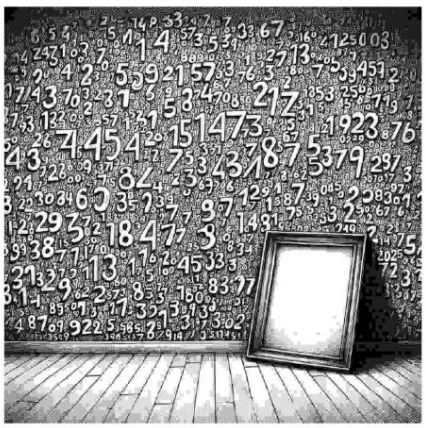
Problema A. AN2DL

Nombre del archivo fuente: AN2DL.c, AN2DL.cpp, AN2DL.java, AN2DL.py Estándar

Aporte:

Producción: Estándar



Mientras paseabas por el Edificio 21, te topaste con una pared completamente cubierta de números, dispuestos en una tabla de n filas y m columnas. Pronto te diste cuenta de que había un marco apoyado en la pared, lo suficientemente grande como para enmarcar r filas y s columnas de la tabla. Y junto al marco, encontraste un lápiz y una hoja de papel que contenía una tabla vacía.

Estás triste porque la mesa en la hoja de papel está vacía, así que decides jugar con el marco para llenarla.

Apoyaste el marco contra la pared de modo que el número de la i-ésima fila y la j-ésima columna estuviera en la esquina superior izquierda, y los bordes del marco fueran paralelos a los bordes de la pared. Considerando los números dentro del marco, y como te gustan los números grandes, decidiste escribir el mayor en la i-ésima fila y la j-ésima columna de la tabla, en la hoja de papel.

Repitió el proceso para cada posición posible del marco en la pared (de modo que el marco esté completamente en la pared y que haya exactamente r × s números dentro de él), asegurándose de que los bordes del marco sean paralelos a los bordes de la pared.

Cuando terminaste, la mesa en el trozo de papel era aún más bonita que la de la pared.
¿Qué números hay en la tabla de la hoja de papel?

Entrada

La primera línea contiene dos números enteros n y m (1 ≤ n, m ≤ 4000), el número de filas y columnas de la tabla en la pared.

Cada una de las siguientes n líneas contiene m números enteros ai,j (|ai,j | ≤ 10000), donde a_i,j es el número en la i-ésima fila y la j-ésima columna de la mesa en la pared.

La última línea contiene dos números enteros r y s (1 ≤ r ≤ n, 1 ≤ s ≤ m), el tamaño del marco.

Salida

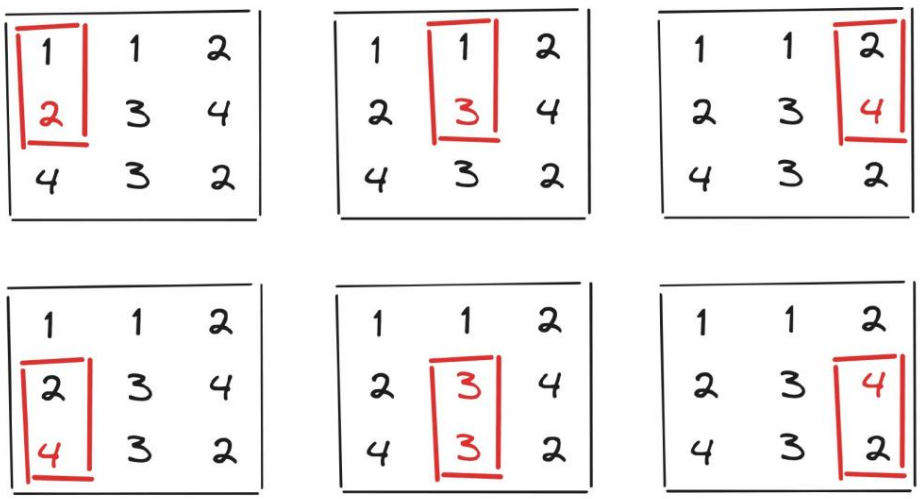
Salida los números escritos en la tabla en la hoja de papel.

Ejemplo

Entrada	Salida 4
3 3 1 1 2 2 3 4 4 3 2 3 3	
3 3 1 1 2 2 3 4 4 3 2 2 1	2 3 4 4 3 4
5 5 -1 -3 -4 -2 4 -8 -7 -9 -10 11 5 2 -8 -2 1 13 -3 -2 -6 -9 11 6 2 7 4 2 3	-1 -2 11 5 2 11 13 2 1 13 7 7

Explicación del primer ejemplo: El marco es lo suficientemente grande como para que quepa toda la mesa en la pared. El número más grande dentro del marco es el 4, por lo que ese es el único número escrito en la mesa del papel.

Aclaración del segundo ejemplo: En la imagen a continuación se muestran todas las posibles posiciones del marco. El número mayor para cada posición está escrito en rojo.

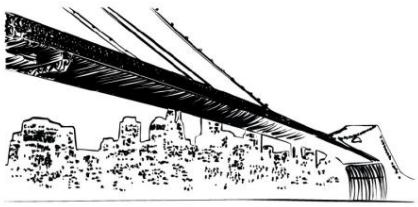


Problema B. Puente

Nombre del archivo fuente: Bridge.c, Bridge.cpp, Bridge.java, Bridge.py Estándar

Aporte:

Producción: Estándar



Cuando Leonhard Euler resolvió el famoso problema del puente de Königsberg, no tenía idea de que había descubierto una área completamente nueva de las matemáticas: ¡la teoría de grafos!

Desafortunadamente, el problema del puente de Königsberg era demasiado fácil para los programadores de esa época, por lo que Euler ideó otro problema: ¡el problema del puente de Zagreb!

Los puentes de Zagreb forman un grafo con n nodos y m aristas, donde las aristas representan los puentes y los nodos las islas ribereñas. El grafo es conexo; es decir, es posible ir de un nodo a otro recorriendo las aristas. Euler preguntó: ¿cuántas aristas hay que, tras eliminarlas, el grafo se vuelve inconexo?

De nuevo, Euler desconocía que este problema también es famoso hoy en día (esos malditos blogs de Codeforces). Así que el autor de este problema decidió plantear uno aún más difícil: ¿cuántas aristas hay tales que, tras eliminar los nodos que conecta, los n − 2 nodos restantes quedan desconectados?

Entrada

La primera línea contiene los números enteros n y m (4 ≤ n ≤ 100000, n − 1 ≤ m ≤ 300000): el número de nodos y aristas respectivamente.

Cada una de las siguientes m líneas contiene números enteros ai y bi (1 ≤ ai , bi ≤ n) - esto significa que los nodos ai y bi son conectados mediante una arista.

No hay bucles ni bordes múltiples.

Salida En

una sola línea, muestre el número de aristas con la propiedad dada.

Ejemplo

Entrada	Salida 1
4 5 1 2 2 3 3 4 4 1 1 3	
6 7 1 2 2 4 2 6 3 5 6 1 4 3 2 5	4

Explicación del
primer ejemplo: Al eliminar la arista (1, 3) y

los nodos correspondientes 1 y 1, el grafo restante tiene dos componentes conexos: el nodo 2 y el nodo 4. En otras palabras, no es conexo. Es fácil comprobar que es la única arista con esta propiedad.

Aclaración del segundo ejemplo:

Las aristas con la propiedad dada son (1, 2), (2, 4), (2, 6) y (2, 5).

Problema C. Campo a través

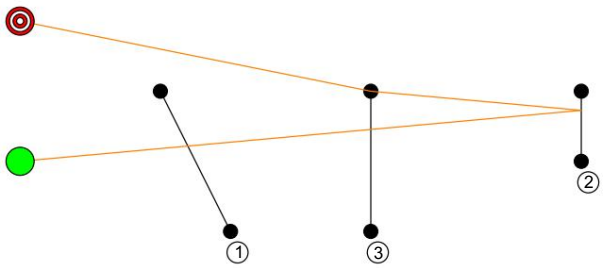
Nombre del archivo fuente: Cross.c, Cross.cpp, Cross.java, Cross.py Estándar

Aporte:

Producción: Estándar

El cross country es un deporte en el que los participantes corren una carrera en un circuito al aire libre sobre terreno natural. Para registrar su progreso, los organizadores instalaron puntos de control RFID que se extienden a lo largo de una línea transversal a lo largo del recorrido.

Un concursante ha finalizado la carrera una vez que ha pasado por todos los puntos de control en orden del 1 al n.
Cruzar un punto de control fuera de orden no supone ninguna ventaja ni penalización para un corredor, ya que simplemente debe volver a cruzarlo más tarde en el momento oportuno. Así, por ejemplo, un corredor puede optar por cruzar un punto de control una vez y luego volver a cruzarlo inmediatamente en otra dirección si esto le permite llegar más rápido.



Ruta de carrera óptima para el recorrido dado en el ejemplo de entrada 3.

Tu objetivo es encontrar la distancia más corta que hay que correr para terminar la carrera, para que podamos usarla como la distancia oficial del recorrido.

Aporte

- Una línea que contiene el número de puntos de control, n (1 ≤ n ≤ 16).
- Una línea que contiene las coordenadas de inicio de la carrera, xs e ys (−106 ≤ x, y ≤ 106).
- n líneas más, la i-ésima de las cuales contiene las dos coordenadas enteras de los puntos finales del i-ésimo punto de control, xaiyaxbyb (−106 ≤ x, y ≤ 106).
- Una línea que contiene la coordenada final de la carrera, xt e yt (−106 ≤ x, y ≤ 106).

Todos los puntos de control tienen una longitud distinta de cero; sin embargo, pueden superponerse entre sí o con los puntos de inicio y finalización.

Salida Salida
la distancia más corta que puedes correr para visitar todos los puntos de control en el orden correcto, independientemente de si tocas algunos de los puntos de control varias veces o en el orden incorrecto en el camino.

La salida debe tener una precisión de un error absoluto o relativo de como máximo 10−6 .

Ejemplo

Entrada	Salida
2 0 1 10 0 10 2 20 2 20 0 30 1	30
4 5 5 10 1 8 -1 12 3 13 0 18 3 17 0 20 1 22 -1 25 5	22.80624847
3 0 0 3 -1 2 1 8 0 8 1 5 -1 5 1 0 2	16.144380531

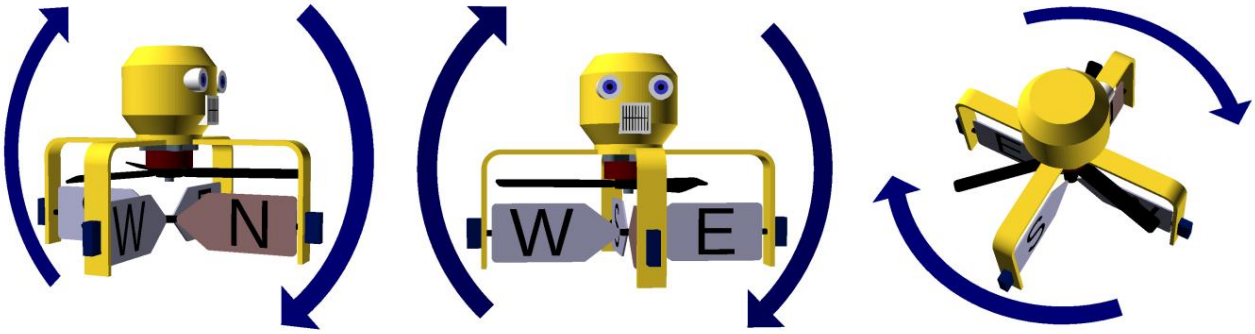
Problema D. Control de drones

Nombre del archivo fuente: Drone.c, Drone.cpp, Drone.java, Drone.py Estándar

Aporte:

Producción: Estándar

Estás diseñando un controlador para una aeronave interesante llamada Monocóptero. Tiene una sola hélice, pero el flujo de aire saliente está determinado por cuatro flaps que controlan tres ángulos de Euler (cabeceo, alabeo y guiñada) que ayudan a mantener la orientación solicitada de la aeronave. Cada flap puede adoptar cualquier ángulo solicitado por el controlador de vuelo, y los efectos de los flaps en ciertos ángulos deberían traducirse en ejercer las fuerzas solicitadas sobre el cabeceo, alabeo y guiñada.



Paso Rollo Guiñada

Inclinación, balanceo y guiñada en un solo helicóptero

Define los ángulos de las aletas como n, e, s y w (para “norte”, “este”, “sur” y “oeste”, respectivamente).
Las fuerzas en las direcciones de cabeceo, balanceo y guiñada se definen mediante las siguientes ecuaciones:

$$p = e - w$$
$$r = n - s$$
$$y = n + e + s + w$$

Como hay cuatro variables y tres restricciones, usted decidió que, desde la perspectiva de la aerodinámica, tiene sentido hacer que el máximo de los ángulos de los flaps sea lo más pequeño posible, es decir, también desea minimizar max{|n|, |e|, |s|, |w|}.

Encuentre los mejores parámetros para enviar al monocopter para lograr el cabeceo, guiñada y balanceo deseados.

Aporte

- Una línea que contiene el número q $1 \le q \le 104$, el número de solicitudes a seguir.
- i líneas más, cada una con tres números reales pi , ri , yi $(-1 \le pi ,$, yi $\le +1)$.

Salida Salida

q líneas. En la i-ésima línea, genera la solución para la i-ésima solicitud, cuatro números ni separados por espacios. , yo , si , ,

Su respuesta se considerará correcta si los valores de cabeceo, alabeo y guiñada resultantes difieren como máximo en 10^{-6} de los solicitados, y el máximo de los valores absolutos de las salidas de los flaps no excede el valor real en más de 10^{-6} .

Ejemplo

Entrada	Salida
8 0 0 0 1 0 0 0 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 1 1	0.0 0.0 0.0 0.0 0.0 0.5 0.0 -0.5 0.5 0.0 -0.5 0.0 0,25 0,25 0,25 0,25 0.5 0.5 -0.5 -0.5 0.5 0.5 0.5 -0.5 0.5 0.5 -0.5 0.5 0,75 0,75 -0,25 -0,25

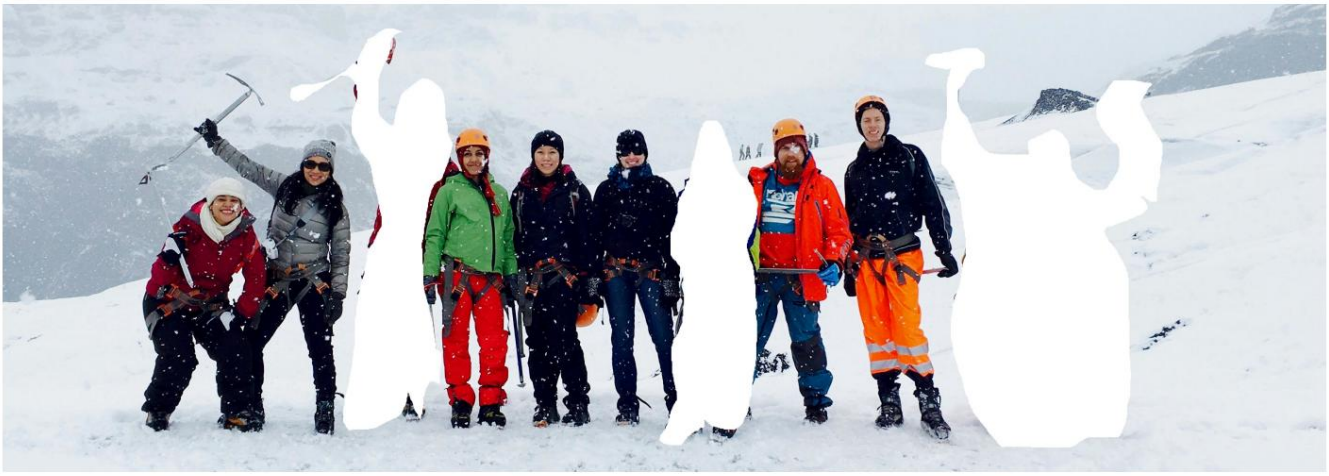
Problema E. Clasificación por erradicación

Nombre del archivo fuente:	Eradicación.c, Eradicación.cpp, Eradicación.java, Eradicación.py Estándar
Aporte:	
Producción:	Estándar

¡Los miembros de la sociedad de escalada recreativa No-Weather-too-Extreme completaron su primera cumbre exitosa hace siete años, tal como lo hicieron hoy!

En ese momento, tomamos una foto de todos los miembros de pie juntos en una fila. Sin embargo, la foto se ve desordenada, ya que los escaladores no estaban ordenados por altura y no tenemos forma de reordenarlos.

Necesitaremos eliminar algunos de los escaladores de la imagen.



Esta imagen de 7 (antes 11) escaladores fue editada para resolver el ejemplo de entrada 3.

Una solución óptima minimiza el tamaño y la cantidad de huecos visibles en la foto. Definimos el coste como la suma de los cuadrados de las longitudes de los huecos que quedan en la foto editada.

Por ejemplo, si se eliminan dos escaladores individuales de la foto y un par de escaladores adyacentes, el coste total es de 2 1.

+ 12 + 22 = 6.

Encuentra el costo mínimo posible que puedes alcanzar eliminando escaladores.

Aporte

- El número de personas en la foto n 1 ≤ n ≤ 106 .
- n números enteros que representan las alturas de las personas en la foto, h1 . . . hn 0 ≤ h ≤ 106 .

Salida

Salida el costo mínimo logrado al eliminar escaladores de la foto, de modo que los escaladores restantes en la foto formen una secuencia no decreciente.

Ejemplo

Entrada	Salida 1
7 1 2 3 0 5 6 7	
9 4 5 6 4 2 3 6 6 6	8
11 3 6 12 7 7 7 6 8 10 5 5	6

Problema F. Encontrar proteínas sospechosas

Nombre del archivo fuente: Finding.c, Finding.cpp, Finding.java, Finding.py Estándar
Aporte:
Producción: Estándar

La pequeña Claire estudia las proteínas, que son secuencias de aminoácidos. Hay 20 aminoácidos a partir de los cuales se construyen las proteínas. Si bien todos los aminoácidos tienen nombres propios, como alanina o glicina, suelen denotarse con una sola letra, de modo que las proteínas pueden verse como secuencias de diferentes longitudes, como DTASDAAAAAALTAABAAAAAKLTABBAAAAAAATAA, TIFLQQQQQQQQQQ o incluso quizás RICKRQLL.

Comparar dos proteínas puede ser difícil, ya que pueden contener sitios activos, que determinan su función en la célula, y partes menos importantes de la secuencia. Avances recientes en redes neuronales artificiales permitieron entrenar una red que, dada una proteína, genera una secuencia de l números, donde cada número corresponde aproximadamente a una característica de la proteína que se correlaciona con sus posibles funciones en la célula.
A esta secuencia se le denomina incrustación.

Claire está particularmente interesada en las proteínas sospechosas, aquellas que son realmente diferentes de las demás. Para ello, considera la llamada distancia de Manhattan entre las incrustaciones de proteínas. Para dos incrustaciones de proteínas p y q de longitud l, la distancia D(p, q) se calcula de la siguiente manera:

$$D(p, q) = \sum_{i=1}^{l} |pi - qi|,$$

donde pi es el i-ésimo elemento de la incrustación p.

Claire quiere encontrar k proteínas sospechosas en la lista dada de n proteínas. Como base para sus estudios, Claire quiere utilizar el siguiente algoritmo voraz:

proteína p (1) que está más distante de la primera proteína en la lista. • Encuentra una

• La segunda proteína, p (2), se elige como la proteína más distante de p (1) .

• El tercero, p (3), se elige de modo que min{D p (1), p(3) , D p (2), p(3) } sea el máximo posible. Que es decir, debe estar lejos de ambas proteínas elegidas previamente.

• Todas las proteínas subsiguientes $p^{(i)}$, $4 \le i \le k$, se eligen de manera similar: el mínimo de las distancias A todas las proteínas previamente elegidas se les debe dar el máximo posible.

Téngase en cuenta que, en caso de empate, se debe elegir la primera proteína coincidente de la lista.

La implementación de Claire funciona bien para números pequeños n y k, pero se vuelve demasiado lenta a medida que aumentan. Debes encontrar una manera de optimizar esto.

Entrada

La primera línea contiene tres números n ($3 \le n \le 104$), l ($1 \le l \le 100$) y k ($2 \le k \le \min\{n, 256\}$): la cantidad total de proteínas, la longitud de cada incrustación de proteína y la cantidad de proteínas a elegir.

Cada una de las siguientes n líneas comienza con un identificador de proteína, que consiste en una secuencia de al menos una y como máximo diez letras mayúsculas y/o números. A continuación, separados por espacios, aparecen l números enteros de un solo dígito v1...l ($0 \le v \le 9$), que definen la inserción de la proteína. Todos los identificadores de proteína serán diferentes.

Salida Salida

los identificadores de las k proteínas elegidas, una por línea, en su orden respectivo (p (1) a p (k)).

Ejemplo

Entrada	Producción
4 2 2 PRIMEROS 3 4 SEGUNDO 1 2 TERCERO 8 7 CUARTO 5 6	TERCERO SEGUNDO
6 5 3 1OGLOBINA 1 1 1 1 1 GLU10 9 9 9 9 9 8EIN 8 9 8 9 9 COLA6EN 6 5 4 3 2 7ILK 3 4 5 6 7 0LBUMIN 1 2 0 2 1	GLU10 1OGLOBINA 7ILK

Problema G. Sopa de letras

Nombre del archivo fuente:	Word.c, Word.cpp, Word.java, Word.py Estándar
Aporte:	
Producción:	Estándar

Encuentra partes de una cuadrícula 2D que coincidan con una palabra 2D.

Aporte

- Una línea que contiene el número de filas y columnas en la clave de búsqueda, rk y ck (1 ≤ r, c ≤ 2000).
- rk líneas adicionales, cada una conteniendo ck caracteres latinos que componen una fila de la clave de búsqueda.
- Una línea que contiene el número de filas y columnas en el pajar, rh y ch (rk ≤ rh ≤ 2000, ck ≤ ch ≤ 2000).
- rh líneas adicionales, cada una conteniendo ch caracteres latinos que componen una fila de la clave de búsqueda.

Salida: Ilustre
las áreas coincidentes del pajar imprimiendo una cuadrícula del mismo tamaño. En las ubicaciones que formen parte de al menos una coincidencia, imprima el carácter original del pajar. En otros casos, imprima un punto "."

Ejemplo

Entrada	Producción
3 3 ghi lmn QRS 5 5 abcde fghij klmno primero uvwxy	..... .ghi. .lmn. .qrs. .....
1 2 desde 6 4 abba bebé abba bebé abba bebé	sobre.. ..ab sobre.. ..ab sobre.. ..ab
4 1 norte a norte a 7 6 ananan nanana ananan nanana ananan nanana ordenanza	.nnn nanana ananan nanana ananan .a.ana ....a.
2 2 oo oo 5 5 Besos y abrazos oxooo ooxoo oooxo oooox	..ooo ..ooo oo.oo oooo.. oooo..

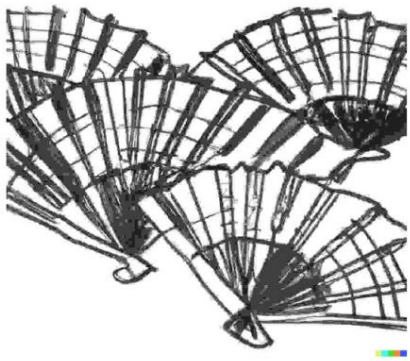
Problema H. Abanico de mano

Nombre del archivo fuente: Handfan.c, Handfan.cpp, Handfan.java, Handfan.py Estándar

Aporte:

Producción: Estándar

El pequeño Fran recibió como regalo un marco de madera con forma de polígono regular. Como el polígono tiene n vértices, también recibió n(n−3) palitos de madera, que corresponden a cada diagonal posible. Los vértices del polígono están etiquetados con números enteros del 1 al n en sentido antihorario. Al principio, Fran dispuso n−3 palitos dentro del marco de tal manera que cada uno tocara dos vértices no adyacentes del marco, y ningún par de palitos se cruzara. En otras palabras, realizó una triangulación. Como esto no le resultó lo suficientemente interesante, decidió experimentar con esta configuración aplicando una operación particular que consta de dos pasos:



1. Retire un palo.
2. Agregamos un nuevo palo de tal manera que obtengamos un nuevo triangulación.

Caracterizamos la operación con un par ordenado de pares no ordenados ((a, b), (c, d)) lo que significa que la pequeña Fran quitó un palo que tocaba los vértices a y b, y agregó un palo que tocaba los vértices c y d.

A Fran le encantan los abanicos, así que mientras hace estas operaciones, a veces se pregunta: “¿Cuántas operaciones se necesitan para transformar esta triangulación en una triangulación en “abanico” en el vértice x, y de cuántas maneras se puede lograr esto?”.

Como está ocupado haciendo operaciones y divirtiéndose, ¡te pide ayuda!

La triangulación en “abanico” en el vértice x es una triangulación donde todas las diagonales tienen un punto final común, es decir, el vértice x.

Sea m el número de operaciones necesarias. Sea f1, f2, ..., fm una secuencia de operaciones que, al aplicarse en el orden dado, logra la triangulación deseada, lo que representa una forma de lograrla. Sea sm otra secuencia similar. Dos secuencias son distintas si existe un índice i tal que s1, s2, ..., fi = si .

Como el número de tales secuencias puede ser enorme, a la pequeña Fran sólo le interesa su resto módulo 109+7.

Entrada

En la primera línea están los números enteros n y q $4 \le n \le 2 \cdot 10^5$, $1 \le q \le 2 \cdot 10^5$, el número de vértices y la número de eventos.

En cada una de las siguientes n − 3 líneas hay números enteros xi , yi (1 ≤ xi , yi ≤ n), las etiquetas de los vértices que toca el i-ésimo palo.

En cada una de las siguientes q líneas está el entero ti (1 ≤ ti ≤ 2) que representa el tipo de evento.

Si ti = 1, le siguen 4 enteros ai di (1 ≤ ai di ≤ n), lo que indica que se está realizando una operación ((ai , bi), (ci , di)) en ese momento. Se garantiza que dicha operación se pueda realizar.

Si ti = 2, le sigue un entero xi (1 ≤ xi ≤ n), lo que significa que a la pequeña Fran le interesan los datos de la triangulación en “abanico” en el vértice xi en relación con la triangulación actual.

Producción

Para cada evento de tipo 2, en orden de entrada, salida dos enteros, número mínimo de operaciones

necesario y número de formas de llegar a la triangulación objetivo utilizando un número mínimo de operaciones.

Ejemplo

Entrada	Salida
4 3 1 3 2 1 1 1 3 2 4 2 1	0 1 1 1
5 4 1 3 3 5 2 1 2 2 1 1 3 2 5 2 2	1 1 2 1 1 1
9 3 1 5 1 7 2 4 2 5 5 7 7 9 2 1 1 2 5 1 4 2 1	4 12 3 6

Explicación del primer ejemplo: La triangulación inicial ya es una triangulación de abanico en el vértice 1, por lo que la pequeña Fran no debe realizar operaciones, aunque solo existe una forma de hacerlo. Tras ejecutar una operación dada, solo hay una forma de volver al estado anterior: aplicar la operación ((2, 4), (1, 3)).

Aclaración del segundo ejemplo:

Única secuencia de operaciones para la primera consulta: ((3, 5), (1, 4)). Única secuencia de operaciones para la segunda consulta: ((1, 3), (2, 5)), ((3, 5), (2, 4)). Única secuencia de operaciones para la tercera consulta: ((3, 5), (2, 5)).

Problema I. Duro de Matar

Nombre del archivo fuente: Diehard.c, Diehard.cpp, Diehard.java, Diehard.py Estándar

Aporte:

Producción: Estándar

John y Hans juegan a un juego con tres dados. Aunque todos tienen seis caras, no se garantiza que sean idénticos.

Primero, John elige uno de los dados y luego Hans elige uno de los dos restantes. Luego, ambos lanzan el dado elegido. Si obtienen el mismo número, vuelven a lanzar el dado. De lo contrario, gana quien haya sacado el número más alto.

En caso de que ni John ni Hans puedan ganar con los dados elegidos, no se molestan en volver a tirar los dados indefinidamente y no se declara ningún ganador.

¿Puedes ayudar a John a elegir un dado que garantice que gana con una probabilidad de al menos $\frac{1}{2}$?

Dados de Miwin del Dr. M. Winkelmann, dominio público

Entrada:

La entrada consta de tres líneas. La línea i contiene 6 enteros positivos xj ($1 \le xj \le 1000$), que describen las caras del i-ésimo dado.

Producción

Genere el i {1, 2, 3} más pequeño, de modo que John pueda elegir el dado i y tenga la garantía de ganar con una probabilidad de al menos $\frac{1}{2}$. Si no existe dicho dado, indique “No hay dados”.

Ejemplo

Entrada	Salida 1
1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6	
1 1 1 1 1 1 2 2 2 2 2 2 3 3 3 3 3 3	3
2 2 4 4 9 9 1 1 6 6 8 8 7 7 5 5 3 3	No hay dados
1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2	No hay dados

Problema J. Red Jabber

Nombre del archivo fuente: Jabber.c, Jabber.cpp, Jabber.java, Jabber.py Estándar

Aporte:

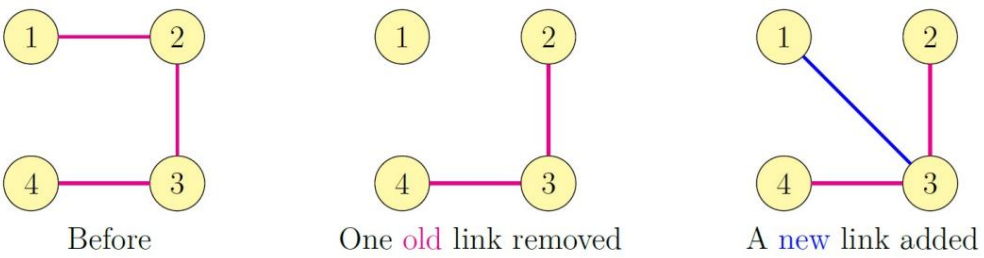
Producción: Estándar

Dave, un antiguo profesor de informática, mantiene una red informática comunitaria local incluso después de jubilarse. Cada miembro de la comunidad tiene una computadora con tres tarjetas de red, y algunas de estas tarjetas pueden estar conectadas por cable. Forman una red conectada y, siguiendo una larga tradición de ahorro de recursos, el número de cables se mantiene al mínimo posible.

Los hábitos de todos los miembros de la comunidad son bastante estables: para cada dos computadoras, el número de paquetes por segundo entre ellas se conoce con exactitud c. Sin embargo, la red se construyó hace mucho tiempo, por lo que las conexiones ya no son necesariamente óptimas. Para dos computadoras numeradas i y j, definimos dij como la ruta más corta entre ellas, medida en número de cables, y cij como el número de paquetes por segundo que deben transferirse de i a j. La tensión de conmutación se define como la suma de cij · dij para todo i < j, y se busca minimizarla.

Dave se dio cuenta de que finalmente había llegado el momento de actualizar los cables; después de todo, se degradan con el tiempo. Quiere aprovechar esta oportunidad para optimizar la red, de modo que la tensión de conmutación sea menor. Sin embargo, ya no es tan rápido como en su juventud, y sus amigos podrían insatisfechos si se producen demasiadas interrupciones a la vez. Así que decidió realizar la actualización utilizando el siguiente escenario.
Para cada uno de los cables viejos, hará lo siguiente:

1. Retire el cable viejo.
2. Vuelva a conectar la red utilizando un cable nuevo, eligiendo las computadoras a conectar de tal manera que La tensión de conmutación resultante es la mínima posible.
3. Si hay muchas maneras de hacer esto, rompa los empates eligiendo las computadoras con los números más pequeños: si (u1, u2) y (v1, v2) dan como resultado el mismo esfuerzo de conmutación, pero u1 < v1 (o u1 = v1 y u2 < v2), entonces se debe elegir (u1, u2) .



Una única operación de reconexión (la primera en la entrada de ejemplo)

Tenga en cuenta que, dado que cada computadora solo tiene tres tarjetas de red, Dave no puede conectar dos computadoras arbitrarias en el segundo paso: si una de ellas ya está conectada a otras tres computadoras, es imposible conectarla a otra. Afortunadamente, no es difícil demostrar que siempre es posible encontrar dos computadoras para conectar: por ejemplo, Dave puede elegir las dos computadoras recién desconectadas.

Desafortunadamente, la tarea resultó ser más difícil de lo que parecía inicialmente. ¿Podrías ayudar a Dave?

Entrada

La primera línea del archivo de entrada contiene un entero n (2 ≤ n ≤ 2 · 103), el número de computadoras en la red.

siguientes n−1 líneas que contienen dos enteros cada una: ai , bi, Donde (1 ≤ ai < bi ≤ n) son los números de las computadoras conectadas inicialmente por un cable antiguo número i. Los cables deben retirarse y reemplazarse en el orden indicado en el archivo de entrada. Se garantiza que es posible acceder a cualquier computadora desde cualquier otra computadora que utilice cables antiguos (es decir, la red está conectada inicialmente) y que ninguna computadora está conectada con más de tres computadoras.

La siguiente línea contiene un entero d (2 ≤ d ≤ 104), el número de pares de computadoras que se sabe que existen. transmitirse datos entre sí.

Las siguientes líneas d contienen tres números enteros cada una si ti di, donde si y ti son los números de las computadoras que se transmiten datos entre sí (1 ≤ si < ti ≤ n), y di (1 ≤ di ≤ 109) es el número de paquetes por segundo que se transmitirán.

Salida donde

por un nuevo (1 ≤ xi , yi ≤ n), deben ser los números Salida n − 1 líneas que contienen dos enteros cada una: xi , yi , de las computadoras conectadas

Tenga en cuenta que, debido a la regla de desempate detallada anteriormente, la salida correcta es única.

Ejemplo

Entrada	Salida
4 1 2 2 3 3 4 6 1 2 1 1 3 10 1 4 1 2 3 10 2 4 1 3 4 10	1 3 2 3 3 4

Problema K. Kindong

Nombre del archivo fuente:	Kindong.c, Kindong.cpp, Kindong.java, Kindong.py Estándar
Aporte:	
Producción:	Estándar

En cierto reino, el rey quiere proteger a sus ciudadanos desplegando guardias. Ha reclutado a varios guardias y los ha equipado con armadura pesada para protegerlos de bandidos, caballeros extranjeros y otros malhechores. Sus guardias son duros, pero por desgracia no son muy brillantes y atacarán a cualquiera que lleve armadura, ¡incluso entre ellos!

El reino consta de varias aldeas conectadas por carreteras. Todas las carreteras son de mala calidad.
Algunos están embarrados, otros tienen puentes destartalados. Ninguno puede soportar un guardia con armadura completa. Por lo tanto, el rey debe decidir qué caminos mejorar para que sus guardias puedan llegar a todo el reino. Los caminos son bidireccionales. Cada guardia solo puede desplegarse en una aldea dentro de un subconjunto específico de las aldeas del reino.

El rey necesita minimizar el coste de la mejora de las carreteras, al tiempo que satisface varias otras restricciones:

- Todos los guardias deben estar desplegados, ninguno debe quedar fuera.
- Cada guardia debe ser desplegado en su subconjunto de aldeas.
- Cada aldea debe ser accesible solo para un guardia. Si dos guardias pueden comunicarse entre sí,... luchar.

Ayude al rey a determinar el costo mínimo de mejorar las carreteras de su reino cumpliendo las restricciones anteriores.

Entrada

La primera línea de entrada contiene tres enteros n ($1 \le n \le 300$), $r \le 0 \le r \le \frac{n(n-1)}{2}$ y g ($1 \le g \le n$), donde n es el número de aldeas, r es el número de caminos y g es el número de guardias. Las aldeas están numeradas del 1 al n.

Cada una de las siguientes r líneas contiene tres enteros a, b ($1 \le a < b \le n$) y c ($1 \le c \le 1000$). Cada línea describe una carretera entre la aldea a y la aldea b, cuyo coste de mejora es c. Las carreteras son bidireccionales; un guardia puede ir de a a b o de b a a. Cada par de aldeas tiene como máximo una carretera entre ellas.

Cada una de las siguientes g líneas comienza con un entero k ($1 \le k \le n$), y luego contiene k enteros v ($1 \le v \le n$).
Cada línea describe las aldeas que componen el subconjunto donde se puede colocar un guardia en particular. Los subconjuntos pueden superponerse.

Salida : Se

genera un entero único, que es el coste mínimo que el rey debe pagar para mejorar suficientes caminos para que cada aldea sea accesible con solo un guardia, y todos los guardias estén desplegados. Salida: −1 si no es posible desplegar los guardias de forma que se cumplan todas las restricciones.

Ejemplo

	Salida 8
Entrada 5 6 2 1 2 1 1 3 4 2 4 2 2 5 5 3 4 7 4 5 3 2 1 2 2 2 4	

Problema L. Cubos Lego

Nombre del archivo fuente: Lego.c, Lego.cpp, Lego.java, Lego.py Estándar

Aporte:

Producción: Estándar

Para su decimotercer cumpleaños, los padres de Donald le compraron un juego de cubos Lego nuevo. El juego contiene n cubos del mismo tamaño, y el i-ésimo cubo es del color i. Con estos cubos, decidió construir una pared.

Donald construirá su muro sobre una base de Lego tipo fila que tiene k lugares donde se pueden colocar cubos. Coloca los cubos de la siguiente manera:

- Primero, coloca el cubo de color 1 en un lugar arbitrario del tablero. base.
- Para cada cubo del 2 al n, lo coloca en un espacio contiguo al cubo anterior. Si ese espacio no está vacío, coloca el nuevo cubo encima de todos los demás.

Después de construir el muro, Donald escribió en un trozo de papel una secuencia de longitud k: en la posición i-ésima de la secuencia escribió el color del cubo superior en el lugar i-ésimo, o 0 si no hay un cubo en ese lugar.

Inmediatamente se preguntó cuántas secuencias diferentes podría haber escrito en el trozo de papel.
Dos sucesiones se consideran diferentes si existe una posición en la que difieren. Después de un tiempo, ha logrado calcular la solución, pero no está seguro de si es correcta, así que pide ayuda.

Entrada

La única línea tiene los números enteros n y k ($2 \le n, k \le 5000$), el número de cubos y la longitud de la base.

Salida En la

única línea, imprima la respuesta a la pregunta de Donald, módulo 109 + 7.

Ejemplo

Entrada	Salida 8
4 3	
3 5	14
100 200	410783331

Explicación

Aclaración del primer ejemplo: Todas las

secuencias posibles son: (0, 3, 4), (2, 3, 4), (0, 4, 3), (1, 4, 3), (4, 3, 0), (4, 3, 2), (3, 4, 0), (3, 4, 1).

Aclaración del segundo ejemplo: Una de las posibles

secuencias es (0, 3, 2, 0, 0). Donald puede lograrla colocando el primer cubo en el segundo lugar, el segundo cubo en el tercero y el tercer cubo en el segundo lugar (encima del primero).