cero, has decidido seguir con tu método frenético, llenando los globos con una cantidad arbitraria de agua, para luego descartar (y explotar) aquellos que sean demasiado grandes o demasiado pequeños.

Por ejemplo, si has hinchado globos con unas cantidades de 16, 10, 9, 12 y 11 unidades de agua, y decides descartar el globo más grande y los dos más pequeños, descartarás los globos con cantidades 9, 10 y 16, lo que te hará malgastar 35 unidades de agua.

Entrada

Existirán varios casos de prueba, representados por un número C en la primera línea. Cada caso de prueba contiene los valores E, A y B. El número E de eventos que tendrán lugar, y los valores A y B, representando el número de globos más pequeños y más grandes a explotar, respectivamente ($0 \leq A+B < E$). A continuación aparecen E líneas, describiendo cada una un evento en orden cronológico con dos posibles formatos:

- H x: Indicando que se hincha un nuevo globo con una cantidad x de agua.
- P: Indicando que te haces la pregunta de cuánta agua se desperdiciaría si decidieras parar de hinchar globos en ese momento. Recuerda que sólo te haces la pregunta, no llegas a explotar ningún globo.

Todos los casos de prueba empiezan por al menos $A + B$ eventos de tipo H, y contienen al menos un evento de tipo P.

Salida

Para cada evento de tipo P, se escribirá en una línea independiente la suma del agua que se gastaría si se explotaran los globos en ese momento. Como la suma puede llegar a ser muy grande, se escribirá el resultado módulo 1.000.000.009.

Al final de cada caso de prueba se escribirán tres guiones '---'.

Entrada de ejemplo

```
2
8 1 1
H 1
H 2
H 3
P
H 4
P
H 5
P
8 2 3
H 1
H 1
H 1
H 1
H 2
P
H 100
P
```

Salida de ejemplo

```
4
5
6
---
6
105
---
```

Límites

- $1 \leq C \leq 10{,}000$
- $1 \leq x \leq 10^{18}$
- $A + B + 1 \leq E$
- El número total de eventos, E, es menor o igual que $2 \cdot 10^5$

Tiempo: 1 segundo

K
La conjetura de Go*d*dbach

La conjetura de Goldbach establece que cualquier número par mayor que 2 puede escribirse como la suma de dos números primos. Por ejemplo:

$$\begin{array}{rcl} 4 & = & 2 + 2 \\ 6 & = & 3 + 3 \\ 8 & = & 3 + 5 \end{array}$$

...

En su lugar, los jueces de Ada Byron proponen la conjetura de Go*d*dbach: "*cualquier número impar mayor que 1, que no sea primo, puede describirse como la suma de dos números primos*". Los primeros números impares mayores que 1, que no son primos son: 9, 15, 21, ... y:

$$\begin{array}{rcl} 9 & = & 2 + 7 \\ 15 & = & 2 + 13 \\ 21 & = & 2 + 19 \end{array}$$

...

Desafortunadamente, la conjetura de Go*d*dbach no se cumple, y el 27 es el primer número impar que no siendo primo, no se puede expresar como la suma de dos números primos. Pero lo que sí puede hacerse es elaborar la *serie de Goddbach*, $\mathcal{G}$, con todos los números impares que no son primos y sí se pueden escribir como la suma de dos números primos. En esta serie, $\mathcal{G}_1 = 9$, $\mathcal{G}_2 = 15$, $\mathcal{G}_3 = 21$, ...

Dado un índice i, se pide calcular el número que hay en esa posición en la serie $\mathcal{G}$ de Go*d*dbach, $\mathcal{G}_i$.

Entrada

Existirán varios casos de prueba, representados por un número C que se indica en la primera línea de la entrada. Cada caso de prueba consiste exactamente en una línea, cada una de las cuales contiene un único índice i, contado a partir de la posición 1.

Salida

La salida debe consistir en tantas líneas como casos de prueba haya definidos en la entrada, C, de modo que cada línea debe mostrar el número de la serie de Go*d*dbach que ocupa la posición especificada en el caso de prueba correspondiente.

Entrada de ejemplo

```
3
1
3
103
```

Salida de ejemplo

```
9
21
759
```

Límites

- $1 \leq C \leq 500{,}000$
- $1 \leq i \leq 500{,}000$

Tiempo: 1 segundo

L
Next Digital: Espacios de Coworking

En Next Digital somos 105 compañeros ("nexters") y gracias a que somos una empresa 100 % remota estamos repartidos por toda la geografía española, o haciendo el nómada por el extranjero. La media de edad es de 27 años, y quieras que no, nos gusta vernos, tomar una cervecita bien fría y comer en algún asturiano. Desde que hemos superado el centenar de nexters es más difícil coincidir, por eso nos hemos planteado tener múltiples espacios de coworking contratados distribuidos de forma que todos los nexters tengamos al menos un espacio a menos de 1 hora de trayecto en uno o más transportes, de forma directa o pasando por otras ciudades y haciendo transbordo. El día que hacemos por vernos no queremos perder tiempo viajando, y como somos remotos, no tiene sentido comprar oficinas que la mayoría del tiempo estarán vacías.

Hemos pedido presupuesto a diferentes empresas de coworking que nos ofrecen varios espacios de trabajo flexibles a demanda. Ya hemos descartado aquellas empresas que o bien no tenían el café incluido o no estaban en ciudades con algún nexter viviendo allí. Ahora tenemos que descartar aquellas que no tengan espacios de trabajo lo suficientemente cerca de todos y cada uno de los nexters. No queremos contratar múltiples empresas, solo queremos sedes suficientemente distribuidas.

Para llegar a una conclusión hemos repartido el trabajo entre varios equipos. El equipo Hormigas ha elaborado un listado con las ciudades donde hay uno o más nexters, el equipo Vacas ha obtenido mediante una API los tiempos de trayecto que hay entre estas ciudades, el equipo Koalas ha tratado toda la información generada previamente y obtenido el trayecto mínimo entre cada ciudad conectada de manera directa o a través de ciudades intermedias sin nexters viviendo. Estaba todo listo para que los Mapaches verificaran el listado de espacios de coworking, pero las constantes incidencias en una de sus aplicaciones han dejado esta tarea en su backlog.

Entrada

La entrada comienza con los números N, E, X y T en una sola línea. Siendo:

- N $(0 < N \leq 200)$ un entero que indica en cuántas ciudades diferentes están los 105 nexters,
- E $(0 \leq E \leq 20,000)$ el número de carreteras que conectan dos ciudades distintas,
- X $(0 \leq X \leq 50)$ el número de empresas de coworking filtradas por el equipo Hormigas, y
- T $(1 < T \leq 200)$ los minutos máximos que un nexter está dispuesto a ir en transporte público en total, si tarda T minutos o más a la ida o la vuelta, ya no querrá ir. No hay tiempo de espera en los transbordos.

Seguidamente aparecen E filas, con tres números enteros u, v y c. La u y la v $(1 \leq u, v \leq N)$ son índices que representan cada ciudad con uno o más nexters. La terna de los tres números nos indica que entre las ciudades u y v se tardan c $(1 \leq c < T)$ minutos con el medio de transporte sin transbordos más rápido tanto en la ida como en la vuelta.

Por último, aparecerán X filas, una por cada empresa de coworking. Cada fila comenzará por el número o $(1 \leq o \leq N)$ indicando en cuántas ciudades nos ofrecen una oficina, seguido por o números donde cada número es el índice de una ciudad.

Salida

Se pide mostrar 1 fila por cada caso de prueba, con la lista de índices de las empresas separadas por espacios en orden ascendente que tengan suficientes espacios de coworking distribuidos para que los nexters tengan al menos una sede suya a menos de T minutos de trayecto total a la ida (o a la vuelta). Si ninguna de las empresas cumplen esta restricción, se escribirá "NO HAY EMPRESAS". El índice de

una empresa será la posición que ocupa en el listado de las X empresas empezando por 1 y acabando por X.

El programa terminará cuando reciba la entrada "$N = E = X = T = 0$" sin mostrar nada en la salida.

Entrada de ejemplo

```
4 4 4 60
1 2 20
2 3 20
4 3 50
1 4 40
1 2
1 3
1 4
2 1 2
5 5 4 60
1 2 20
2 3 20
4 3 50
1 4 40
5 1 55
1 2
1 3
1 4
2 2 3
0 0 0 0
```

Salida de ejemplo

```
2 4
NO HAY EMPRESAS
```

Límites

- $0 < N \leq 200$
- $0 \leq E \leq 20,000$
- $0 \leq X \leq 50$
- $1 < T \leq 200$
- $1 \leq u, v, o \leq N$
- $1 \leq c < T$
- Se garantizan en el conjunto de casos de prueba las siguientes condiciones:
  - La suma de N sobre todos los casos de prueba no excederá 200
  - La suma de E sobre todos los casos de prueba no excederá 20,000
  - La suma de X sobre todos los casos de prueba no excederá 50