Problema A
Pique Esconde

fonte: `esconde.c`, `esconde.cpp` *ou* `esconde.java`

Descrição

João, que gosta de informática, procura pensar em como seria um programa para automatizar tudo que ele faz no seu dia a dia. Maria o desafiou a fazer um programa que fosse útil de algum jeito para quando eles brincarem de pique esconde.

João teve uma ideia muito boa de um programa, mas como ele ainda está em suas primeiras aulas de informática, ainda não descobriu como implementar essa ideia. Assim, você deve ajudá-lo. A ideia de João é numerar as crianças participando da brincadeira de 1 a n. O programa recebe como entrada o número total de crianças e também o número correspondente à criança que está procurando os demais participantes. Depois, o programa recebe uma lista com $n - 2$ números identificando as crianças que já foram encontradas e deve responder qual a criança que está faltando.

Sua tarefa é implementar esse programa e ajudar João a mostrar para Maria que a informática pode ser útil até nas mais simples tarefas.

Entrada

A entrada é composta por diferentes casos de teste. A primeira linha de cada caso de teste contém n, o número de crianças, e m, o número da criança que está procurando as demais, $2 \leq n \leq 1000$ e $1 \leq m \leq n$. As $n-2$ linhas seguintes contém um inteiro cada representando as crianças que já foram encontradas.

A entrada termina com $n = m = 0$. Essa linha não deve ser processada.

Saída

Para cada caso de teste, seu programa deve imprimir uma linha contendo o número da criança que ainda não foi encontrada.

Exemplo

Entrada	**Saída**
3 2 1 5 1 2 3 4 4 4 3 2 0 0	3 5 1

Problema B
Concatenação de Strings

fonte: `concstr.c`, `concstr.cpp` *ou* `concstr.java`

Descrição

Seja s uma string. Denotamos por s^n a concatenação de n cópias de s. Por exemplo, se $s =$ "eri" e $n = 3$, temos $s^3 =$ "erierieri".

Dadas strings r e s, sua tarefa é determinar se existem inteiros m e n tais que $r^m = s^n$.

Entrada

Cada linha da entrada conterá duas strings, separadas por um espaço, compostas apenas por letras minúsculas. Cada uma das strings terá pelo menos 1 e no máximo 1.000.000 caracteres. A última linha da entrada será composta por duas strings "#' separadas por espaço. Esta linha não deve ser processada.

Saída

Para cada linha da entrada, você deve gerar uma linha de saída contendo os menores inteiros m e n separados por espaço caso tais inteiros existam, e "NAO", em caso contrário.

Exemplo

Entrada	**Saída**

`aa aaa`
`abra cadabra`
`abcabcabcabcabcabc abcabcabcabc`
`rai mano`
`# #`
	`3 2`
	`NAO`
	`2 3`
	`NAO`

Problema C
Cada um no seu quadrado

fonte: `quadrado.c`, `quadrado.cpp` *ou* `quadrado.java`

Descrição

Ana e Bob gostam de brincar de geometria. Semana passada, enquanto ouviam uma canção que falava de quadrados, Bob, que estava brincando com seus blocos de madeira, se perguntou quantos quadrados poderiam ser formados usando os blocos como vértices, mas sem movê-los. Ana, que aprendeu a programar recentemente, teve a ideia de desenvolver um programa para resolver esse problema. Entretanto, Ana teve dificuldade para resolver esse problema e decidiu pedir a sua ajuda. Como os blocos são pequenos quando comparados à distância entre eles, você pode assumir que eles são pontos em um plano.

Entrada

A entrada consiste de multiplos casos de teste. Cada caso de teste começa com uma linha contendo um único inteiro $4 \leq n \leq 1.000$. As n linhas seguintes contém 2 inteiros x e y cada, $-1.000.000 \leq x, y \leq 1.000.000$, referentes às coordenadas dos blocos. A entrada termina com uma linha contendo $n = 0$, que não deve ser processada.

Saída

Para cada caso de teste, você deve imprimir uma linha contendo um único inteiro, o número de quadrados que podem ser obtidos a partir das posições dadas.

Exemplo

Entrada	**Saída**
6 1 0 0 1 2 1 1 2 0 -1 2 -1 0	2

Problema D
Os últimos serão os primeiros

fonte: `ultimos.c`, `ultimos.cpp` *ou* `ultimos.java`

Descrição

Éric Ruiz Irrigado, o famoso Erí, é conhecido entre seus amigos por querer fazer previsões. Em todo tipo de competição ou evento esportivo ele sempre tenta adivinhar os vencedores, os perdedores, artilheiros e coisas similares. Apesar das brincadeiras e deboches de seus amigos, Erí nunca desistiu e sempre busca padrões onde os outros vêem apenas coincidências.

Acompanhando os times da Maratona de Programação, Erí percebeu que a colocação dos times de seu estado na primeira fase sempre se invertiam na segunda fase, ainda que outros times de outras regiões do país estivessem entre eles. Assim, se o time da *Uni1* ficar na frente da *Uni2* na primeira fase, Erí imagina que o time da *Uni2* ficará na frente do time da *Uni1* na segunda fase.

Para validar sua hipótese, ele quer desenvolver um programa que, dada uma lista de colocação dos times na primeira fase, mostre qual será a posição relativa destes mesmos times na segunda fase.

Entrada

A entrada é composta por diferentes casos de teste. A primeira linha de cada caso de teste contém $n \leq 100$, o número de times do estado de Erí. As n linhas seguintes conterão n inteiros distintos entre 1 e n, inclusive, um por linha, cada inteiro representando um time.

A entrada termina com $n = 0$. Essa linha não deve ser processada.

Saída

Para cada caso de teste, seu programa deve imprimir a posição relativa de cada um dos times de acordo com a previsão de Erí, com um número por linha. Após a lista de times, deve ser impressa uma linha contendo um único "0". Veja o exemplo abaixo.

Exemplo

Entrada	**Saída**
5 3 4 2 5 1 2 1 2 0	1 5 2 4 3 0 2 1 0

Problema E
Hora do rush

fonte: `rush.c`, `rush.cpp` *ou* `rush.java`

Descrição

Em toda cidade vemos que algumas vias são mais largas que outras. Isto é, algumas tem mais faixas de rolamento para veículos do que outras. Cada faixa de rolamento a mais numa via significa que mais carros podem ir de um extremo a outro desta via num mesmo intervalo de tempo.

É comum termos a sensação de que, na hora do rush (aquele período em que a maior parte da população se desloca de casa para o trabalho ou vice-versa), nunca existem faixas de rolamento suficientes por conta do congestionamento que se forma.

Roberval, nosso herói trabalhador, trabalha em uma empresa muito preocupada com a qualidade de vida de seus funcionários. O novo projeto da empresa é mudar a hora de saída do trabalho para um horário de menos trânsito. Roberval, claro, foi eleito para descobrir qual é o melhor horário.

Sua idéia é bem simples: simular a situação do trânsito em diferentes horários na cidade. Mais especificamente, o trânsito do local de trabalho para alguns bairros de interesse, onde moram os funcionários da empresa.

Dado um mapa de ruas da cidade onde Roberval mora e a quantidade de faixas de rolamento livres em cada trecho para um dado horário, diga se é possível que todos os funcionários da empresa saiam do trabalho ao mesmo tempo ou se irão encontrar a mesma situação desagradável do dia-a-dia: mais carros na rua do que as vias comportam.

Assuma que o prédio possui várias saídas de garagem para cada rua adjacente ao prédio, e que todos os cidadãos da cidade são educados no trânsito, ou seja, ninguém trafega no espaço entre dois carros posicionados em faixas de rolamento adjacentes.

Entrada

A entrada é composta por vários casos de teste.

A primeira linha de cada caso de teste contém três inteiros, p, n e m, ($1 \leq p \leq 1000$, $2 \leq n \leq 100$, $1 \leq m \leq n(n-1)$) representando o número de funcionários da empresa, o número de interseções e o número de vias existentes na cidade, respectivamente. As interseções são numeradas de 1 a n, onde 1 representa o local de trabalho. As interseções que não possuem vias levando a outras interseções representam os bairros de interesse.

Em seguida, seguem-se m linhas, cada uma contendo três inteiros u, v e w ($1 \leq u, v \leq n$, $1 \leq w \leq 10$) representando uma via que leva da interseção u até a interseção v com w faixas de rolamento livres. Entre duas interseções existe, no máximo, uma única via em cada direção.

A entrada termina com $p = n = m = 0$. Este caso não deverá ser processado.

Saída

Para cada caso de teste haverá uma linha na saída. Caso seja possível que todos os p funcionários saiam ao mesmo tempo do trabalho sem enfrentar trânsito até suas casas, imprima "HORARIO BOM". Caso contrário, imprima "HORARIO RUIM". Atenção, não utilize acentuação!

Exemplo

Entrada	**Saída**
1 2 1 1 2 1 4 4 4 1 2 2 1 3 2 2 4 2 3 4 1 0 0 0	HORARIO BOM HORARIO RUIM

Problema F
Caminho de Bêbado

fonte: `caminho.c`, `caminho.cpp` *ou* `caminho.java`

Descrição

Desde que saiu da cadeia, Marcos não consegue parar de beber. Ele, um ex-maratonista, havia sido preso injustamente sob acusação de roubar questões de competições de programação e vendê-las para competidores russos e chineses. Depois de 5 longos anos na cadeia, ele foi considerado inocente por falta de provas. Porém, passar tanto tempo atrás das grades de forma injusta o fez entrar em depressão e, desde então, passa todas as suas noites andando de bar em bar, de esquina em esquina. Você, como melhor amigo de Marcos, já está cansado de sempre ter que procurá-lo pela cidade depois de suas noites de bebedeira e resolveu facilitar sua própria vida criando um programa que tenta prever onde encontrar Marcos ao final de suas noites.

Toda noite, antes de sair para beber, Marcos avisa a você a qual bar da cidade ele vai, assim você sempre sabe onde sua noite de bebedeira começa. Depois disso, depois de cada dose bebida, há uma certa chance de Marcos ir a outro bar. Você sabe uma maneira simples de estimar essa chance: a probabilidade de Marcos ir a outro bar é proporcional à divisão da qualidade do bar pela distância entre o bar onde ele está agora e o bar de destino. Também há uma chance de Marcos continuar no mesmo bar, que é proporcional à raíz da qualidade daquele bar.

A qualidade do bar i é dada por Q_i e a distância entre os bares i e j é dada por $D_{i,j} = D_{j,i}$. Assim, de maneira geral, a probabilidade $P_{i,j}$ de Marcos ir do bar i ao bar j é dada por:

$$P_{i,j} = \frac{\frac{Q_j}{D_{i,j}}}{\sum_{k \neq i} \frac{Q_k}{D_{i,k}} + \sqrt{Q_i}} \tag{1}$$

Já a probabilidade de Marcos permanecer no mesmo bar é dada por:

$$P_{i,i} = \frac{\sqrt{Q_i}}{\sum_{k \neq i} \frac{Q_k}{D_{i,k}} + \sqrt{Q_i}} \tag{2}$$

Note que as fórmulas garantem a seguinte propriedade:

$$\sum_{j=1}^{n} P_{i,j} = 1, \forall 1 \leq i \leq n \tag{3}$$

Como Marcos bebe muito mais que a média, uma das histórias que ele mais gosta de contar quando está bêbado é a de como ele ganhou um campeonato russo de virada de *vodka*, você sabe que ele pode tomar uma quantidade arbitrariamente grande de doses. Para fins práticos, assuma que Marcos bebe uma quantidade infinita de doses numa noite.

Assim, dados como entrada os bares, suas qualidade e suas distâncias, faça um programa que descubra em qual bar Marcos estará com maior probabilidade depois de sua noite de bebedeira.

Dica: Não se esqueça que a probabilidade de Marcos estar em num bar é sempre igual a 1.

Entrada

A entrada possui vários casos de teste. Cada caso de teste começa com uma linha que contém dois inteiros n e b, $1 \leq b \leq n \leq 100$, que representam a quantidade de bares na cidade e o bar no qual Marcos começa sua noite.

Em seguida, a entrada contém n linhas com um inteiro cada, onde o inteiro da i-ésima representa a qualidade do bar i. Por fim, seguem n linhas com n inteiros cada, representando a distância entre o bar da i-ésima linha ao bar do j-ésimo inteiro da linha.

A entrada termina com uma linha "0 0" que não deve ser processada.

Saída

Seu programa deve produzir uma linha de saída para cada caso de teste. Esta linha deve conter apenas um inteiro relativo ao índice do bar onde há a maior probabilidade de Marcos estar. Em caso de empate, imprima o valor relativo ao bar de menor índice.

Exemplo

Entrada	**Saída**
3 1 1 2 3 0 1 2 1 0 2 2 2 0 4 1 3 8 5 7 0 1 2 3 1 0 5 3 2 5 0 4 3 3 4 0 0 0	3 2

Problema G
Arquipélago

fonte: `arquipelago.c`, `arquipelago.cpp` *ou* `arquipelago.java`

Descrição

Você foi encarregado de dispor os navios da frota naval do seu país para proteger um arquipélago. O mapa do arquipélago é dividido em quadrantes e cada quadrante de água precisa ser vigiado por pelo menos um navio.

As condições para a distribuição dos navios são as seguintes:

- Cada navio consegue monitorar os quadrantes nas direções horizontais e verticais do quadrante do navio, a não ser que sua visão seja bloqueada por um quadrante de terra.
- Navios somente podem ser colocados em quadrantes de água.
- Alguns quadrantes de terra são portos e possuem necessidades especiais, devendo ser guardados por uma certa quantidade de navios em quadrantes adjacentes a eles, a norte, sul, leste ou oeste.
- A fim de evitar fogo amigo, navios não podem ser capazes de vigiar quadrantes de água contendo outro navio da sua frota.

Encontre uma disposição de navios que satisfaça as restrições, se possível.

Entrada

A entrada contém vários casos de teste. A descrição de cada caso de teste começa com uma linha contendo dois inteiros $2 \leq n \leq 50$ e $2 \leq m \leq 50$ separados por um espaço, onde n representa o número de linhas do mapa e m o número de colunas.

Em seguida, seguem n linhas contendo m caracteres cada que descrevem o mapa do arquipélago. O caracter '`.`' (ponto) marca um quadrante de água. Quadrantes de terra são representados pelo caracter 'X'. Portos, que devem ter uma quantidade específica de navios ancorados em suas adjacências, são representados por um número entre 0 e 4 cujo valor é exatamente a quantidade de navios que devem estar nas adjacências.

Uma linha com os inteiros $n = m = 0$ marca o final da entrada e não deve ser tratada.

Saída

Para cada caso de teste, caso seja possível encontrar uma configuração satisfazendo as restrições acima, seu programa deve imprimir na saída padrão uma cópia do mapa dado como entrada onde os quadrantes onde ficarão os navios estão marcados pelo caracter 'N'. Caso contrário, uma única linha contendo "IMPOSSIVEL" deve ser impressa.

Caso exista mais de uma solução viável, qualquer uma delas será aceita.

Exemplo

Entrada	Saída
4 4 ...1 X... X... ..2. 7 7 ....2X. X..3... 1...... .X...3. ......X ...0..X .XX.... 0 0	..N1 X... X... .N2N ...N2X. X.N3N.. 1....N. NX.N.3N .....NX .N.0..X .XX...N

Problema H
O Jogo

fonte: `ojogo.c`, `ojogo.cpp` *ou* `ojogo.java`

Descrição

Todas as pessoas do mundo estão jogando O Jogo, mesmo que não saibam disso. Aliás, as pessoas que não sabem dele, estão ganhando! O Jogo é um jogo muito simples, tendo apenas duas regras:

- Toda vez que alguém pensa no jogo, perde.
- Toda vez que alguém perde, deve alertar todos a sua volta que ele perdeu.

Bom, você acaba de perder O Jogo... Mas não se preocupe, pois assim que esquecê-lo, voltará a jogá-lo!

Obviamente, um jogo como esse nunca terá um vencedor já que as regras somente definem um critério de perda. Contudo, você como funcionário da maior rede de micro-blogging do mundo e inconformado de não haver um vencedor neste jogo resolveu criar um prêmio para o usuário da sua rede que menos pensou no jogo.

Para alcançar seu objetivo, você terá que analisar o histórico de todos os eventos que aconteceram na rede. O funcionamento da sua rede é baseado numa arquitetura publish/subscribe e conta com as seguintes três primitivas:

- SUBSCRIBE A, B - usuário A passa a seguir B, ou seja, recebe mensagens dele
- UNSUBSCRIBE A, B - usuário A deixa de seguir B
- PUBLISH A - usuário A publica uma mensagem a todos que o seguem

As mensagens de PUBLISH já fora filtradas e só aparecem no histórico aquelas que anunciam O Jogo, fazendo com que todos que a recebem lembrem-se dele. Parece que nem todas as pessoas obedecem as regras do jogo, sendo comum que alguém receba uma mensagem falando sobre O Jogo e não avise aos seus amigos que lembrou do jogo. Contudo, tanto quem publica uma mensagem quanto quem a recebe lembra do jogo.

O seu histórico é dado em texto puro com uma primitiva por linha. Em toda linha, a primitiva é precedida por um inteiro, marcando o dia no qual a primitiva foi trocada. Nem todo dia há troca de mensagens. Como exemplo, é dado o trecho abaixo:

```
123 SUBSCRIBE alice, bob
124 PUBLISH bob
124 PUBLISH alice
126 PUBLISH alice
```

No exemplo acima, no dia 123, alice passa a seguir bob sem que bob siga alice. No dia seguinte, bob lembra do jogo e acaba avisando alice através de uma mensagem. Nesse mesmo dia, alice publica uma mensagem sobre o jogo, lembrando também seus seguidores. Já no dia 126, somente alice lembra do jogo. Neste caso, alice lembrou do jogo dois dias e bob apenas um.

Como sua rede possui muitos usuários, você não sabe exatamente quantos usuários aparecem no histórico e deve, portanto, estar preparado para suportar vários. Cada usuário é representado por uma string de no máximo 15 caracteres, somente letras minúsculas são usadas.

Dado o histórico da rede em ordem cronológica, inclusive em relação às mensagens trocadas no mesmo dia, descubra qual usuário lembrou menos do jogo, ou seja, quem passou mais dias sem lembrar do jogo.

Entrada

A entrada contém vários casos de teste. Cada caso de teste começa com um número $1 \leq n \leq 10.000$ que representa a quantidade de mensagens no histórico. Essa linha é seguida de n linhas contendo primitivas como descrito anteriormente. O número de usuários distintos em um histórico será no máximo 10.000.

A entrada termina com um caso de teste contendo um valor $n = 0$ e não deve ser processado.

Saída

A saída deve conter uma linha para cada caso de teste. Para cada caso de teste, escreva na saída o usuário que pensou no jogo por menos dias. Em caso de empate, imprima todos os usuários em ordem alfabética e separados por um espaço.

Exemplo

Entrada	**Saída**
4 1 SUBSCRIBE ana, carlos 1 SUBSCRIBE ana, sabrina 2 PUBLISH sabrina 3 SUBSCRIBE carlos, ana 6 1 SUBSCRIBE antonio, barbara 2 SUBSCRIBE barbara, cristina 3 SUBSCRIBE cristina, antonio 4 PUBLISH antonio 4 PUBLISH barbara 4 PUBLISH cristina 0	carlos antonio barbara cristina

Problema I
Soma de números consecutivos

fonte: `somacons.c`, `somacons.cpp` *ou* `somacons.java`

Descrição

Dado n, dizer se n pode ser obtido como a soma de 2 ou mais números inteiros positivos consecutivos.

Entrada

Cada linha da entrada consistirá de um único inteiro $1 < n \leq 1.000.000.000$. A entrada termina com $n = 0$, que não deve ser processado.

Saída

Para cada da linha da entrada, uma linha de saída deve ser produzida, contendo a palavra "SIM" caso n possa ser obtido da forma descrita e "NAO" caso contrário.

Exemplo

Entrada	**Saída**
7 8 9 10 0	SIM NAO SIM SIM

Problema J
Loteria falha

fonte: `loteria.c`, `loteria.cpp` *ou* `loteria.java`

Descrição

Muita gente sonha em ganhar dinheiro fácil. Algumas pessoas tentam fazer isso através da loteria. Compram jogos como as "raspadinhas" e bilhetes de loteria aguardando sorteios multimilionários.

Gilmar, um rapaz muito malandro, decidiu usar seus conhecimentos matemáticos para tentar aumentar suas chances nestes jogos de sorte. Ele comprou vários bilhetes de um mesmo tipo de raspadinha e analisou as cartelas, até que percebeu uma propriedade muito curiosa: cada raspadinha tinha impresso um número identificador do cartão que permitia a ele ter noção das chances de ser premiado.

De cada 10 cartões que comprou na banca, aproximadamente 1 ou 2 vinham premiados de alguma forma: no mínimo uma outra raspadinha grátis ele ganhava, ou um prêmio simbólico em dinheiro. Quando aplicou seu método para escolher que cartões comprar, percebeu que de cada 10 cartões, em média 8 continham algum prêmio!

Como a tarefa é muito cansativa para ser feita manualmente, ele pensou que você, amigo de longa data, poderia ajudá-lo com um programa que, dado o número identificador do cartão, diz se ele faz parte do conjunto de cartões com maior chance de prêmio. O truque é procurar os cartões cujo número identificador seja múltiplo de 42. Você está apto a ajudar seu colega nesta empreitada?

Entrada

A entrada é composta por vários casos de teste, um por linha.

Cada caso de teste contém um inteiro n de até 30 dígitos decimais.

A entrada termina com $n = 0$. Este caso não deverá ser processado.

Saída

Para cada caso de teste haverá uma linha na saída. Caso o número identificador do cartão seja um múltiplo de 42, imprima "PREMIADO". Caso contrário, imprima "TENTE NOVAMENTE".

Exemplo

Entrada	**Saída**
42 17283940536172938433 17283940536172938432 10000000000000000000 0	PREMIADO TENTE NOVAMENTE PREMIADO TENTE NOVAMENTE

Problema K
Centros de Distribuição Gêmeos

fonte: `gemeos.c`, `gemeos.cpp` *ou* `gemeos.java`

Descrição

A Empresa Rural Internacional (ERI) está buscando expandir seus negócios. Para isso, ela precisa encontrar uma nova cidade para instalar um novo centro de distribuição. Entretanto, como algum centro pode ficar inoperante, como aconteceu no último verão devido às chuvas, ela quer se certificar de que não ficará sem distribuir seus produtos caso alguma cidade fique incomunicável. Para isso, ao invés de instalar seu novo centro de distribuição em uma única cidade, ela escolherá duas cidades vizinhas, de forma a garantir que haja redundância, caso algum problema ocorra com uma das localidades.

Simonal, diretor de logística da ERI, conhece bem a região e seus possíveis clientes. Dada uma cidade, Simonal sabe a área de alcance de sua empresa, ou seja, as cidades alcançadas caso um novo centro de distribuição seja instalado nessa cidade. A fim de que a redundância desejada seja satisfeita, as duas cidades candidatas a receber o novo centro de distribuição devem ter a mesma área de alcance.

Você, estagiário buscando um aumento de salário para comprar a mais nova expansão do seu MMORPG favorito, quer agradar o seu chefe, Dionísio Cabeço Fechantes, também conhecido como Dudu, e, usando os dados obtidos com Simonal, quer mostrar o quão bom você é em computação e contar de quantas formas distintas estas cidades podem ser escolhidas.

Entrada

A entrada é composta por diferentes casos de teste. A primeira linha de cada caso de teste contém n e m, $n \leq 1000$ e $m \leq \frac{n(n-1)}{2}$, onde n é o número de cidades da região. A seguir, m linhas, contendo dois inteiros distintos a e b cada, $1 \leq a, b \leq n$, indicando que a faz parte da área de alcance de b e vice-versa.

A entrada termina com $n = m = 0$. Essa linha não deve ser processada.

Saída

Para cada caso de teste, seu programa deve imprimir uma linha contendo o número de diferentes pares de cidades que podem receber novos centros.

Exemplo

Entrada	**Saída**
3 2 1 2 2 3 3 3 3 2 1 3 2 1 5 5 1 2 1 5 2 3 2 5 4 5 0 0	0 3 0

Problema L
Soma de quantidade prima de primos consecutivos

fonte: `somaprima.c`, `somaprima.cpp` *ou* `somaprima.java`

Descrição

Um número k é primo se e somente se $k \geq 2$ e k tem exatamente dois divisores, 1 e k. Dois primos k e l são consecutivos se e somente se $k < l$ e não existe um primo p tal que $k < p < l$.

Dado n, dizer se n pode ser obtido como a soma de q primos consecutivos, onde q é primo.

Entrada

A entrada consiste de vários casos de teste. Cada caso de teste consiste de uma única linha contendo um inteiro $2 \leq n \leq 1.000.000$.

A última linha da entrada conterá um inteiro $n = 0$. Esse caso não deve ser processado.

Saída

Para cada linha da entrada, uma linha da saída deve ser gerada, contendo "SIM" caso n possa ser escrito da forma desejada ou "NAO", caso contrário.

Exemplo

Entrada	**Saída**
5 6 7 8 9 10 0	SIM NAO NAO SIM NAO SIM