II Simulado para Ingressantes 2016

Caderno de Problemas

Universidade de São Paulo

Quinta-feira, 28 de abril de 2016.

Instruções

- A competição tem duração de 5 horas;
- Faltando 1 hora para o término da competição, o placar será congelado, ou seja, o placar não será mais atualizado;
- Faltando 15 minutos para o término da competição, os times não receberão mais a resposta de suas submissões.

- A entrada de cada problema deve ser lida da entrada padrão (teclado);
- A saída de cada problema deve ser escrita na saída padrão (tela);
- Siga o formato apresentado na descrição da saída, caso contrário não é garantido que seu código será aceito;
- Na saída, toda linha deve terminar com o caracter '`\n`';
- O nome do arquivo de códigos em Java deve ser **exatamente** como indicado abaixo do nome de cada problema. Para C/C++ é recomendado usar o nome indicado;
- Para códigos em Java, o nome da classe principal deve ser **igual** ao nome do arquivo.

Respostas das submissões		
Not answered yet	-	Paciência
YES	-	Código aceito. Parabéns!
NO	Compilation error	Erro de compilação
	Wrong answer	Errado. Pode tentar de novo.
	Time limit exceeded	Seu programa demora muito para dar a resposta (certa ou errada)
	Runtime error	Erro em tempo de execução (ex.: *segmentation fault*)
	Problem name mismatch	Leia as duas últimas instruções
	Presentation error	Não está imprimindo no formato exigido no enunciado
	If possible, contact staff	Não sei, você conseguiu fazer algo inesperado

Problema A: MaratonIME joga Cîrokime

Arquivo: `jogo.[c|cpp|java|py]`

Os membros do MaratonIME adoram se divertir, gostam tanto que inventaram um jogo e o nomearam "Cîrokime". O jogo funciona da seguinte forma:
Primeiro n copos com *Cîroc*[1] são enfileirados em uma linha, à frente de cada copo i é escrito um número a_i. É garantido que, se existem dois copos i e j com $i < j$ então $a_i < a_j$, além disso não existe $i \neq j$ tal que $a_i = a_j$. Em seguida os números são cobertos e o jogo começa. O jogador deve então achar o copo que possui um determinado número x, é garatindo que tal copo existe. Para isso ele deve escolher um copo i e beber a bebida, em seguida o seu número a_i é revelado e se for igual ao valor x o jogo encerra, caso contrário, o jogador deve escolher outro copo e assim por diante.

"Cîrokime" é uma tradição entre os membros do MaratonIME, eles jogam em toda festa[2]. Na última festa, Sussu ficou em último lugar, teve que beber os n copos e só no fim encontrou o copo certo. Além de ter ficado em último lugar, Sussu passou mal por ter bebido tanto e teve que ser carregado para casa[3]. Mas a próxima festa está marcada e Sussu quer recuperar sua dignidade, para isso ele quer saber, no pior caso, qual o número máximo de copos que ele terá que beber se jogar de forma ótima.

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (`EOF`).

A entrada consiste em duas linhas. Na primeira, um inteiro n, o número de copos. A segunda contém n inteiros a_i com $1 \leq i \leq n$, os valores escondidos em cada copo.

Saída

A saída consiste em uma única linha contendo um único inteiro: o número mínimo de copos que Sussu deve beber no pior caso se jogar de forma ótima.

Restrições

- $1 \leq n \leq 10^9$
- $1 \leq a_i \leq 10^9$ com $1 \leq i \leq n$

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
3 2 5 7 8 1 2 3 4 5 6 7 8	2 3

[1] Cîroc é uma marca de vodka feita na França, conhecida por seu alto preço de venda no mercado.
[2] Balalaika serve.
[3] História baseada em fatos reais.

Problema B: MaratonIME vai à aula (ou não)

Arquivo: `aula.[c|cpp|java|py]`

No MaratonIME, assim como em muitos outros grupos, alguns alunos querem apenas comparecer a aulas o suficiente para não reprovarem por faltas (como sabemos, na USP é necessário 70% de presença), porém outros são dedicados e tentam conseguir a maior presença possível, indo para a faculdade mesmo quando estão doentes ou incapacitados. Curiosamente não existem outros tipos de alunos no MaratonIME.

Madeira, um antigo membro do MaratonIME, precisa de ajuda, ele está cursando a matéria MAC4815162342, e compareceu a k das m aulas que já foram ministradas, sendo que MAC4815162342 tem n aulas totais por semestre. Ele te pediu ajuda para descobrir como melhor cumprir seus objetivos, mas como você é novo no MaratonIME, não sabe que tipo de aluno ele é. Com vergonha de perguntar mais, você decide resolver os dois problemas, assim não há como errar.

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (`EOF`).

A única linha de entrada possui três inteiros n, m e k.

n é o número de aulas de MAC4815162342 por semestre, m é quantas dessas aulas já foram dadas e k a quantas aulas Madeira compareceu.

Saída

Na primeira linha imprima o número mínimo de aulas que Madeira precisa comparecer para conseguir *pelo menos* 70% de presença, ou -1 se for impossível conseguir 70% de presença.

Na segunda linha imprima a maior porcentagem de presença que Madeira pode conseguir, se for para todas as aulas a partir da próxima. Esse valor deve estar *arredondado para baixo* para o inteiro mais próximo. Não imprima '%'.

Restrições

- $1 \leq n \leq 10^7$
- $0 \leq k \leq m \leq m$

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
10 5 2 11 2 1	5 70 7 90

Notas

No segundo exemplo, a porcentagem máxima que Madeira pode conseguir é 90.9090.

Problema C: MaratonIME vai ao cinema

Arquivo: `cinema.[c|cpp|java|py]`

Acredite se quiser, estudos indicam que não é apenas de maratona que vive o MaratonIME.

Num sábado, depois de simular uma prova, nossa turma de heróis resolveu ir aproveitar a tarde numa sessão de cinema. Mas eles não escolheram ir a um cinema qualquer, mas sim ir assistir um filme n-D, ou seja, com n dimensões.

Em certa cena do filme, durante uma perseguição, o personagem principal pulou por cima de uma corrente de estacionamento. É óbvio que no filme este ato foi feito de forma casual e atlética, mas isto deixou Yan furioso.

"Hollywood fica fazendo essas coisas parecerem fáceis, mas é muito difícil pular por cima de uma corrente!"

O resto da turma, após o filme, argumentou com Yan que o ator com certeza pulou por cima da corrente, e este discordou, afirmando que isso só poderia ser feito por um truque de câmera ou por efeitos especiais.

Para provar seu ponto, ele comprou outro ingresso para o filme e levou instrumentos de medição muito precisos para a sala. O plano de Yan era mostrar que a distância do pé do ator ao chão era menor que a distância da corrente ao chão, o que provaria que o ator de fato não pulou a corrente.

Mas as contas em n dimensões são complicadas. Todo mundo sabe que a distância no plano entre dois pontos (x_0, y_0) e (x_1, y_1) é dada por

$$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$$

Muita gente também sabe que a distância entre dois pontos (x_0, y_0, z_0) e (x_1, y_1, z_1) no espaço tridimensional é calculada pela fórmula

$$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2 + (z_0 - z_1)^2}$$

Ambas as fórmulas descrevem a chamada distância euclideana, no caso bi e tri-dimensional. Seu trabalho aqui é, dado três pontos em n dimensões, dizer se o par mais próximo é o entre o pé e o chão, ou entre o chão e a corrente, de acordo com a distância euclideana.

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (`EOF`).

A entrada começa com um inteiro n seguido de três linhas, cada uma representando um ponto num espaço n-dimensional.

A segunda linha representa as coordenadas do **chão**, e consiste de n inteiros a_i.

A terceira linha representa as coordenadas do **pé do mocinho**, e consiste de n inteiros b_i.

A quarta linha representa as coordenadas da **corrente**, e consiste de n inteiros c_i.

Saída

Imprima quem está certo nesta história toda, ou seja, imprima "Yan" se a distância do chão pro pé do mocinho é **menor ou igual** a distância do chão pra corrente, ou imprima "MaratonIME" se a corrente estiver mais próxima.

Restrições

- $0 < n \leq 10^5$
- Para todo i, $|a_i| \leq 10^4$
- Para todo i, $|b_i| \leq 10^4$
- Para todo i, $|c_i| \leq 10^4$

Exemplos

Exemplo de entrada	Saída para o exemplo de entrada
2 0 0 1 1 2 2	Yan

Exemplo de entrada	Saída para o exemplo de entrada
4 0 0 0 0 2 2 2 2 1 1 1 1	MaratonIME

Problema D: MaratonIME pega o circular

Arquivo: `circular.[c|cpp|java|py]`

Para fazer com que a viagem dos alunos ao metrô não seja tão cansativa, a UESP, Universidade do Estado de São Paulo testou uma de suas mais famosas invenções no ônibus da universidade: eles criaram os circulares de Comprimento Interno Infinito! Em tais maravilhas da Engenharia Moderna sempre existam duplas de bancos vazias para que os alunos se sentem e possam conversar um pouco em sua viagem.

Os integrantes do MaratonIME são muito populares, tão populares que eles possuem amigos em todos institutos da UESP, e assim como a grande maioria de estudantes desta universidade eles também têm que pegar o circular após um longo dia aprendendo a consertar Wi-Fi. Por não fazerem esportes como, por exemplo, remo, todos os alunos da UESP se sentam logo após entrarem no circular, formando duplinhas sempre que possível. Pensando nisso, Gi, experiente Maratonista, bola um problema para pensar no caminho ao metrô: dado um número n que indica quantos institutos estão em uma avenida da UESP e n inteiros a_i que representam a quantidade de pessoas esperando o Circular no ponto do instituto i. Gi quer resolver o mesmo problema várias vezes pra cada avenida, mais especificamente m vezes. Mas é sem graça resolver exatamente o mesmo problema, então, ela pensa em m pares de inteiros. Para todo par l_j, r_j ela quer saber se caso todas as pessoas que estão esperando em qualquer ponto entre l_j e r_j (inclusive) entrassem num ônibus vazio, seria possível ninguém ficar sozinho num par de bancos, ou seja, todas as pessoas que entraram iriam conseguir ficar sentadas com uma duplinha.

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (`EOF`).

A entrada consiste em uma linha contendo dois inteiros n e m, o número de institutos e o número de perguntas de Gi. Depois disso, seguem-se n inteiros a_i, a quantidade de pessoas esperando nos pontos de ônibus de cada instituto e, após isso, m linhas com dois inteiros cada, l_i e r_i, dizendo os institutos inicial e final contidos na pergunta da Gi.

Saída

A saída consiste em uma única linha contendo a string "Sim" se é possível organizar as duplinhas para que ninguém fique sozinho, ou "Nao" caso contrário.

Restrições

- $1 \leq n, m \leq 10^5$
- $0 \leq a_i \leq 10^5 \forall 1 < i \leq n$
- $1 \leq l_j \leq r_j \leq n \forall 1 < j \leq m$

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**

`5 2`
`1 4 10 3 2`
`3 5`
`2 3`
	`Nao`
	`Sim`

Notas

No primeiro exemplo temos uma avenida da UESP com 5 institutos, nos quais temos a seguinte quantidade de pessoas esperando: 1 4 10 3 e 2. Gi, cheia de dúvidas, se faz 2 perguntas: se é possível formar organizar duplinhas caso o circular passe em todos institutos entre os institutos de índices 3 e 5, ou ainda se a mesma distribuição é possível para os índices 2 e 3. Para a primeira pergunta temos 15 pessoas no circular, logo não conseguimos dividir todos em duplinhas, alguém deverá sentar-se sozinho no circular, enquanto que no segundo caso temos 14 pessoas, conseguimos então criar 7 duplinhas no ônibus e não deixar ninguém só.

Problema E: MaratonIME vai ao japonês (de novo)

Arquivo: `japones.[c|cpp|java|py]`

Depois de um longo dia com muito treino, os membros do MaratonIME decidiram ir no restaurante japonês. Sim, nós amamos comida japonesa.

Depois de inúmeras barcas de sushi, quando todos já estavam mais que satisfeitos, eles pediram ao sushiman Sussuchi uma última barca. Desafiado, ele respondeu:

– Vocês querem MAIS UMA barca? Vocês terão mais uma barca...

A barca que ele trouxe foi a maior que qualquer maratonista já tinha visto. Alguns dizem que foi a maior barca que já existiu, superando o limite anterior de 10^5 sushis conquistado pela sushiwoman Gioza em 742, em um festival para o monarca da época, Carlos-sama.

Apesar disso, os maratonistas aceitaram o desafio, e conseguiram comer todos os sushis. Depois, estes estavam tão cheios que não aturavam nem tocar uns aos outros. Tendo comido muita comida, os maratonistas estão desnorteados. Ajude-os a descobrir quais pares de amigos estão se tocando, para que possam se afastar.

Os maratonistas são modelados como círculos num plano, e dois maratonistas se tocam se os seus círculos se tocam. É garantido que nenhum par de círculos se intersecta propriamente, ou seja, a área de sua intersecção é nula.

Entrada

Na primeira linha, um único inteiro n indicando o número de maratonistas.

Cada uma das próximas n linhas tem três inteiros x_i, y_i e r_i, a $(i+1)$-ésima linha descreve o i-ésimo maratonista. (x_i, y_i) são as coordenadas do centro do círculo, e r_i seu raio.

É garantido que a intersecção de qualquer par de círculos tem área nula.

Saída

Para cada par de círculos que se tocam, imprima em uma linha os índices desses círculos. As colisões podem ser impressas em qualquer ordem, os índices dos dois círculos podem ser impressos em qualquer ordem também.

Não imprima as colisões mais de uma vez, ou seja, se i se intersecta com j, imprima i j ou j i, mas não ambos.

Restrições

- $2 \leq n \leq 2000$
- $-10^4 \leq x_i, y_i \leq 10^4$
- $1 \leq r_i \leq 200$

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
3 0 0 2 5 0 3 10 10 1	1 2

Problema F: MaratonIME vai ao karaokê

Arquivo: `karaoke.[c|cpp|java|py]`

Depois de milênios repetindo o título do enunciado deste problema, sempre num tom animado e convidativo, Nathan finalmente conseguiu convencer seus colegas a irem ao karaokê. Ele está simplesmente radiante com esta conquista.

Mas há um problema. Depois de tanto ter insistido com seus amigos para irem ao karaokê, Nathan está com medo de passar vergonha na hora de cantar os clássicos da música brasileira:

- Garçon – Reginaldo Rossi
- Boate azul – Joaquim e Manuel
- Coração de papel – Sérgio Reis
- Borbulhas de amor – Fagner
- Você não me ensinou a te esquecer – Fernando Mendes

Para evitar passar vexame, e para não desencorajar seus coleguinhas sobre futuras idas ao karaokê, Nathan resolveu imprimir todas as cifras de músicas que estão disponíveis no karaokê, para poder consultar enquanto canta. No entanto, isso gerou uma quantidade colossal de papel, que ele simplesmente não consegue carregar.

Mas a perseverança e engenhosidade de um programador com dor de cotovelo não é algo a ser subestimado.

Nathan reparou que, afinal de contas, só existem 7 notas musicais. O pessoal que entende do assunto costuma representá-las pelas letras A,B,C,D,E,F e G. Mais ainda, é comum a mesma nota se repetir em sequência. Ele resolveu então, comprimir as músicas, trocando toda ocorrência de notas repetidas pela nota e quantas vezes ela ocorre.

Por exemplo, dado a sequência

$$(A, A, A, B, B, B, C, G, G, G, G, G, G, G, G, G, G, G)$$

a versão comprimida é

$$A3B3C1G11$$

Infelizmente, Nathan também precisa arrumar seu terno florido e pentear sua barba – dois trabalhos homéricos – e ficou sem tempo para comprimir as notas. Ajude-o a não passar vergonha, escrevendo um programa que faça este serviço.

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (`EOF`).

Cada entrada consiste de apenas uma linha, uma sequência de caractéres S tal que $|S| \leq 10^5$, formada apenas pelas letras A,B,C,D,E,F e G.

Saída

Para cada entrada, imprima uma única linha, tal que cada sequência de notas iguais seja substituída pela nota que ocorre e quantas vezes ela ocorre, conforme o exemplo.

Restrições

- $|S| \leq 10^5$

Exemplos

Exemplo de entrada	Saída para o exemplo de entrada
AAABBBCGGGGGGGGGG	A3B3C1G11

Problema G: MaratonIME vai ao kart

Arquivo: `kart.[c|cpp|java|py]`

Certo dia após um contest, os maratonistas sentiam-se cabisbaixos por conta de resultados ruins. Vendo a situação, Renzo sugeriu que fizessem algo divertido para descontrair a cabeça. Após discussão fervorosa, decidiram correr de kart. Na procura de um kartódromo que fosse viável para todos, acharam o Kartforces, um kartódromo próximo à Cidade Universitária. Entretanto, a pista era muito pequena e comportava apenas dois corredores por bateria. Como bons maratonistas apaixonados por competição, montaram um torneio justo onde todos corriam contra todos, dois a dois, uma única vez. Em cada corrida, o vencedor somava um ponto no placar. Empates são admitidos. O grande campeão foi o maior pontuador. Sabe-se que N maratonistas estavam presentes e:

- Cada maratonista possui uma habilidade h_i.
- Se $h_i > h_j$ onde $1 \leq i, j \leq N$ e $i \neq j$, então o maratonista i vence a corrida contra o maratonista j.

Você teve acesso a habilidade de todos os maratonistas e agora se pergunta quem foi o grande campeão.

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (`EOF`).

Cada instância é composta por duas linhas. A primeira linha contém um inteiro N, o número de maratonistas. A segunda linha contém N inteiros h_i, a habilidade do $i-$ésimo maratonista.

Saída

A saída consiste em um único inteiro i, o maratonista campeão. Caso não seja possível determinar o campeão, imprima -1.

Restrições

- $1 \leq N \leq 10^5$
- $0 \leq h_i \leq 10^{10}$

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
4 9 1 2 7 5 10 13 11 20 0	1 4

Problema H: MaratonIME joga Nim

Arquivo: nim.[c|cpp|java|py]

Você abre seus olhos, mas tudo continua escuro. O mundo está escuro, e tudo balança. Você percebe que está trancado, mas antes de começar a se desesperar, você ouve um porta abrindo e a luz invade sua visão e te cega por alguns momentos.

Te ajudam a sair, você estava trancado num porta-malas. Você não reconhece as faces mascaradas, mas se lembra que no último treino para Maratona te falaram que "o começo ainda estava por vir". "Então esse é o lendário desafio de iniciação do MaratonIME", você já tinha ouvido boatos desse evento, e se sente honrado de ter sido selecionado.

Depois de entrar em um prédio abandonado, te sentam em uma cadeira velha. O primeiro teste é assistir um jogo de futebol sem nenhuma reação de animação. Fácil. O segundo é instalar Linux em um notebook em menos de 5 minutos. Você já estava preparado, sempre carrega seu pendrive com ArchLinux em caso de emergência. Você enfrenta mais testes, e consegue passar em todos apesar de algumas dificuldades.

Depois de horas, os integrantes tiram suas máscaras, e cada um retira uma moeda do bolso. "Ganhei! E ainda fiquei rico" você pensa, mas percebe que eles colocam as moedas numa mesa na sua frente, divididas em duas pilhas. Renzo, o grande chefe do MaratonIME, pega uma cadeira e senta na sua frente. Vocês devem jogar uma partida de Nim, e se você vencer se torna um membro honorário do MaratonIME, ou seja, ganha um balão.

Nim é um jogo de dois jogadores, que alternam turnos. Duas pilhas de moedas ficam em uma mesa e em uma jogada você pode tirar qualquer quantidade não nula de moedas de uma das pilhas. O último jogador a fazer uma jogada (e deixar as duas pilhas vazias) ganha.

Você começa o jogo. Para não ser injusto foi garantido que existe uma forma de você ganhar. Escreva um programa que vença Renzo 100% das vezes.

Entrada

Na primeira linha, dois interos, x e y, o tamanho das pilhas, com $0 \leq x, y \leq 10^4$. É garantido que é possível vencer esse jogo.

Interação

Na sua jogada, imprima dois inteiros, i e x, onde i é o número da pilha que você vai tirar moedas ($i \in \{1, 2\}$), e x o número de moedas que vai retirar ($x \geq 1$, tal que a pilha i tenha pelo menos x pedras).

Na jogada do Renzo, leia dois inteiros, no mesmo formato que os da sua jogada. Toda jogada do Renzo vai ser válida.

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
`2 1` `1 1`	`1 1` `2 1`

Notas

É claro que não fazemos um desafio de iniciação assim :P

No exemplo, temos uma pilha com duas moedas e outra com uma. Você tira uma moeda da primeira pilha, agora qualquer moeda que Renzo retirar, você pode retirar a outra e ganhar.

Lembre-se de, depois de imprimir sua jogada, fazer a descarga da saída, como `fflush(stdout);` em C, `cout.flush();` em C++, ou `sys.stdout.flush()` em Python.

Problema I: MaratonIME ajuda Pablito

Arquivo: `pablito.[c|cpp|java|py]`

Como toda pessoa culta e letrada sabe, os ratos são os seres mais inteligentes do planeta. Os golfinhos são os segundos.

O MaratonIME sabe da hierarquia entre as espécies, e a utiliza em seu favor. Em geral, quando eles precisam de recurso, eles sabem que é sempre bom ter um inteligente rato auxiliando. Mas os ratos não sentem muita empatia por nós, primatas, e só nos ajudam se eles nos devem favores.

Assim, o MaratonIME decidiu ajudar o ratinho Pablito. Pablito está estudando a genealogia dos ratos, para facilitar clonagem e mapeamento genômico. Felizmente, parte do trabalho já está feita pela forma com a qual os ratos se identificam.

A sociedade dos ratos, historicamente, se organiza de forma matriarcal. No princípio, cada Matriarca foi identificada com um número que só é divisível por 1 e por ele mesmo. Era a chamada era primordial, dos primeiros ratos, primário da civilização.

Os demais ratos são sempre descendentes dessa linhagem. Assim, cada um deles é identificado com o produto da identidade de seus pais, e tudo está bem definido.

Pablito precisa, dado dois ratos, saber se eles tem algum descendente em comum. Sua única ferramenta é número de identificação de cada ratinho, que é sempre um inteiro positivo com no máximo 16 dígitos.

Crie um programa que diz se o par de ratos que tem aquela identificação tem algum antecessor em comum.

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (`EOF`).

A entrada começa com um inteiro positivo $t \leq 10^6$.

Depois seguem t linhas, cada uma com dois inteiros a_i e b_i que identificam dois ratinhos.

Saída

Para cada linha, responda "Sim" se os ratinhos a_i e b_i tem um parente em comum e "Não" caso contrário.

Restrições

- $1 < a_i, b_i < 10^16$

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
10 1 10 2 10 3 10 4 10 5 10 6 10 7 10 8 10 9 10 10 10	Não Sim Não Sim Sim Sim Não Sim Não Não

Problema J: MaratonIME Volta para o alojamento

Arquivo: `volta.[c|cpp|java|py]`

Muito frequentado pelo maratonistas mais tradicionais, o acampamento de verão é muito conhecido por suas festas nababescas. Contudo após as festas os competidores sempre devem voltar para seus respectivos alojamentos. Essa volta nem sempre é tranquila, os competidores não muito sóbrios acabam andando em zig-zague e muitas vezes perdendo ou achando dinheiro, além de serem roubados no caminho. O MaratonIME entretanto sempre tem um membro sóbrio em seu grupo, o que garante que eles não vão perder dinheiro no caminho a não ser que sejam roubados. Voltando para o alojamento o MaratonIME sabe que podem ligar para seu coach, Renzo, que irá buscá-los instantaneamente. Entretanto, eles querem a sua ajuda para escrever um programa que, sabendo o caminho, diga qual é o Máximo de dinheiro que é possível levar para o alojamento.

Então sua tarefa será escrever um programa que dado um grid N por M e sabendo que os membros do MaratonIME andam em zig-zague quando bêbados diga qual é o máximo de dinheiro que é possível levar para o alojamento. Deve se levar em consideração os seguintes fatos:

1. Os membros do MaratonIME saem da festa e chegam na rua pelo ponto superior esquerdo e começam andando para a direita.
2. Os membros do MaratonIME saem completamente sem dinheiro da festa.
3. Sempre ao chegar ao fim da rua eles descem um ponto e andam na direção contrária.
4. Sempre que encontram um ladrão no caminho, este lhes roubam todo o dinheiro acumulado.
5. Eles podem ligar para Renzo a qualquer momento, e Renzo irá busca-los instantaneamente.

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (`EOF`).

A enxtrada consiste de dois inteiros, N e M, indicando respectivamente o tamanho da rua e a largura dela, seguidos por N linhas contendo M caracteres que podem ser:

- `'_'` indicando que não há nada naquele ponto da rua.
- `'.'` indicando que há uma moeda de um real naquele ponto da rua.
- `'L'` indicando que há um ladrão naquele ponto da rua.

Saída

A saída consiste de um único inteiro indicando o máximo de dinheiro que MaratonIME pode trazer para o alojamento.

Restrições

- $1 \leq N, M \leq 10^3$

Exemplos

Exemplo de entrada	Saída para o exemplo de entrada
3 4 --·- L__· -·--	2

Problema K: MaratonIME vai a raia

Arquivo: `raia.[c|cpp|java|py]`

Como todos sabem, os maratonistas são exímios remadores. A raia é um ótimo lugar para se remar, correr e fazer academia. O que poucas pessoas apreciam são as capivaras que moram lá. Capivaras são animais fascinantes! Além de muito bonitas, as capivaras possuem vários comportamentos peculiares. Você sabia que as capivaras podem viver em grupos de até 100 indivíduos? Numa bela manhã de sol, Yan corria na raia como de costume. Observando as capivaras, percebeu que elas se organizavam numa linha para tomar sol. Cada capivara estava pareada com somente uma outra. Duas capivaras podem estar pareadas se e somente se ambas se enxergam. Uma capivara enxerga todas as outras na direção que olha. Curioso, Yan decidiu representar as capivaras pelas letras A e B, onde A indica que uma capivara está olhando para a direita e B para a esquerda. Por exemplo, a sequência $AABABB$ representa capivaras tomando sol pois é possível parear todas as capivaras seguindo as regras descritas. De tão fascinado, Yan se descuidou e acabou caindo na raia, fazendo uma bagunça com suas representações. Ele conseguiu recuperar algumas, mas agora elas estavam misturadas com outras coisas. Você pode ajudá-lo descobrir se uma dada representação é de capivaras tomando sol?

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (`EOF`).

Cada instância contém uma sequência S de caracteres, composta apenas por 'A' e 'B', uma representação de Yan.

Saída

A saída contém uma única linha. Imprima "Sim" caso a representação seja de capivaras tomando sol ou "Nao" caso contrário.

Restrições

- $1 \leq |S| \leq 10^5$

Exemplos

Exemplo de entrada	Saída para o exemplo de entrada
AABABB	Sim
ABABA	Nao
AAABBB	Sim
BABA	Nao

Problema L: MaratonIME vai ao restaurante japonês

Arquivo: `restaurante.[c|cpp|java|py]`

Nathan e Yan são programadores de todo coração. Eles aplicam seus conhecimentos de algoritmos em contextos que transcendem a maratona, otimizando as coisas mais triviais do dia a dia.

Alguns questionam se eles só não são meio malucos mesmos, e se não era melhor apenas fazer o que precisava ser feito.

De qualquer forma, eventualmente rolam conflitos quando as abordagens deles divergem sobre o que deve ser feito. Isso rola quando eles vão em restaurantes japoneses.

Todo mundo sabe que o objetivo, num rodízio, é comer a maior quantidade de comidas distintas. Nathan e Yan diferem na abordagem. Cada um aplica um algoritmo e diz que o dele é o que garante que haverá mais iguarias diferentes sendo devoradas.

Nathan sempre come o prato que estiver mais fresco na mesa, ou seja, o que chegou mais recentemente. Em caso de empate, ele prefere o prato que ele comeria mais rápido.

Yan respeita metodicamente a ordem de chegada dos alimentos, e, em caso de empate, também prefere o que ele comeria mais rápido.

Dado o momento de chegada dos pratos, quanto tempo cada um leva pra comer, e quanto tempo eles ficarão no restaurante, diga qual dos dois vai degustar mais pratos diferentes, ou se haverá empate.

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (`EOF`).

A primeira linha da entrada tem dois inteiros: $0 < n \leq 10^5$, que representa quantos pratos serão servidos, e $0 < T \leq 10^6$, que representa quantos minutos eles vão ficar no restaurante.

Depois seguem n linhas, cada uma com dois inteiros t_i e c_i, que representam em minutos, respectivamente, o tempo que demora pra comer aquele prato e o momento da chegada dele.

Você pode assumir que todos os c_i serão dados em ordem não decrescente, ou seja, que para todo i vale $c_i \leq c_{i+1}$.

Saída

Imprima "Nathan", "Yan" ou "Empate", respondendo a pergunta sobre pratos distintos.

Restrições

- Para todo i, vale que $0 \leq c_i \leq 100$
- Para todo i, vale que $0 \leq t_i \leq 10^6$

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**

`5 10`
`10 0`
`10 1`
`5 1`
`10 6`
`10 7`
	`Nathan`

Notas

Repare que o atendimento em alguns restaurantes é muito ruim, e a comida pode acabar só chegando depois que o MaratonIME já foi embora de barriga cheia.

Problema M: MaratonIME joga Xadrez

Arquivo: `xadrez.[c|cpp|java|py]`

Nosso querido Nathan, quando guri, gostava muito de jogar Xadrez, mas isso já faz muito tempo. Um dia desses ele foi desafiado pelo famigerado @luisgust para uma partida de xadrez e, como gosta muito de um desafio, aceitou. O problema é que Nathan anda se esquecendo de algumas das regras do jogo, então ele pediu sua ajuda para determinar, em alguns momentos do jogo, se uma dada peça inimiga pode ser capturada com um único movimento.
O xadrez, no MaratonIME, é representado como um grid de caracteres. Ao invés de jogar com peças brancas e pretas, os times jogam com peças maiúsculas (representadas por caracteres maiúsculos) e minúsculas (represetadas por caracteres minúsculos). O Nathan escolheu jogar com as peças minúsculas, porque elas jogam antes.
Além disso, usualmente, as posições de um tabuleiro de xadrez são dadas num sistema de coordenadas específico. Toda posição é um par com um caractere entre `a` (inclusive) e `h` (inclusive), representando a coluna, e um inteiro entre `1` (inclusive) e `8` (inclusive), representando a linha. Por exemplo, a posição `d2` se refere à quarta casa (da esquerda para a direita) da segunda linha coluna (de cima para baixo) e a posição `f6` se refere à sexta casa da sexta linha. As peças minúsculas começam do lado de baixo do tabuleiro, ou seja, nas linhas `7` e `8`.
Além disso, o termo posição adjacente de uma dada posição se refere a qualquer uma das posições que compartilham pelo menos um vértice com ela, isto é, duas posições são adjacentes se e somente se o máximo entre a distância entre as linhas e a distância entre as colunas das duas posições for igual a 1. Por exemplo, a posição `c4` é adjacente a 7 posições: `b3`, `b4`, `b5`, `c3`, `c5`, `d3`, `d4` e `d5`.
O MaratonIME, além disso, usa um sistema um pouco simplificado de movimentos de xadrez. Para te ajudar com o seu programa, foi fornecido um manual com os movimentos que cada peça pode realizar para capturar outra peça e as letras que representam cada peça.

- O peão, representado pelos caracteres `p` ou `P`, pode capturar peças adjacentes a ela que compartilhem uma diagonal com ela e estejam à frente dela, ou seja, um peão minúsculo pode capturar uma peça adjacente a ele que tenha uma diagonal em comum com ele e que esteja em uma linha estritamente menor que a dele, por exemplo, um peão na posição `c4` pode capturar peças nas posições `b3` ou `d3`.

- O cavalo, representado por `c` ou `C`, faz movimentos em L em qualquer um dos 8 sentidos possíveis, ou seja, ele anda duas casas em uma direção qualquer e, depois, uma casa em uma direção perpendicular à primeira. Um cavalo na posição `c4` pode capturar, em um movimento, peças nas posições `a3`, `a5`, `b2`, `b6`, `d2`, `d6`, `e3` e `e5`.

- A torre, representada por `t` ou `T`, consegue capturar qualquer peça que tenha alguma coordenada igual a ela e que seja visível por ela em uma linha reta, ou seja, não existe nenhuma peça na linha entre a torre e a peça a ser capturada. Por exemplo, uma torre em `c4` pode capturar qualquer peça na coluna `c` e na linha `4`, desde que não haja nenhuma peça entre elas. Se houver uma torre em `c4` e uma outra peça qualquer em `c6`, a torre não pode capturar ninguém em `c7` nem em `c8`, mas uma torre em `c4` pode capturar uma peça em `a4` desde que não tenha ninguém em `b4`.

- O bispo, representado por `b` ou `B`, consegue capturar qualquer peça que esteja em alguma das diagonais que passam pelo bispo e que seja visivel por ele em uma linha reta, ou seja, uma peça tal que a diferença entre as coordenadas-linha dela e do bispo e as coordenadas-coluna dela e do bispo sejam iguais. Por exemplo, um bispo em `c4` consegue capturar alguém em `f7` desde que não haja ninguém nem em `d5` e em `e6`.

- A rainha, representada por `r` ou `R`, consegue fazer os movimentos da torre e os movimentos do bispo, ou seja, consegue, em um movimento, capturar qualquer peça que tenha alguma coordenada igual à dela ou que esteja em alguma diagonal a qual ela pertence desde que

não haja nenhum obstáculo (outra peça) na linha reta entre a rainha e a peça que se quer capturar.

- O rei consegue capturar qualquer peça que esteja em uma casa adjacente à dele, seguindo a definição de adjacente já explicada.
- O caractere `.` representa uma posição vazia.

Seu programa deve receber uma matriz e a posição da peça inimiga que o Nathan quer capturar e responder `Sim` se é possível capturar essa peça com um movimento e `Nao` caso contrário.
É garatido que existe sempre uma peça inimiga na posição dada, é garantido também que cada time contém, no máximo, dois bispos, duas torres, dois cavalos, 8 peões, um rei e uma rainha.

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (`EOF`).

Cada instância consiste de 8 linhas com 8 caracteres cada. A primeira linha contém os caracteres nas posições `a1`, `b1`, ..., `h1`. A segunda linha contém os caracteres nas posições `a2`, `b2`, ..., `h2`. E assim por diante. Depois da matriz, em uma linha separada, são dadas as coordenadas que indicam a posição da peça inimiga que deve ser capturada.

Saída

A saída consiste em uma única linha para cada instância contendo a string `Sim` se é possível capturar a peça ou `Nao` caso contrário.

Exemplos

Exemplo de entrada	**Saída para o exemplo de entrada**
TCBRKBCT PPPPPPPP .... pppppppp tcbrkbct d1 ....R..... ...p.... .... c4	Nao Sim