acm International Collegiate Programming Contest 2018

Maratón de programación de la SBC – ACM ICPC – 2018

El set de problemas de este cuaderno es utilizado simultáneamente en las siguientes competencias:
Maratona de Programação da SBC 2018
Tercera Fecha Gran Premio de México 2018
Tercera Fecha Gran Premio de Centroamérica 2018

15 de Septiembre de 2018

Cuaderno de Problemas

Información general

Este cuaderno contiene 13 problemas; Las páginas están numeradas de 1 a 22, sin contar esta página. Verifique que su cuaderno está completo.

A) Sobre los nombres de los programas
1) Para soluciones en C/C++ y Python, el nombre del archivo de código fuente no es significativo, puede ser cualquier nombre.
2) Si su solución es en Java, el archivo debe ser llamado: *codigo_de_problema*`.java` donde *codigo_de_problema* es la letra mayúscula que identifica al problema. Recuerde que en Java el nombre de la clase principal debe ser igual que el nombre del archivo.
3) Si su solución es en Kotlin, el archivo debe ser llamado: *codigo_de_problema*`.kt` donde *codigo_de_problema* es la letra mayúscula que identifica al problema. Recuerde que en Kotlin el nombre de la clase principal debe ser llamado igual que el nombre del archivo

B) Sobre la entrada
1) La entrada de su programa debe ser leída de *entrada standard*.
2) La entrada está compuesta de un único caso de prueba, descrito en un número de línea que depende del problema.
3) Cuando una línea de entrada contiene varios valores, estos están separados por un único espacio en blanco; la entrada no contiene ningún otro espacio en blanco.
4) Cada línea, incluyendo la última, contiene exactamente un caracter de final-de-línea.
5) El final de la entrada coincide con el final del archivo.

C) Sobre la salida
1) La salida de su programa debe ser escrita en *salida standard*.
2) Cuando una línea de salida contiene varios valores, estos deben ser separados por un único espacio en blanco; la salida no debe contener ningún otro espacio en blanco.
3) Cada línea, incluyendo la última, debe contener exactamente un caracter de final-de-línea.

Problema A

Aventurándose en Slackline

Beltrano recientemente se interesó en el *slackline*. Slackline es un deporte de equilibrio sobre una cinta elástica estirada entre dos puntos fijos, lo que le permite al practicante caminar y hacer maniobras sobre la cinta. Durante las vacaciones todo lo que Beltrano ha querido hacer es practicar y para eso se ha ido a la granja de un amigo, donde hay una plantación de eucaliptos.

La plantación está muy bien organizada. Los eucaliptos están acomodados en N filas con M árboles en cada una. Hay un espacio de un metro entre cada fila y los árboles en cada fila están todos perfectamente alineados con un espacio de un metro entre ellos.

Beltrano va a montar el slackline uniendo dos árboles. Al montar el slackline a Beltrano no le gusta que la distancia entre los dos árboles sea muy pequeña debido a que las mejores maniobras requieren que la cinta tenga al menos L metros de longitud. Tampoco es posible estirar la cinta demasiado ya que tiene una longitud máxima de R metros. Tenga en cuenta que al estirar la cinta entre los dos árboles elegidos no debería haber ningún otro árbol en la línea formada, de lo contrario no sería posible utilizar toda la cinta para realizar las maniobras.

A Beltrano le gustaría saber de cuantas formas diferentes es posible montar el slackline usando los árboles de la granja. Dos formas se consideran diferentes si por lo menos uno de los árboles donde se amarra la cinta es diferente.

Entrada

La entrada consiste de una única linea que contiene cuatro números enteros, N, M, L, R, representando respectivamente el número de filas y columnas de la plantación y las longitudes mínima y máxima del slackline ($1 \leq N, M \leq 10^5$; $1 \leq L \leq R \leq 10^5$).

Salida

Su programa debe producir una sola línea con un número entero que representan de cuantas maneras diferentes se puede poner el slackline. Como el resultado puede ser grande, la respuesta debe ser ese número módulo $10^9 + 7$.

Ejemplo de entrada 1	**Ejemplo de salida 1**
`2 2 1 1`	`4`

Ejemplo de entrada 2	**Ejemplo de salida 2**
`2 3 1 4`	`13`

Ejemplo de entrada 3	**Ejemplo de salida 3**
`3 4 1 4`	`49`

Problema B

Buscando ganar

Usar canicas como moneda no dio buenos resultados en Cubiconia. En el intento de redimirse con sus amigos, después de robar sus canicas, el emperador decidió invitar a todos a su palacio para una noche de juegos.

Naturalmente, los juegos utilizan canicas, después de todo el emperador necesita encontrarles alguna utilidad. N canicas son extendidas en un tablero cuyas filas son enumeradas de 0 a L y sus columnas enumeradas de 0 a C.

Los jugadores alternan turnos y en cada turno el jugador debe escoger una de las canicas para moverla. El primer jugador que logre mover una canica a la posición $(0, 0)$ es el vencedor. Para que el juego sea interesante, los movimientos son limitados; de lo contrario, el primer jugador siempre movería la canica a la posición $(0, 0)$ y ganaría. Un movimiento consiste en escoger un entero u mayor que 0 y una canica, cuya posición denotaremos (l, c), que puede ser movida a una de las siguientes posiciones, siempre y cuando estén dentro del tablero:

- $(l - u, c)$;
- $(l, c - u)$;
- $(l - u, c - u)$.

Note que más de una canica puede ocupar la misma posición en el tablero.

Como al emperador no le gusta perder, usted debe ayudar a determinar en cuales partidas él debe participar. Como es de esperar, siempre que el emperador juega, tiene el primer turno. Suponiendo que todos juegan de manera óptima, su programa debe analizar la distribución inicial de las canicas en el tablero y determinar si es posible que el emperador gane en caso de que juegue.

Entrada

La primera línea contiene un entero N $(1 \leq N \leq 1000)$. Cada una de las siguientes N líneas contiene dos enteros l_i y c_i indicando en cual fila y columna está la i-ésima canica en el tablero. $(1 \leq l_i, c_i \leq 100)$.

Salida

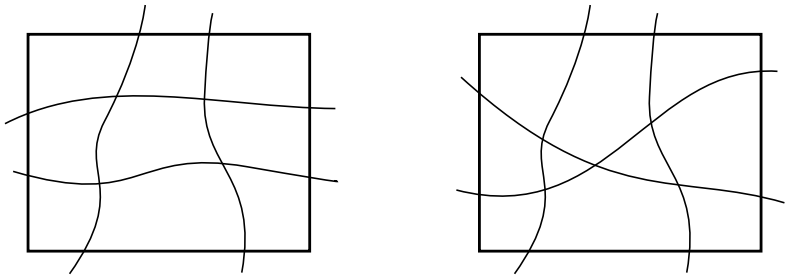
El programa debe imprimir una única línea con el caracter `Y` en caso de que sea posible que el emperador gane el juego o `N` en caso contrario.

Ejemplo de entrada 1	**Ejemplo de salida 1**
2 1 3 2 3	Y

Ejemplo de entrada 2	**Ejemplo de salida 2**
1 1 2	N

Problema C
Cortador de Pizza

El abuelo Giusseppe ganó de regalo un cortador profesional de pizza, de aquellos tipo carretilla, y, para celebrar, ¡ Ha hecho una pizza rectangular gigante para sus nietos!. Él siempre ha cortado sus pizzas en pedazos haciendo líneas continuas como cortes las cuales no necesariamente son rectilíneas, y son de dos tipos: Unas las empieza en el borde izquierdo de la pizza, siguen de manera monótona hacia la derecha hasta terminar en el borde derecho; las otras comienzan en el borde inferior, siguen de manera monótona hacia arriba para terminar en el borde superior. Pero el abuelo Giusseppe siempre seguía una regla: dos cortes del mismo tipo nunca se podían interceptar. Vea un ejemplo con 4 cortes, dos de cada tipo, en la parte izquierda de la imagen, que dividen una pizza en 9 pedazos.



Resulta que el abuelo Giusseppe simplemente ama la geometría, topología, combinatoria y cosas así; es por eso que ha decidido mostrar a los niños que puede obtener más pedazos con el mismo número de cortes si fuera permitido que cortes del mismo tipo se interceptaran. La parte derecha de la imagen muestra, por ejemplo, que si los dos cortes del tipo que van de izquierda a derecha se pudieran interceptar entonces la pizza sería dividida en 10 pedazos.

El abuelo Giusseppe ha descartado la regla, pero no hará cortes aleatorios. Los cortes, además de ser de uno de los dos tipos deberán obedecer las siguientes restricciones:

- Dos cortes tienen como máximo un punto de intersección y, si lo tienen, es porque los cortes se cruzan en ese punto;
- Tres cortes no se interceptan en un mismo punto;
- Dos cortes no se interceptan en el borde de la pizza;
- Un corte no intercepta una esquina de la pizza.

Dados los puntos de inicio y fin de cada corte, su programa debe contar el número de pedazos resultantes de los cortes del abuelo Giusseppe.

Entrada

La primera línea de entrada contiene dos números enteros X y Y, $(1 \leq X, Y \leq 10^9)$, representando las coordenadas (X, Y) de la esquina superior derecha de la pizza. La esquina inferior izquierda tiene siempre coordenadas $(0, 0)$. La segunda línea contiene dos números enteros H y V, $(1 \leq H, V \leq 10^5)$, indicando, respectivamente, el número de cortes que van de izquieda a derecha, y el número de cortes que van de abajo hacia arriba. Cada una de las siguientes H líneas contiene dos números enteros Y_1 y Y_2 definiendo las ordenadas de encuentro de los lados verticales de la pizza con un corte que va del lado izquierdo en la ordenada Y_1 hasta el lado derecho en la ordenada Y_2. Cada una de las siguientes V líneas contiene dos números enteros X_1 y X_2 definiendo las abscisas en las que se encuentran los lados horizontales de la pizza con un corte que va del lado inferior en la abscisa X_1 hasta el lado superior en la abscisa X_2.

Salida

Imprima una línea que contiene un número entero representando el número de pedazos resultantes.

Ejemplo de entrada 1	**Ejemplo de salida 1**
3 4 3 2 1 2 2 1 3 3 1 1 2 2	13

Ejemplo de entrada 2	**Ejemplo de salida 2**
5 5 3 3 2 1 3 2 1 3 3 4 4 3 2 2	19

Ejemplo de entrada 3	**Ejemplo de salida 3**
10000 10000 1 2 321 3455 10 2347 543 8765	6

Problema D

Desvelando Monty Hall

En el escenario de un programa de auditorio hay tres puertas cerradas: puerta 1, puerta 2 y puerta 3. Detrás de una de esas puertas hay un coche, detrás de cada una de las otras dos puertas hay un chivo. La producción del programa sortea aleatoriamente la puerta donde va a estar el coche, sin hacer trampa. Solamente el presentador del programa sabe dónde está el coche. Él pide al jugador elegir una de las puertas. Ahora, como sólo hay un coche, detrás de una de las dos puertas que el jugador no eligió, ¡tiene que haber un chivo!

Por lo tanto, el presentador siempre puede hacer lo siguiente: entre las dos puertas que el jugador no eligió, abre la que tiene el chivo, de modo que el jugador y los espectadores pueden ver el chivo. El presentador entonces pregunta al jugador "¿Desea cambiar su puerta por la otra que aún está cerrada?" ¿Es ventajoso cambiar o no? El jugador quiere quedarse con la puerta que tiene el coche, ¡claro!.

Paulino vio una demostración rigorosa de que la probabilidad de que el carro esté detrás de la puerta que el jugador eligió inicialmente es 1/3 y que la probabilidad de que el carro esté detrás de la otra puerta, que aún está cerrada y que el jugador no eligió inicialmente es 2/3, y que por lo tanto el intercambio es ventajoso. Paulino no se conforma, su intuición le dice que la probabilidad es 1/2 para ambas puertas que aún están cerradas.

En este problema, para acabar con la duda de Paulino, vamos a simular este juego miles de veces y contar cuántas veces el jugador ganó el coche. Vamos a suponer que:

- El jugador siempre elige inicialmente la puerta 1;
- El jugador siempre cambia de puerta después de que el presentador revela al chivo abriendo una de las dos puertas que no fueron elegidas inicialmente.

Con estas condiciones, en un juego, dado el número de la puerta que contiene el coche, podemos saber exactamente si el jugador va a ganar o no el coche.

Entrada

La primera línea de entrada contiene un número entero N ($1 \leq N \leq 10^4$), que representa el número de juegos a simular. Cada una de las siguientes N líneas contiene un número entero: 1, 2 o 3; representando el número de puerta que tiene el coche en ese juego.

Salida

Su programa debe imprimir una única línea conteniendo un número entero que representa el número de veces que el jugador ganó el coche en la simulación, suponiendo que el jugador siempre elige inicialmente la puerta 1 y que siempre cambia de puerta después que el presentador revela un chivo abriendo una de las puertas que no fueron elegidas inicialmente.

Ejemplo de entrada 1	**Ejemplo de salida 1**
5 1 3 2 2 1	3

Ejemplo de entrada 2	**Ejemplo de salida 2**
1 1	0

Ejemplo de entrada 3	**Ejemplo de salida 3**
15 3 2 3 1 1 3 3 2 2 1 2 3 2 1 1	10

Problema E

Enigma

Dada una configuración inicial, la máquina criptográfica alemana Enigma, de la segunda guerra mundial, sustituía cada letra digitada en el teclado por otra letra. La sustitución era bastante compleja, pero la máquina tenía una vulnerabilidad: ¡una letra nunca sería sustituida por ella misma! esa vulnerabilidad fue explorada por Alan Turing, quien trabajo en el criptoanálisis de Enigma durante la guerra. El objetivo era encontrar una configuración inicial de la máquina usando la suposición de que el mensaje contenía cierta expresión usual de comunicación, como por ejemplo la palabra `ARMADA`. Esas expresiones eran llamadas *cribs*. Si el mensaje cifrado era, por ejemplo, `FDMLCRDMRALF`, el trabajo de probar las posibles configuraciones de la máquina era simple porque la palabra `ARMADA`, si estuviese en ese mensaje cifrado, solo podría estar en dos posiciones, ilustradas en la tabla de abajo con una flecha. Las otras cinco posiciones no podrían corresponder al mismo *crib* `ARMADA` porque al menos una letra del *crib*, subrayada en la tabla de abajo, es emparejada con su correspondiente en el mensaje cifrado; como Enigma nunca sustituirá una letra por ella misma, esas cinco posiciones son descartadas por las pruebas.

F	D	M	L	C	R	D	M	R	A	L	F
A	R	M	A	D	A						
	A	R	M	A	D	A	←				
		A	R	M	A	D	A				
			A	R	M	A	D	A	←		
				A	R	M	A	D	A		
					A	R	M	A	D	A	
						A	R	M	A	D	A

En este problema, dado un mensaje cifrado en un *crib*, la tarea es calcular el número de posiciones posibles para el *crib* en el mensaje cifrado.

Entrada

La primera linea de entrada contiene el mensaje cifrado, que es una secuencia de por lo menos una letra y máximo 10^4 letras. La segunda línea contiene el *crib*, que es una secuencia de por lo menos una letra y máximo el mismo número de letras que el mensaje. El *crib* y el mensaje únicamente incluyen las 26 letras del abecedario, mayúsculas y sin tildes.

Salida

Imprima una línea con un entero, indicando el número de posiciones posibles para un *crib* en el mensaje cifrado.

Ejemplo de entrada 1	**Ejemplo de salida 1**
FDMLCRDMRALF ARMADA	2

Ejemplo de entrada 2	**Ejemplo de salida 2**
AAAAABABABABABABABA ABA	7

Problema F

Festival

Los festivales de música deberían ser pura diversión, sin embargo algunos se tornan tan grandes que terminan causando dolores de cabeza a los asistentes. El problema es que son tantas presentaciones buenas en varios escenarios que la simple tarea de escoger a qué presentaciones asistir se torna compleja.

Para ayudar a los asistentes de los festivales de música, Fulano decidió crear una aplicación que, después de evaluar las canciones escuchadas en los servicios de streaming favoritos del usuario, sugiere a cuáles presentaciones asistir de modo que no exista otra combinación de presentaciones mejor de acuerdo a los siguientes criterios:

- Para aprovechar la experiencia al máximo, es importante asistir a cada una de las presentaciones hasta el final de la misma;
- Ir al festival y no asistir a alguno de los escenarios está fuera de consideración;
- Para garantizar que la selección de los artistas será compatible con el usuario, se deben contar cuántas canciones de cada artista el usuario conoce por haberlas escuchado en los servicios de streaming. El total de las canciones conocidas de los artistas elegidos debe ser lo más grande posible.

Infelizmente, la versión beta de la aplicación recibió varias críticas, porque los usuarios podían pensar en selecciones mejores que las sugeridas. La tarea en este problema es ayudar a Fulano a escribir un programa que, dadas las descripciones de los shows que ocurren en cada escenario, calcule la lista ideal para cada usuario.

El tiempo de desplazamiento entre cada escenario es ignorado; por lo tanto desde que no haya intersección entre el horario de dos presentaciones escogidas, se considera posible asistir a ambos. En particular, si un show acaba exactamente cuando otro show empieza, es posible asistir a ambos.

Entrada

La primera línea contiene un número entero $1 \leq N \leq 10$ representando el número de escenarios. Las siguientes N líneas describen las presentaciones que ocurren en cada escenario. La i-ésima de ellas está compuesta por un entero $M_i \geq 1$, representando el número de presentaciones marcados para el i-ésimo escenario seguido por M_i descripciones de las presentaciones. Cada descripción de la presentación contiene 3 enteros i_j, f_j y o_j ($1 \leq i_j < f_j \leq 86400$ y $1 \leq o_j \leq 1000$), representando respectivamente los horarios de inicio y fin de la presentación y el número de canciones del cantante presentándose que fueron previamente escuchadas por el usuario.La suma de los M_i no excederá 1000.

Salida

Su programa debe producir una sola línea con un entero que representan el total de canciones previamente escuchadas de los artistas elegidos o -1 en caso de que no haya solución valida.

Ejemplo de entrada 1	**Ejemplo de salida 1**
3 4 1 10 100 20 30 90 40 50 95 80 100 90 1 40 50 13 2 9 29 231 30 40 525	859

Ejemplo de entrada 2	**Ejemplo de salida 2**
3 2 13 17 99 18 19 99 2 13 14 99 15 20 99 2 13 15 99 18 20 99	-1

Problema G

Gasolina

Terminada la huelga de los camioneros, usted y los demás especialistas en logística de Nlogonia tienen la tarea de planear el reabastecimiento de los puestos de servicio de la ciudad. Para ello, se recogió información sobre las reservas de R refinerías y sobre la demanda de los P puestos de servicio de gasolina. Además, hay restricciones contractuales que hacen que algunas refinerías no puedan atender algunos puestos de servicio; cuando una refinería puede reabastecer un puesto, se sabe el menor tiempo del recorrido para transportar el combustible de un lugar a otro.

La tarea de los especialistas es minimizar el tiempo de abastecimiento de todos los puestos, satisfaciendo completamente sus demandas. Las refinerías tienen una cantidad suficientemente grande de camiones, de modo que es posible asumir que cada camión hará máximo un viaje, de una refinería a un puesto de gasolina. La capacidad de cada camión es mayor que la demanda de cualquier puesto, pero puede ser necesario usar más de una refinería para atender la demanda de un puesto.

Su tarea es encontrar el tiempo mínimo en el que es posible abastecer totalmente todos los puestos de servicio, respetando las restricciones de las refinerías.

Entrada

La primera línea de entrada tiene tres enteros, P, R y C, el número de puestos de servicio, el número de refinerías y el número de pares de refinerías y puestos cuyo tiempo de recorrido será dado, ($1 \leq P, R \leq 1000$ e $1 \leq C \leq 20000$) respectivamente. La segunda línea contiene P enteros D_i ($1 \leq D_i \leq 10^4$), representando las demandas, en litros de gasolina, de los puestos $i = 1, 2, \ldots, P$, en ese orden. La tercera línea contiene R enteros E_i ($1 \leq E_i \leq 10^4$), representando las reservas, en litros de gasolina, de las refinerías $i = 1, 2, \ldots, R$, en ese orden. Finalmente, las últimas C líneas describen los tiempos de recorrido, en minutos, entre puestos y refinerías. Cada una de esas líneas contiene tres enteros, I, J e T ($1 \leq I \leq P$ y $1 \leq J \leq R$ e $1 \leq T \leq 10^6$), donde I identifica un puesto, J identifica una refinería y T el tiempo de recorrido de un camión de la refinería J a un puesto I. No habrá pares (J, I) repetidos. No todos los pares serán informados; en caso de que un par no sea informado, hay restricciones contractuales que impiden a la refinería atender el puesto.

Salida

Imprima un entero T que indique el tiempo mínimo en minutos para que todos los puestos de servicio sea reabastecidos totalmente. En caso de que no sea posible, imprima -1.

Ejemplo de entrada 1	**Ejemplo de salida 1**
3 2 5 20 10 10 30 20 1 1 2 2 1 1 2 2 3 3 1 4 3 2 5	4

Ejemplo de entrada 2	**Ejemplo de salida 2**
3 2 5 20 10 10 25 30 1 1 3 2 1 1 2 2 4 3 1 2 3 2 5	5

Ejemplo de entrada 3	**Ejemplo de salida 3**
4 3 9 10 10 10 20 10 15 30 1 1 1 1 2 1 2 1 3 2 2 2 3 1 10 3 2 10 4 1 1 4 2 2 4 3 30	-1

Ejemplo de entrada 4	**Ejemplo de salida 4**
1 2 2 40 30 10 1 1 100 1 2 200	200

Problema H

Hipótesis Policial

El sistema de transporte público de Nlogonia cuenta con una red express que conecta los principales puntos turísticos del país. Se utilizan $N - 1$ trenes bala para conectar N atracciones de modo que a partir de cualquiera de los puntos turísticos es posible llegar a cualquier otro punto usando la red.

Como en cualquier otro lugar del mundo, es común que haya graffiti en las estaciones del tren. Lo que llamó la atención de la policía del país es el hecho de que en cada una de las estaciones es posible encontrar exactamente una letra pintada con un estilo en específico. La hipótesis es que los criminales pueden estar alterando los graffitis como medio de comunicación y por lo tanto se ha decidido crear un sistema capaz de monitorear los graffitis y sus alteraciones.

Dado un patrón `P`, la descripción de las conexiones entre las estaciones y las letras sospechosas en cada una de las estaciones, tu tarea es escribir un programa capaz de realizar las siguientes operaciones:

- `1` u v: imprime cuantas ocurrencias del patrón `P` existen en el camino de u a v si miramos las letras sospechosas asociadas a los vértices consecutivos en el camino;
- `2` u `x`: Cambia la letra del graffiti de la estación u por la letra `x`.

Entrada

La primera línea contiene dos números enteros N y Q ($1 \leq N, Q \leq 10^5$), representando el número de estaciones y la cantidad de operaciones que deben ser procesadas. La segunda línea contiene el patrón P ($1 \leq |P| \leq 100$). La tercera línea contiene una cadena S con N caracteres representando las letras que se encuentran inicialmente asociadas a cada una de las N estaciones. Cada una de las siguientes $N - 1$ líneas contiene dos números enteros u y v indicando que existe un tren bala que conecta a las estaciones u y v. Las Q líneas siguientes describen las operaciones que deben ser procesadas conforme a las operaciones descritas previamente.

Salida

El programa debe imprimir una línea para cada operación de tipo `1` conteniendo un número entero que representa el número de ocurrencias del patrón `P` en el camino analizado.

Ejemplo de entrada 1	**Ejemplo de salida 1**
4 4 xtc xtzy 1 2 2 3 3 4 1 1 3 2 3 c 1 1 3 1 3 1	0 1 0

Ejemplo de entrada 2	**Ejemplo de salida 2**
6 7 lol dlorlx 1 2 1 3 3 4 3 5 5 6 1 2 6 2 3 l 2 6 l 2 5 o 1 2 6 2 1 o 1 6 2	0 1 2

Ejemplo de entrada 3	**Ejemplo de salida 3**
5 2 aba ababa 1 2 2 3 3 4 4 5 1 1 5 1 5 1	2 2

Problema I

Interruptores

En el panel de control de un gran anfiteatro existen N interruptores, numerados de 1 a N. que controlan las M lámparas del local, enumeradas de 1 a M. Note que el número de interruptores y lámparas no es necesariamente el mismo porque cada interruptor está asociado a un conjunto de lámparas y no sólo a una lámpara. Cuando un interruptor es accionado, el estado de cada una de las lámparas asociadas es invertido. Es decir, aquellas apagadas se encienden y las encendidas se apagan.

Algunas lámparas están encendidas inicialmente y el encargado de seguridad del anfiteatro necesita apagar todas las lámparas. Él comenzó intentando accionar los interruptores manera aleatoria, pero como no estaba consiguiendo apagar todas las lámparas al mismo tiempo decidió seguir una estrategia fija. Él va a accionar los interruptores en la secuencia $1, 2, 3, \ldots, N, 1, 2, 3, \ldots$ es decir, cada vez después de accionar el interruptor con número N, él reanuda la secuencia a partir del interruptor 1. Él pretende accionar los interruptores, siguiendo esa estrategia, hasta que todas las lámparas estén apagadas al mismo tiempo (momento en el que parará de accionar los interruptores). ¿Será que esa estrategia le va a funcionar?

En este problema, dadas las lámparas encendidas inicialmente y dados los conjuntos de lámparas que están asociados a cada interruptor, la tarea es calcular el número de veces que el encargado de seguridad va accionar los interruptores. En caso de que la estrategia nunca apague todas las lámparas al mismo tiempo, el programa debe imprimir -1.

Entrada

La primera línea contiene dos enteros N e M ($1 \leq N, M \leq 1000$) representando, respectivamente, el número de interruptores y el número de lámparas. La segunda línea contiene un entero L ($1 \leq L \leq M$) seguido por L enteros diferentes X_i ($1 \leq X_i \leq M$), representando las lámparas encendidas inicialmente. Cada una de las N líneas siguientes contienen un entero K_i ($1 \leq K_i \leq M$) seguido por K_i enteros diferentes Y_i ($1 \leq Y_i \leq M$),representando las lámparas asociadas al interruptor i ($1 \leq i \leq N$).

Salida

El programa debe imprimir una única línea con un entero representando el número de veces que el encargado de seguridad va accionar los interruptores, siguiendo la estrategia descrita, hasta que todas las lámparas estén apagadas al mismo tiempo. Si ese caso nunca pasa, imprimir -1.

Ejemplo de entrada 1	**Ejemplo de salida 1**
6 3 2 1 3 3 1 2 3 2 1 3 2 1 2 2 2 3 1 2 3 1 2 3	5

Ejemplo de entrada 2	**Ejemplo de salida 2**
3 3 2 2 3 1 3 2 1 2 1 2	-1

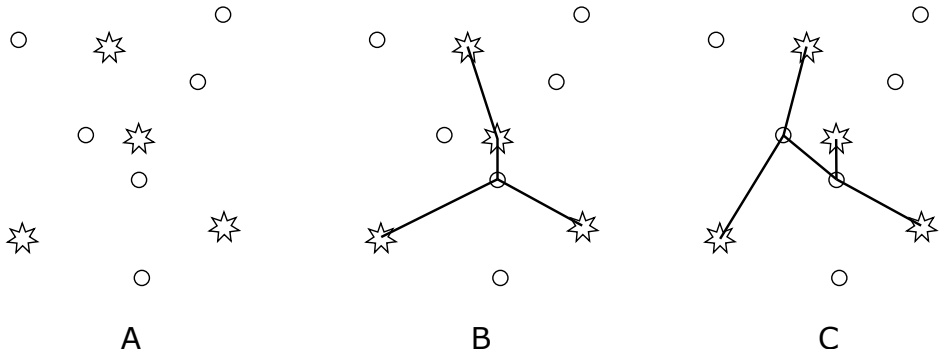
Problema J

Juntando capitales

Un reino lejano posee N ciudades, dentro de las cuales K son capitales. El rey Richard quiere construir líneas de transmisión, cada una de ellas uniendo dos ciudades. Es necesario que haya un camino, es decir, una secuencia de líneas de transmisión entre cualquier par de capitales.

Cada línea de transmisión posee un costo asociado, que es la distancia euclidiana entre las ciudades que la línea de transmisión conecta. Como el rey es un poco ávaro, el desea que las líneas de transmisión sean creadas de modo tal que el costo total (suma de los costos de las líneas) sea lo menor posible.

La figura, en la parte A, muestra un ejemplo de reino con $N = 10$ ciudades, siendo $K = 4$ capitales. El jefe de ingeniería del reino presentó al rey la solución mostrada en la parte B, el cual minimiza el costo total. Pero al rey no le gusto ver una capital con más de una línea de transmisión. Entonces, determino una nueva regla: una capital solo puede estar unida únicamente a otra ciudad. De esa manera y después de mucho trabajar, el jefe de ingeniería ha presentado una nueva solución, ilustrada en la parte C de la figura. Pero ¡ no está seguro de que esa sea la solución óptima y necesita de tu ayuda!



Dadas las coordenadas de las ciudades, tu programa debe calcular el costo total mínimo posible para construir las líneas de transmisión de modo que todo par de capitales esté unida por un camino y que cada capital este unida solo a una ciudad.

Entrada

La primera línea de entrada contiene dos números enteros N y K, $4 \leq N \leq 100$ y $3 \leq K < \min(10, N)$, representando, respectivamente el número de ciudades y el número de capitales. Las siguientes N líneas tienen, cada una, dos enteros X y Y, $-1000 \leq X, Y \leq 1000$, representando las coordenadas de una ciudad. Las primeras K ciudades son las capitales. No hay dos ciudades con las mismas coordenadas.

Salida

El programa debe imprimir una línea con un número decimal con 5 posiciones decimales, indicando el costo total mínimo para construir las líneas de transmisión de acuerdo a las restricciones descritas.

Ejemplo de entrada 1	**Ejemplo de salida 1**
6 4 -20 10 -20 -10 20 10 20 -10 -10 0 10 0	76.56854

Ejemplo de entrada 2	**Ejemplo de salida 2**
22 9 -3 -25 0 -6 -1 -9 2 -21 -5 -19 0 -23 -2 24 -4 37 -3 33 -3 -12 2 39 3 -49 -3 -26 2 24 5 3 -4 -9 -2 -9 -4 8 3 -33 -2 31 -1 -13 0 2	95.09318

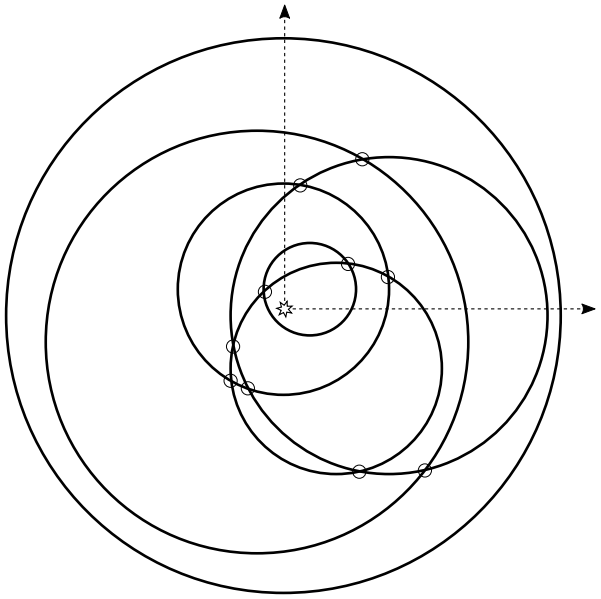
Problema K

Kepler

En este extraño sistema planetario, N planetas siguen órbitas circulares al rededor de una estrella que está posicionada en las coordenadas $(0, 0)$ del sistema. La estrella está dentro del interior de todos los círculos que definen las órbitas, sin embargo, el centro de las órbitas no es necesariamente la coordenada $(0, 0)$.

Las órbitas circulares están en posición general: si dos órbitas se interceptan, entonces, se interceptan en dos puntos distintos; además, nunca se interceptan tres órbitas en un mismo punto.

El científico Juan Kepler está interesado en probar una nueva teoría y, para eso, ha pedido tu ayuda para calcular el número de puntos de intersección entre las órbitas en caso de que ese número sea menor o igual a $2N$. En caso contrario, solo necesita saber que el número es mayor que $2N$.



Entrada

La primer línea de entrada contiene un número entero N ($2 \leq N \leq 10^5$), representando el número de órbitas. Cada una de las siguientes N líneas contienen tres números reales, con exactamente 3 dígitos decimales, X, Y y R, definiendo las coordenadas del centro y el radio de las órbitas.

Salida

Imprima una línea con un entero, representando el número de puntos de intersección entre las órbitas siempre que ese número sea menor o igual a $2N$, en caso contrario, imprima "`greater`".

Ejemplo de entrada 1	**Ejemplo de salida 1**
6 0.000 1.000 4.000 0.000 0.000 10.500 4.000 0.000 6.000 1.000 1.000 1.750 -1.000 -1.000 8.000 2.000 -2.000 4.000	10

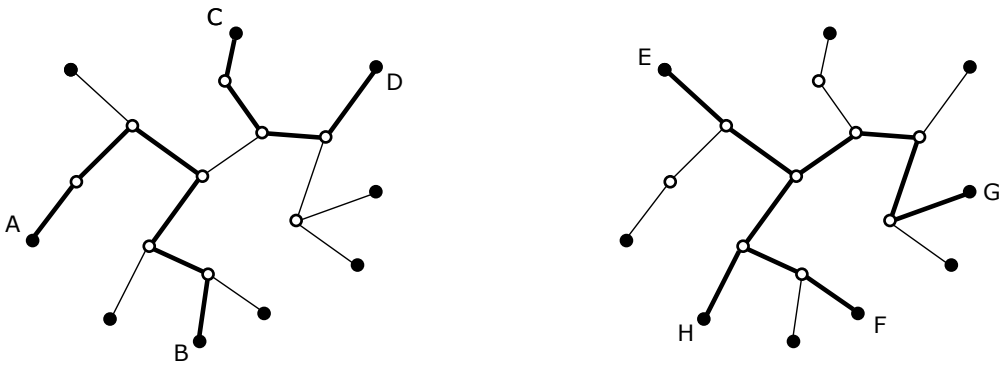
Ejemplo de entrada 2	**Ejemplo de salida 2**
4 -1.000 -1.000 3.000 1.000 -1.000 3.001 -3.004 3.003 5.002 1.000 1.000 3.005	greater

Problema L

Líneas del metro

El sistema de metro de una gran ciudad está formado por un conjunto de estaciones y por túneles que conectan algunos pares de estaciones. El sistema fue diseñado de tal forma que existe exactamente una secuencia de túneles conectando cualquier par de estaciones. Hay varias líneas de trenes que hacen viajes de ida y de vuelta entre dos estaciones terminales, transitando por el camino único entre ellas. La población se está quejando de las líneas actuales, por eso, el alcalde de la ciudad ordenó una reformulación total de las líneas. Como el sistema tiene muchas estaciones, necesitamos ayudar a los ingenieros que están intentando decidir cuales pares de estaciones pasarán a definir una línea.

La figura ilustra un sistema donde las estaciones terminales son mostradas como círculos rellenados y las estaciones no terminales como círculos vacíos. En la parte izquierda note que el par (A,B) define una línea y el par (C,D) define otra y no tienen ninguna estación en común. Sin embargo, en la parte derecha, podemos ver que los pares (E,F) y (G,H) definen dos líneas con dos estaciones en común.



Dada la descripción del sistemas de túneles y una secuencia de Q consultas compuestas por dos pares de terminales, su programa debe calcular, para cada consulta, cuántas estaciones en común tendrían esas líneas definidas por esos dos pares de estaciones.

Entrada

La primera línea de entrada contiene dos enteros N ($5 \leq N \leq 10^5$) y Q ($1 \leq Q \leq 20000$), representando respectivamente el número de estaciones y el número de consultas. Las estaciones están enumeradas de 1 hasta N. Cada una de las $N-1$ líneas siguientes contiene dos enteros distintos U y V, $1 \leq U, V \leq N$, indicando que existe un túnel entre las estaciones U y V. Cada una de las Q líneas siguiente contiene cuatro enteros distintos A, B, C y D ($1 \leq A, B, C, D \leq N$), representando una consulta: las dos líneas de tren son definidas por los pares (A, B) y (C, D).

Salida

Para cada consulta, su programa debe imprimir una línea con un entero representando cuantas estaciones en común tendrían las dos líneas de tren definidas en la consulta.

Ejemplo de entrada 1	**Ejemplo de salida 1**
10 4 1 4 4 5 3 4 3 2 7 3 6 7 7 8 10 8 8 9 6 10 2 5 1 9 5 10 9 10 2 1 5 10 2 9	0 4 0 3

Ejemplo de entrada 2	**Ejemplo de salida 2**
5 1 1 5 2 5 5 3 5 4 1 2 3 4	1

Problema M
Modificando el SAT

El problema de satisfacibilidad booleana (conocido como SAT) consiste en decidir, dada una fórmula booleana en la forma normal conjuntiva, si existe alguna asignación de valores "verdadero" o "'falso" a sus variables de modo que la fórmula entera sea verdadera.

En la forma normal conjuntiva, la fórmula se da en un formato específico. En primer lugar, las únicas operaciones lógicas utilizadas son "Y" , "O", y la negación, denotadas por $\wedge, \vee$ y $\neg$, respectivamente. Una fórmula se forma a través de la operación "Y" de diferentes partes, llamadas cláusulas, $C_1, \ldots, C_m$. De esta forma, una fórmula φ tendrá el siguiente formato:

$$\varphi = C_1 \wedge \cdots \wedge C_m.$$

Además, cada una de las cláusulas también tiene un formato específico. En particular, cada una de las cláusulas es compuesta por la operación "O" de literales que son variables o negaciones de variables, encerrados por paréntesis. Assi, $(x_1 \vee \neg x_2)$ es una cláusula válida, mientras que $(x_1 \wedge \neg x_2)$ no lo es porque usa el operador "Y". Un ejemplo de una fórmula completa es:

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3).$$

Una variación al problema SAT es conocido como k-SAT, donde cada cláusula puede tener máximo k literales. La fórmula de arriba sería un ejemplo de una instancia del problema 3-SAT, pero no del 2-SAT. Note que, en todos estos problemas, para que una fórmula sea verdadera, cada una de las cláusulas debe ser verdadera y, por lo tanto, por lo menos uno de los literales (de la forma x_i o $\neg x_i$) de cada cláusula debe ser verdadero.

Una *asignación* es un modo de definir las variables como verdaderas o falsas. En este problema estamos interesados en una variación del problema 3-SAT, en el que una asignación válida debe tener exactamente 1 o exactamente 3 literales verdaderos en cada cláusula. Dada una fórmula, su tarea es decidir si existe una atribución válida, tomando en cuenta la restricción mencionada. En caso de que haya al menos una atribución válida, debe imprimir la mayor lexicográficamente. El orden lexicográfico es definido de la siguiente manera: dadas dos atribuciones diferentes, podemos compararlas buscando la variable de menor índice que difiere en las dos asignaciones; de las dos, la mayor asignación es la que da valor verdadero para dicha variable.

Entrada

La primera línea de entrada contiene dos números enteros M y N ($1 \leq M, N \leq 2000$), representando, respectivamente, el número de cláusulas y el número de variables. Las siguientes M líneas, cada una describe una cláusula (véase el ejemplo para detalles del formato). Cláusulas consecutivas son separadas por la cadena " `and`". Cada cláusula contiene un máximo de 3 literales. Las variables son denotadas por "`x`" seguido de un número entre 1 y N. No habrá dos espacios consecutivos, ni habrá espacio en blanco al final de las líneas.

El primer ejemplo muestra la fórmula φ mencionada anteriormente.

Salida

Su programa debe imprimir una única línea con N caracteres correspondientes a la asignación lexicográficamente mayor que es válida, o el texto `impossible` en caso de que no exista alguna asignación válida. El i-ésimo caracter debe ser `T` si la variable es verdadera en la asignación y `F` en caso contrario.

Ejemplo de entrada 1	**Ejemplo de salida 1**
4 3 (x1 or x2 or x3) and (not x1) and (x1 or not x2 or x3) and (x2 or not x3)	impossible

Ejemplo de entrada 2	**Ejemplo de salida 2**
5 6 (not x1) and (x1 or x2 or x4) and (x1 or x3 or x5) and (not x2 or x3 or x5) and (x2 or x3 or not x4)	FTTFFT

Ejemplo de entrada 3	**Ejemplo de salida 3**
1 1 (x1 or x1 or not x1)	F