acm International Collegiate Programming Contest 2018

Maratona de Programação da SBC 2018

Sub-Regional Brasil do ACM ICPC

15 de Setembro de 2018

Caderno de Problemas

Informações Gerais

Este caderno contém 13 problemas; as páginas estão numeradas de 1 a 22, não contando esta página de rosto. Verifique se o caderno está completo.

A) Sobre os nomes dos programas
1) Para soluções em C/C++ e Python, o nome do arquivo-fonte não é significativo, pode ser qualquer nome.
2) Se sua solução é em Java, ela deve ser chamada *codigo_de_problema*`.java` onde *codigo_de_problema* é a letra maiúscula que identifica o problema. Lembre que em Java o nome da classe principal deve ser igual ao nome do arquivo.
3) Se sua solução é em Kotlin, ela deve ser chamada *codigo_de_problema*`.kt` onde *codigo_de_problema* é a letra maiúscula que identifica o problema. Lembre que em Kotlin o nome da classe principal deve ser igual ao nome do arquivo.

B) Sobre a entrada
1) A entrada de seu programa deve ser lida da *entrada padrão*.
2) A entrada é composta de um único caso de teste, descrito em um número de linhas que depende do problema.
3) Quando uma linha da entrada contém vários valores, estes são separados por um único espaço em branco; a entrada não contém nenhum outro espaço em branco.
4) Cada linha, incluindo a última, contém exatamente um caractere final-de-linha.
5) O final da entrada coincide com o final do arquivo.

C) Sobre a saída
1) A saída de seu programa deve ser escrita na *saída padrão*.
2) Quando uma linha da saída contém vários valores, estes devem ser separados por um único espaço em branco; a saída não deve conter nenhum outro espaço em branco.
3) Cada linha, incluindo a última, deve conter exatamente um caractere final-de-linha.

Promoção:

SBC Sociedade Brasileira de Computação

v1.0

Problema A

Aventurando-se no Slackline

Beltrano recentemente se interessou por *slackline*. Slackline é um esporte de equilíbrio sobre uma fita elástica esticada entre dois pontos fixos, o que permite ao praticante andar e fazer manobras em cima da fita. Durante as férias tudo que Beltrano quer fazer é praticar, e para isso ele foi para a fazenda de um amigo, onde há uma plantação de eucaliptos.

A plantação é muito bem organizada. Os eucaliptos estão dispostos em N fileiras com M árvores em cada. Há um espaço de um metro entre cada fileira e as árvores nas diferentes fileiras estão todas perfeitamente alinhadas com um espaço de um metro entre elas.

Beltrano vai montar o slackline usando duas árvores. Ao montar o slackline Beltrano não gosta que a distância entre as duas árvores seja muito pequena, já que as melhores manobras exigem que a fita tenha pelo menos L metros. Também não é possível esticar demais a fita já que ela tem um comprimento máximo de R metros. Note que ao esticar a fita entre as duas árvores escolhidas não pode haver nenhuma outra árvore na linha formada, caso contrário não seria possível utilizar a fita toda para as manobras.

Beltrano gostaria de saber de quantas formas diferentes é possível montar o slackline usando as árvores da fazenda. Duas formas são consideradas diferentes se pelo menos uma das árvores onde a fita foi amarrada é diferente.

Entrada

A entrada consiste de uma única linha que contém quatro inteiros, N, M, L, R, representando respectivamente o número de linhas e colunas da plantação e os comprimentos mínimo e máximo do slackline ($1 \leq N, M \leq 10^5$; $1 \leq L \leq R \leq 10^5$).

Saída

Seu programa deve produzir uma única linha com um inteiro representando de quantas formas diferentes o slackline pode ser montado. Como o resultado pode ser grande, a resposta deve ser esse número módulo $10^9 + 7$.

Exemplo de entrada 1	**Exemplo de saída 1**
`2 2 1 1`	`4`

Exemplo de entrada 2	**Exemplo de saída 2**
`2 3 1 4`	`13`

Exemplo de entrada 3	**Exemplo de saída 3**
`3 4 1 4`	`49`

Problema B

Bolinhas de Gude

Usar bolinhas de gude como moeda não deu muito certo em Cubicônia. Na tentativa de se redimir com seus amigos, depois de roubar suas bolinhas de gude, o imperador decidiu convidar todos para uma noite de jogos em seu palácio.

Naturalmente, os jogos utilizam bolinhas de gude, afinal agora o imperador precisa encontrar alguma utilidade para tantas bolinhas. N bolinhas de gude são espalhadas em um grande tabuleiro cujas linhas são numeradas de 0 a L e as colunas numeradas de 0 a C. Os jogadores alternam turnos e em cada turno o jogador da vez deve escolher uma das bolinhas de gude e movê-la. O primeiro jogador que mover uma bolinha para a posição $(0, 0)$ é o vencedor. Para que o jogo seja interessante, os movimentos são limitados; do contrário, o primeiro jogador sempre moveria a bolinha para a posição $(0, 0)$ e venceria. Um movimento consiste em escolher um inteiro u maior que 0 e uma bolinha, cuja localização denotaremos por (l, c), e movê-la para uma das seguintes posições, desde que a mesma não saia do tabuleiro:

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

Note que mais de uma bolinha de gude pode ocupar a mesma posição no tabuleiro.

Como o imperador não gosta de perder você deve ajudá-lo a determinar em quais partidas ele deve participar. Como é de se esperar, sempre que joga o imperador fica com o primeiro turno. Assumindo que todos jogam de forma ótima, seu programa deve analisar a distribuição inicial das bolinhas de gude no tabuleiro e informar se é possível ou não que o imperador vença caso ele jogue.

Entrada

A primeira linha contém um inteiro N ($1 \leq N \leq 1000$). Cada uma das N linhas seguintes contém dois inteiros l_i e c_i indicando em qual linha e coluna a i-ésima bolinha de gude se encontra no tabuleiro ($1 \leq l_i, c_i \leq 100$).

Saída

Seu programa deve produzir uma única linha contendo o caractere `Y` caso seja possível para o imperador ganhar o jogo ou `N` caso contrário.

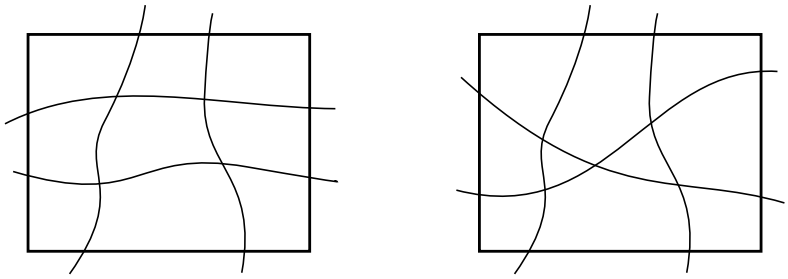
Exemplo de entrada 1	**Exemplo de saída 1**
2 1 3 2 3	Y

Exemplo de entrada 2	**Exemplo de saída 2**
1 1 2	N

Problema C

Cortador de Pizza

Vô Giuseppe ganhou de presente um cortador profissional de pizza, daqueles do tipo carretilha e, para comemorar, assou uma pizza retangular gigante para seus netos! Ele sempre dividiu suas pizzas em pedaços fazendo cortes ao longo de linhas contínuas, não necessariamente retilíneas, de dois tipos: algumas começam na borda esquerda da pizza, seguem monotonicamente para a direita e terminam na borda direita; outras começam na borda inferior, seguem monotonicamente para cima e terminam na borda superior. Mas Vô Giuseppe sempre seguia uma propriedade: dois cortes do mesmo tipo nunca podiam se interceptar. Veja um exemplo com 4 cortes, dois de cada tipo, na parte esquerda da figura, que dividem a pizza em 9 pedaços.



Acontece que Vô Giuseppe simplesmente ama geometria, topologia, combinatória e coisas assim; por isso, resolveu mostrar para as crianças que poderia obter mais pedaços, com o mesmo número de cortes, se cruzamentos de cortes de mesmo tipo fossem permitidos. A parte direita da figura mostra, por exemplo, que se os dois cortes do tipo dos que vão da esquerda para a direita puderem se interceptar, a pizza será dividida em 10 pedaços.

Vô Giuseppe descartou a propriedade, mas não vai fazer cortes aleatórios. Além de serem de um dos dois tipos, eles vão obedecer às seguintes restrições:

- Dois cortes têm no máximo um ponto de interseção e, se tiverem, é porque os cortes se cruzam naquele ponto;
- Três cortes não se interceptam num mesmo ponto;
- Dois cortes não se interceptam na borda da pizza;
- Um corte não intercepta um canto da pizza.

Dados os pontos de começo e término de cada corte, seu programa deve computar o número de pedaços resultantes dos cortes do Vô Giuseppe.

Entrada

A primeira linha da entrada contém dois inteiros X e Y, $(1 \leq X, Y \leq 10^9)$, representando as coordenadas (X, Y) do canto superior direito da pizza. O canto inferior esquerdo tem sempre coordenadas $(0, 0)$. A segunda linha contém dois inteiros H e V, $(1 \leq H, V \leq 10^5)$, indicando, respectivamente, o número de cortes que vão da esquerda para a direita, e o número de cortes que vão de baixo para cima. Cada uma das H linhas seguintes contém dois inteiros Y_1 e Y_2 definindo as ordenadas de encontro dos lados verticais da pizza com um corte que vai do lado esquerdo, na ordenada Y_1, para o lado direito, na ordenada Y_2. Cada uma das V linhas seguintes contém dois inteiros X_1 e X_2 definindo as abscissas de encontro dos lados horizontais da pizza com um corte que vai do lado inferior, na abscissa X_1, para o lado superior, na abscissa X_2.

Saída

Imprima uma linha contendo um inteiro representando o número de pedaços resultantes.

Exemplo de entrada 1	**Exemplo de saída 1**
3 4 3 2 1 2 2 1 3 3 1 1 2 2	13

Exemplo de entrada 2	**Exemplo de saída 2**
5 5 3 3 2 1 3 2 1 3 3 4 4 3 2 2	19

Exemplo de entrada 3	**Exemplo de saída 3**
10000 10000 1 2 321 3455 10 2347 543 8765	6

Problema D

Desvendando Monty Hall

No palco de um programa de auditório há três portas fechadas: porta 1, porta 2 e porta 3. Atrás de uma dessas portas há um carro, atrás de cada uma das outras duas portas há um bode. A produção do programa sorteia aleatoriamente a porta onde vai estar o carro, sem trapaça. Somente o apresentador do programa sabe onde está o carro. Ele pede para o jogador escolher uma das portas. Veja que agora, como só há um carro, atrás de pelo menos uma entre as duas portas que o jogador não escolheu, tem que haver um bode!

Portanto, o apresentador sempre pode fazer o seguinte: entre as duas portas que o jogador não escolheu, ele abre uma que tenha um bode, de modo que o jogador e os espectadores possam ver o bode. O apresentador, agora, pergunta ao jogador: "você quer trocar sua porta pela outra porta que ainda está fechada?". É vantajoso trocar ou não? O jogador quer ficar com a porta que tem o carro, claro!

Paulinho viu uma demonstração rigorosa de que a probabilidade de o carro estar atrás da porta que o jogador escolheu inicialmente é 1/3 e a probabilidade de o carro estar atrás da outra porta, que ainda está fechada e que o jogador não escolheu inicialmente, é 2/3 e, portanto, a troca é vantajosa. Paulinho não se conforma, sua intuição lhe diz que tanto faz, que a probabilidade é 1/2 para ambas as portas ainda fechadas...

Neste problema, para acabar com a dúvida do Paulinho, vamos simular esse jogo milhares de vezes e contar quantas vezes o jogador ganhou o carro. Vamos supor que:

- O jogador sempre escolhe inicialmente a porta 1;
- O jogador sempre troca de porta, depois que o apresentador revela um bode abrindo uma das duas portas que não foram escolhidas inicialmente.

Nessas condições, em um jogo, dado o número da porta que contém o carro, veja que podemos saber exatamente se o jogador vai ganhar ou não o carro.

Entrada

A primeira linha da entrada contém um inteiro N ($1 \leq N \leq 10^4$), indicando o número de jogos na simulação. Cada uma das N linhas seguintes contém um inteiro: 1, 2 ou 3; representando o número da porta que contém o carro naquele jogo.

Saída

Seu programa deve produzir uma única linha, contendo um inteiro representando o número de vezes que o jogador ganhou o carro nessa simulação, supondo que ele sempre escolhe inicialmente a porta 1 e sempre troca de porta depois que o apresentador revela um bode abrindo uma das duas portas que não foram escolhidas inicialmente.

Exemplo de entrada 1	**Exemplo de saída 1**
5 1 3 2 2 1	3

Exemplo de entrada 2	**Exemplo de saída 2**
1 1	0

Exemplo de entrada 3	**Exemplo de saída 3**
15 3 2 3 1 1 3 3 2 2 1 2 3 2 1 1	10

Problema E

Enigma

Dada uma configuração inicial, a máquina de criptografia alemã Enigma, da Segunda Guerra Mundial, substituía cada letra digitada no teclado por alguma outra letra. A substituição era bastante complexa, mas a máquina tinha uma vulnerabilidade: uma letra nunca seria substituída por ela mesma! Essa vulnerabilidade foi explorada por Alan Turing, que trabalhou na criptoanálise da Enigma durante a guerra. O objetivo era encontrar a configuração inicial da máquina usando a suposição de que a mensagem continha uma certa expressão usual da comunicação, como por exemplo a palavra `ARMADA`. Essas expressões eram chamadas de *cribs*. Se a mensagem cifrada era, por exemplo, `FDMLCRDMRALF`, o trabalho de testar as possíveis configurações da máquina era simplificado porque a palavra `ARMADA`, se estivesse nessa mensagem cifrada, só poderia estar em duas posições, ilustradas na tabela abaixo com uma seta. As demais cinco posições não poderiam corresponder ao *crib* `ARMADA` porque ao menos uma letra do *crib*, sublinhada na tabela abaixo, casa com sua correspondente na mensagem cifrada; como a Enigma nunca substituiria uma letra por ela própria, essas cinco posições poderiam ser descartadas nos testes.

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

Neste problema, dada uma mensagem cifrada e um *crib*, seu programa deve computar o número de posições possíveis para o *crib* na mensagem cifrada.

Entrada

A primeira linha da entrada contém a mensagem cifrada, que é uma sequência de pelo menos uma letra e no máximo 10^4 letras. A segunda linha da entrada contém o *crib*, que é uma sequência de pelo menos uma letra e no máximo o mesmo número de letras da mensagem. Apenas as 26 letras maiúsculas, sem acentuação, aparecem na mensagem e no *crib*.

Saída

Imprima uma linha contendo um inteiro, indicando o número de posições possíveis para o *crib* na mensagem cifrada.

Exemplo de entrada 1	**Exemplo de saída 1**
`FDMLCRDMRALF` `ARMADA`	2

Exemplo de entrada 2	**Exemplo de saída 2**
`AAAAABABABABABABABA` `ABA`	7

Problema F

Festival

Festivais de música deveriam ser pura diversão, porém alguns deles se tornam tão grandes a ponto de causar dor de cabeça para os frequentadores. O problema é que são tantas atrações boas tocando em tantos palcos que a simples tarefa de escolher quais shows assistir se torna complexa.

Para ajudar frequentadores de tais festivais, Fulano decidiu criar um aplicativo que, após avaliar as músicas ouvidas em seus serviços de streaming favoritos, sugere quais shows assistir de modo que não exista outra combinação de shows melhor de acordo com os critérios descritos a seguir:

- Para aproveitar a experiência ao máximo é importante assistir cada um dos shows escolhidos por completo;
- Ir no festival e não ver um dos palcos está fora de cogitação;
- Para garantir que a seleção dos artistas seja compatível com o usuário, contou-se quantas músicas de cada artista o usuário conhece por já ter ouvido-as nos serviços de streaming. O total de músicas conhecidas dos artistas escolhidos deve ser o maior possível.

Infelizmente a versão beta do aplicativo recebeu várias críticas, pois os usuários conseguiram pensar em seleções melhores que aquelas sugeridas. Sua tarefa nesse problema é ajudar Fulano e escrever um programa que, dadas as descrições dos shows acontecendo em cada palco, calcula a lista ideal para o usuário.

O tempo de deslocamento entre os palcos é ignorado; portanto, desde que não haja interseção entre os horários de quaisquer dois shows escolhidos considera-se que é possível assistir a todos por completo. Em particular, se um show acaba exatamente quando um outro começa, é possível assistir a ambos.

Entrada

A primeira linha contém um número inteiro $1 \leq N \leq 10$ representando o número de palcos. As N linhas seguintes descrevem os shows acontecendo em cada palco. A i-ésima delas é composta por um inteiro $M_i \geq 1$, representando o número de shows marcados para o i-ésimo palco seguido por M_i descrições de shows. Cada descrição de show contém 3 inteiros i_j, f_j e o_j ($1 \leq i_j < f_j \leq 86400$ e $1 \leq o_j \leq 1000$), representando respectivamente os horários de início e fim do show e o número de músicas do cantor se apresentando que foram previamente ouvidas pelo usuário. A soma dos M_i não excederá 1000.

Saída

Seu programa deve produzir uma única linha com um inteiro representando o total de músicas previamente ouvidas dos artistas escolhidos, ou -1 caso não haja solução válida.

Exemplo de entrada 1	**Exemplo de saída 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

Exemplo de entrada 2	**Exemplo de saída 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 a greve dos caminhoneiros, você e os demais especialistas em logística da Nlogônia agora têm a tarefa de planejar o reabastecimento dos postos da cidade. Para isso, foram coletadas informações sobre os estoques das R refinarias e sobre as demandas dos P postos de gasolina. Além disso, há restrições contratuais que fazem com que algumas refinarias não possam atender alguns postos; quando uma refinaria pode fornecer a um posto, sabe-se o menor tempo de percurso para transportar o combustível de um lugar ao outro.

A tarefa dos especialistas é minimizar o tempo de abastecimento de todos os postos, satisfazendo completamente suas demandas. As refinarias têm uma quantidade suficientemente grande de caminhões, de modo que é possível supor que cada caminhão precisará fazer no máximo uma viagem, de uma refinaria para um posto de gasolina. A capacidade de cada caminhão é maior do que a demanda de qualquer posto, mas pode ser necessário usar mais de uma refinaria para atender a demanda de um posto.

Seu programa deve encontrar o tempo mínimo no qual é possível abastecer totalmente todos os postos, respeitando os estoques das refinarias.

Entrada

A primeira linha da entrada contém três inteiros, P, R e C, respectivamente o número de postos, o número de refinarias e o número de pares de refinaria e posto cujo tempo de percurso será dado ($1 \leq P, R \leq 1000$ e $1 \leq C \leq 20000$). A segunda linha contém P inteiros D_i ($1 \leq D_i \leq 10^4$), representando as demandas, em litros de gasolina, dos postos $i = 1, 2, \ldots, P$, nessa ordem. A terceira linha contém R inteiros E_i ($1 \leq E_i \leq 10^4$), representando os estoques, em litros de gasolina, das refinarias $i = 1, 2, \ldots, R$, nessa ordem. Finalmente, as últimas C linhas descrevem tempos de percurso, em minutos, entre postos e refinarias. Cada uma dessas linhas contém três inteiros, I, J e T ($1 \leq I \leq P$ e $1 \leq J \leq R$ e $1 \leq T \leq 10^6$), onde I é a identificação de um posto, J é a identificação de uma refinaria e T é o tempo do percurso de um caminhão da refinaria J ao posto I. Não haverá pares (J, I) repetidos. Nem todos os pares são informados; caso um par não seja informado, há restrições contratuais que impedem a refinaria de atender o posto.

Saída

Imprima um inteiro T que indica o tempo mínimo em minutos para que todas os postos sejam completamente abastecidos. Caso isso não seja possível, imprima -1.

Exemplo de entrada 1	**Exemplo de saída 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

Exemplo de entrada 2	**Exemplo de saída 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

Exemplo de entrada 3	**Exemplo de saída 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

Exemplo de entrada 4	**Exemplo de saída 4**
1 2 2 40 30 10 1 1 100 1 2 200	200

Problema H

Hipótese Policial

O sistema de transporte público da Nlogônia conta com uma rede expressa conectando os principais pontos turísticos do país. São usados $N-1$ trens-bala para conectar N atrações de modo que a partir de um dos pontos turísticos é possível alcançar qualquer outro ponto usando apenas essa rede expressa.

Como em qualquer lugar do mundo, é comum que haja pichações nas estações de trem. O que chamou a atenção da polícia do país é o fato de que em cada uma das estações é possível encontrar exatamente uma letra pichada com um estilo específico. A hipótese é de que criminosos podem estar alterando as pichações como meio de comunicação e portanto decidiu-se criar um sistema capaz de monitorar as pichações e suas alterações.

Dado um padrão `P`, a descrição das conexões entre as estações e as letras suspeitas em cada uma das estações, sua tarefa é escrever um programa capaz de lidar com as seguintes operações:

- `1` u v: imprime quantas ocorrências do padrão `P` existem no caminho de u até v se olharmos para os caracteres associados a vértices consecutivos do caminho;
- `2` u `x`: Altera a letra suspeita na estação u para `x`.

Entrada

A primeira linha contém dois inteiros N e Q ($1 \leq N, Q \leq 10^5$), representando o número de estações e a quantidade de operações que devem ser processadas. A segunda linha contém o padrão P monitorado ($1 \leq |P| \leq 100$). A terceira linha contém uma string S com N caracteres representando as letras inicialmente associadas a cada uma das estações. Cada uma das $N-1$ linhas seguintes contém dois inteiros u e v indicando que existe um trem-bala entre as estações u e v. As Q linhas seguintes descrevem as operações que devem ser processadas conforme descrito acima.

Saída

Seu programa deve imprimir uma linha para cada operação do tipo `1` contendo um inteiro que representa o número de ocorrências do padrão `P` no caminho analisado.

Exemplo de entrada 1	**Exemplo de saída 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

Exemplo de entrada 2	**Exemplo de saída 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

Exemplo de entrada 3	**Exemplo de saída 3**
5 2 aba ababa 1 2 2 3 3 4 4 5 1 1 5 1 5 1	2 2

Problema I

Interruptores

No painel de controle de um grande anfiteatro existem N interruptores, numerados de 1 a N, que controlam as M lâmpadas do local, numeradas de 1 a M. Note que o número de interruptores e lâmpadas não é necessariamente o mesmo e isso acontece porque cada interruptor está associado a um conjunto de lâmpadas e não apenas a uma lâmpada. Quando um interruptor é acionado, o estado de cada uma das lâmpadas associadas a ele é invertido. Quer dizer, aquelas apagadas acendem e as acesas se apagam.

Algumas lâmpadas estão acesas inicialmente e o zelador do anfiteatro precisa apagar todas as lâmpadas. Ele começou tentando acionar interruptores aleatoriamente mas, como não estava conseguindo apagar todas as lâmpadas ao mesmo tempo, decidiu seguir uma seguinte estratégia fixa. Ele vai acionar os interruptores na sequência $1, 2, 3, \ldots, N, 1, 2, 3, \ldots$ ou seja, toda vez após acionar o interruptor de número N, ele recomeça a sequência a partir do interruptor 1. Ele pretende acionar interruptores, seguindo essa estratégia, até que todas as lâmpadas estejam apagadas ao mesmo tempo (momento em que ele para de acionar os interruptores). Será que essa estratégia vai funcionar?

Neste problema, dadas as lâmpadas acesas inicialmente e dados os conjuntos de lâmpadas que estão associados a cada interruptor, seu programa deve computar o número de vezes que o zelador vai acionar os interruptores. Caso a estratégia do zelador nunca apague todas as lâmpadas ao mesmo tempo, seu programa deve imprimir -1.

Entrada

A primeira linha contém dois inteiros N e M ($1 \leq N, M \leq 1000$) representando, respectivamente, o número de interruptores e o número de lâmpadas. A segunda linha contém um inteiro L ($1 \leq L \leq M$) seguido por L inteiros distintos X_i ($1 \leq X_i \leq M$), representando as lâmpadas acesas inicialmente. Cada uma das N linhas seguintes contém um inteiro K_i ($1 \leq K_i \leq M$) seguido por K_i inteiros distintos Y_i ($1 \leq Y_i \leq M$), representando as lâmpadas associadas ao interruptor i ($1 \leq i \leq N$).

Saída

Se programa deve produzir uma única linha contendo um inteiro representando o número de vezes que o zelador vai acionar os interruptores, seguindo a estratégia descrita, até todas as lâmpadas estarem apagadas ao mesmo tempo. Caso isso nunca vá acontecer, imprima -1.

Exemplo de entrada 1	**Exemplo de saída 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

Exemplo de entrada 2	**Exemplo de saída 2**
3 3 2 2 3 1 3 2 1 2 1 2	-1

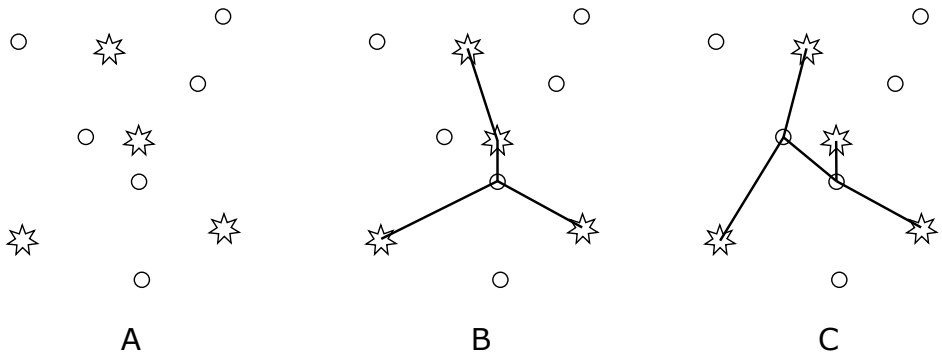
Problema J

Juntando Capitais

Um reino longínquo possui N cidades, dentre as quais K são capitais. O rei Richard quer construir linhas de transmissão, cada uma delas ligando duas cidades. É preciso haver um caminho, ou seja, uma sequência de linhas de transmissão, entre qualquer par de capitais.

Cada linha de transmissão possui um custo associado, que é a distância euclidiana entre as cidades que a linha de transmissão conecta. Como o rei é avarento, ele deseja que as linhas de transmissão sejam criadas de modo que o custo total (soma dos custos das linhas) seja o menor possível.

A figura, na parte A, mostra um exemplo de reino com $N = 10$ cidades, sendo $K = 4$ capitais. O engenheiro chefe apresentou ao rei a solução mostrada na parte B, que minimiza de fato o custo total. Mas o rei não gostou de ver uma capital possuindo mais de uma linha de transmissão. Ele, então, determinou uma nova restrição: uma capital só pode estar ligada a uma outra cidade. Desse jeito, depois de trabalhar muito, o engenheiro chefe apresentou a nova solução, ilustrada na parte C da figura. Só que ele não tem certeza se essa solução é ótima e precisa da sua ajuda!



Dadas as coordenadas das cidades, seu programa deve computar o custo total mínimo possível para construir linhas de transmissão de modo que todo par de capitais esteja ligado por um caminho e toda capital esteja ligada a apenas uma cidade.

Entrada

A primeira linha da entrada contém dois inteiros, N e K, $4 \leq N \leq 100$ e $3 \leq K < \min(10, N)$, indicando respectivamente o número de cidades e o número de capitais. As N linhas seguintes contêm, cada uma, dois inteiros X e Y, $-1000 \leq X, Y \leq 1000$, representando as coordenadas de uma cidade. As K primeiras cidades são as capitais. Não há duas cidades com as mesmas coordenadas.

Saída

Imprima uma linha contendo um número real, com 5 casas decimais, indicando o custo total mínimo para construir as linhas de transmissão, de acordo com as restrições acima.

Exemplo de entrada 1	**Exemplo de saída 1**
6 4 -20 10 -20 -10 20 10 20 -10 -10 0 10 0	76.56854

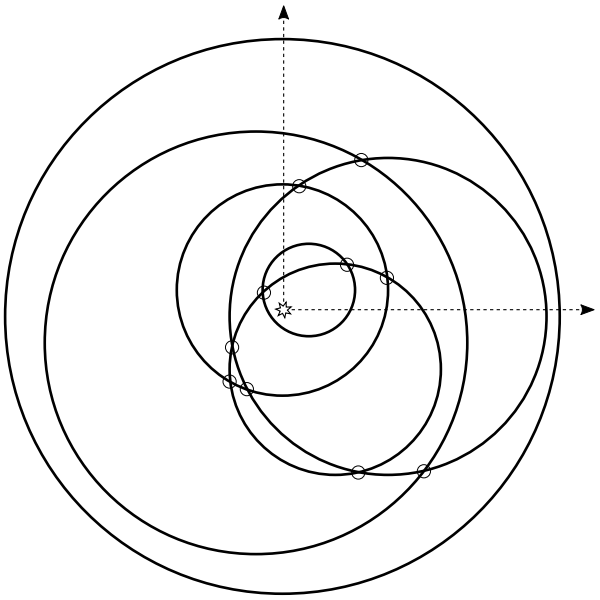
Exemplo de entrada 2	**Exemplo de saída 2**
22 9 -3 -25 0 -6 -1 -9 2 -21 -5 -19 0 -23 -2 24 -4 37 -3 33 -3 -12 2 39 3 -49 -3 -26 2 24 5 3 -4 -9 -2 -9 -4 8 3 -33 -2 31 -1 -13 0 2	95.09318

Problema K

Kepler

Neste estranho sistema planetário, N planetas seguem órbitas circulares ao redor de uma estrela que está nas coordenadas $(0, 0)$ do sistema. A estrela está estritamente contida no interior de todos os círculos que definem as órbitas, mas o centro dessas órbitas não está necessariamente nas coordenadas $(0, 0)$. As órbitas circulares estão em posição geral: se duas órbitas se interceptam, então elas se interceptam em dois pontos distintos; além disso, três órbitas não se interceptam em um ponto comum.

O cientista João Kepler está interessado em testar uma nova teoria e, para isso, pediu sua ajuda para computar o número de pontos de interseção entre as órbitas, caso esse número seja menor que ou igual a $2N$. Caso contrário, precisamos apenas saber que o número é maior do que $2N$.



Entrada

A primeira linha da entrada contém um inteiro N ($2 \leq N \leq 150000$), representando o número de órbitas. Cada uma das N linhas seguintes contém três números reais, com exatamente 3 dígitos decimais, X, Y ($-25.0 \leq X, Y \leq 25.0$) e R ($1.0 \leq R \leq 200000.0$), definindo as coordenadas do centro e o raio das órbitas.

Saída

Imprima uma linha contendo um inteiro, representando o número de pontos de interseção entre as órbitas, se esse número for menor ou igual a $2N$. Caso contrário, imprima "`greater`".

Exemplo de entrada 1	**Exemplo de saída 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

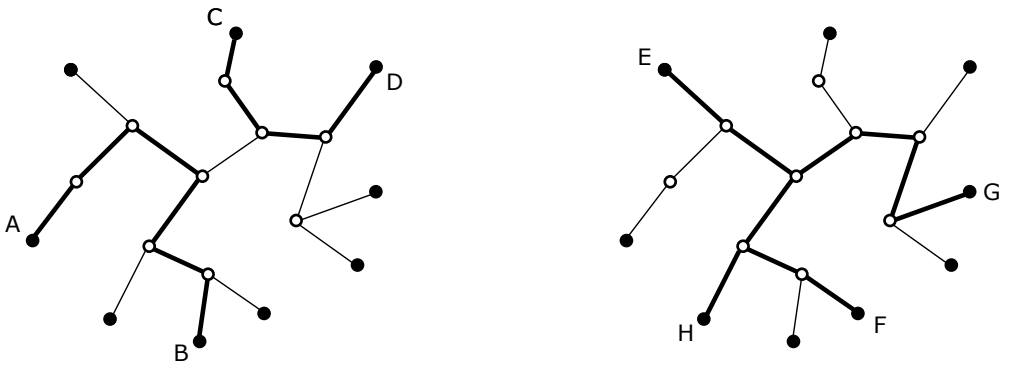
Exemplo de entrada 2	**Exemplo de saída 2**
4 -1.000 -1.000 3.000 1.000 -1.000 3.001 -3.004 3.003 5.002 1.000 1.000 3.005	greater

Problema L

Linhas de Metrô

O sistema de metrô de uma grande cidade é formado por um conjunto de estações e por túneis que ligam alguns pares de estações. O sistema foi desenhado de forma que existe exatamente uma sequência de túneis ligando qualquer par de estações. As estações nas quais apenas um túnel chega são chamadas de terminais. Há várias linhas de trens que fazem viagens de ida e volta entre duas estações terminais, transitando pelo caminho único entre elas. A população está reclamando das linhas atuais e, por isso, o prefeito ordenou uma reformulação total das linhas. Como o sistema possui muitas estações, nós precisamos ajudar os engenheiros que estão tentando decidir quais pares de terminais passarão a definir uma linha.

A figura ilustra um sistema onde as estações terminais são mostradas como círculos preenchidos e as não-terminais são mostradas como círculos vazios. Na parte esquerda, veja que se o par (A,B) definir uma linha e o par (C,D) definir outra, elas não terão qualquer estação em comum. Mas, na parte direita, podemos ver que se os pares (E,F) e (G,H) definirem duas linhas, elas terão duas estações em comum.



Dada a descrição do sistema de túneis e uma sequência de Q consultas constituídas de dois pares de terminais, seu programa deve computar, para cada consulta, quantas estações em comum as linhas definidas pelos dois pares teriam.

Entrada

A primeira linha da entrada contém dois inteiros N ($5 \leq N \leq 10^5$) e Q ($1 \leq Q \leq 20000$), representando respectivamente o número de estações e o número de consultas. As estações são numeradas de 1 até N. Cada uma das $N-1$ linhas seguintes contém dois inteiros distintos U e V, $1 \leq U, V \leq N$, indicando que existe um túnel entre as estações U e V. Cada uma das Q linhas seguintes contém quatro inteiros distintos A, B, C e D ($1 \leq A, B, C, D \leq N$), representando uma consulta: as duas linhas de trem são definidas pelos pares (A, B) e (C, D).

Saída

Para cada consulta, seu programa deve imprimir uma linha contendo um inteiro representando quantas estações em comum teriam as duas linhas de trem definidas pela consulta.

Exemplo de entrada 1	**Exemplo de saída 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

Exemplo de entrada 2	**Exemplo de saída 2**
5 1 1 5 2 5 5 3 5 4 1 2 3 4	1

Problema M

Modificando SAT

O problema da Satisfatibilidade Booleana (conhecido como SAT) consiste em decidir, dada uma fórmula booleana na forma normal conjuntiva, se existe alguma atribuição de valores "verdadeiro" ou "falso" a suas variáveis de forma que a fórmula inteira seja verdadeira.

Na forma normal conjuntiva, a fórmula é dada em um formato bem específico. Em primeiro lugar, as únicas operações lógicas utilizadas são o "E", o "OU" e a negação, denotados por $\wedge, \vee$ e $\neg$, respectivamente. Uma fórmula é formada através da operação "E" de diferentes partes, chamadas cláusulas, $C_1, \ldots, C_m$. Desta forma, uma fórmula φ terá o seguinte formato:

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

Além disso, cada uma das cláusulas também possui um formato específico. Em particular, cada uma das cláusulas é composta pelo "OU" de literais, que são variáveis ou negações de variáveis, cercada por parênteses. Assim, $(x_1 \vee \neg x_2)$ é uma cláusula válida, enquanto $(x_1 \wedge \neg x_2)$ não o seria, por usar o operador "E". Um exemplo completo de fórmula seria:

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

Uma variação do problema SAT é conhecida como k-SAT, onde cada cláusula possui no máximo k literais. A fórmula acima seria um exemplo de instância do problema 3-SAT, mas não de 2-SAT. Note que, em todos estes problemas, para uma fórmula ser verdadeira, cada uma das cláusulas deve ser verdadeira e, portanto, pelo menos um dos literais (da forma x_i ou $\neg x_i$) de cada cláusula deve ser verdadeiro.

Uma *atribuição* é um modo de definir as variáveis como verdadeiras ou falsas. Neste problema estamos interessados em numa variação do problema 3-SAT, no qual uma atribuição válida deve ter exatamente 1 ou exatamente 3 literais verdadeiros em cada cláusula. Dada uma fórmula, sua tarefa é decidir se existe uma atribuição válida, levando em conta tal restrição extra. Caso haja uma atribuição válida, você deve imprimir a lexicograficamente máxima. A ordem lexicográfica é definida do seguinte modo: dadas duas atribuições diferentes, podemos compará-las olhando para a variável de menor índice que difere nas duas atribuições; das duas, a maior atribuição é a que dá valor verdadeiro para tal variável.

Entrada

A primeira linha da entrada contém dois inteiros M e N ($1 \leq M, N \leq 2000$), descrevendo o número de cláusulas e variáveis, respectivamente. Em seguida, serão fornecidas M linhas, cada uma descrevendo uma cláusula (veja o exemplo para detalhes do formato). Cláusulas consecutivas são separadas pela string "` and`". Cada cláusula contém no máximo 3 literais. As variáveis são denotadas por "`x`" seguido de um número entre 1 e N. Não haverá dois espaços consecutivos, nem haverá espaço no final das linhas.

O primeiro exemplo descreve a fórmula φ acima.

Saída

Seu programa deve imprimir uma única linha contendo N caracteres correspondentes a atribuição válida lexicograficamente máxima, ou `impossible` caso não haja atribuição válida. O i-ésimo caractere deve ser `T` se a variável é verdadeira na atribuição e `F` caso contrário.

Exemplo de entrada 1	**Exemplo de saída 1**
4 3 (x1 or x2 or x3) and (not x1) and (x1 or not x2 or x3) and (x2 or not x3)	impossible

Exemplo de entrada 2	**Exemplo de saída 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

Exemplo de entrada 3	**Exemplo de saída 3**
1 1 (x1 or x1 or not x1)	F