Problema A

Conversão de RAs

Arquivo fonte: `ra.{c | cpp | java}`

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

O Centro Paula Souza vem passando por significativas modificações nos últimos anos e uma delas se refere ao software para controle acadêmico que foi implantado de forma unificada para todas as Fatecs. O SIGA (esse é o nome do sistema de gestão) padronizou o registro acadêmico em todas as unidades, além de oferecer uma série de recursos muito interessantes para os professores, coordenadores de curso e secretários acadêmicos. Uma das mudanças mais perceptíveis proporcionadas pela implantação do SIGA foi a mudança no formato dos números de matrícula dos alunos, os chamados RAs. Antigamente esse dado variava grandemente entre as unidades, agora todas usam o mesmo formato, de 13 dígitos numéricos, na forma `FFFCCCAASTNNN`, onde:

`FFF` = Código numérico da Fatec (Sorocaba é 003, São Paulo é 002, etc)

`CCC` = Código do curso:

048 – Análise e Desenvolvimento de Sistemas

061 – Sistemas Biomédicos

073 – Eletrônica Automotiva

074 – Logística

080 – Polímeros

081 – Processos Metalúrgicos

099 – Projetos Mecânicos

100 – Fabricação Mecânica

`AA` = Ano de ingresso do aluno

`S` = Semestre de ingresso do aluno (1=primeiro semestre, 2= segundo semestre)

`T` = Turno do curso (1=manhã/matutino, 2=tarde/vespertino, 3=noite/noturno)

`NNN` = Número sequencial do aluno

Por exemplo, o novo RA `0030481321099` significa:

Fatec:	003 (Sorocaba)
Curso:	048 (Análise e Desenvolvimento de Sistemas)
Ano de ingresso:	13 (2013)
Semestre de ingresso:	2 (Segundo semestre)
Turno:	1 (manhã)
Número sequencial:	099

Na Fatec de Sorocaba, por exemplo, o RA antigo era composto por 8 caracteres, na forma `CCAASNNN`, onde:

CC = Sigla do curso:

AD – Análise e Desenvolvimento de Sistemas (manhã)

AN – Análise e Desenvolvimento de Sistemas (noite)

SD – Sistemas Biomédicos (manhã)

LT – Logística (tarde)

PL – Polímeros (tarde)

PD – Projetos Mecânicos (manhã)

PN – Projetos Mecânicos (noite)

OD – Fabricação Mecânica (manhã)

ON – Fabricação Mecânica (noite)

AA = Ano de ingresso do aluno

S = Semestre de ingresso do aluno (1=primeiro semestre, 2= segundo semestre)

NNN = Número sequencial do aluno

Ainda temos, em Sorocaba, alunos remanescentes do período em que, no ato da matrícula, o RA era gerado no formato antigo. Sua equipe foi designada para gerar um programa que, para um RA antigo da Fatec de Sorocaba informado, gere uma versão correspondente no novo formato de 13 caracteres, aproveitando nos últimos três dígitos a informação correspondente do RA antigo. Por exemplo, o RA antigo `AD132099` seria o equivalente ao RA novo `0030481321099`.

Entrada

Inicialmente um valor *N* é informado, indicando a quantidade de casos de teste a serem processados. Seguem-se *N* linhas, cada uma contendo uma *string* de 8 caracteres representando um RA antigo válido da Fatec de Sorocaba. Não serão fornecidos RAs antigos dos cursos de Eletrônica Automotiva e de Processos Metalúrgicos, pois são cursos muito recentes e já foram implantados com os novos RAs.

Saída

Para cada caso de teste, imprima o novo RA de 13 caracteres correspondente ao RA antigo lido.

Exemplos

Entrada:	**Saída:**
9 AD101001 AN122035 SD111111 LT121221 PL091252 PD082124 PN071255 OD121022 ON121212	0030481011001 0030481223035 0030611111111 0030741212221 0030800912252 0030990821124 0030990713255 0031001211022 0031001213212

Solução

Problema simples de manipulação de strings, envolvia receber uma cadeia contendo o RA original, decompô-la em suas partes básicas (curso, ano de ingresso, semestre de ingresso e número sequencial) e, com base na especificação constante no enunciado, montar o RA correspondente no novo formato. Alguma habilidade adicional sobre os recursos de sua linguagem de programação para manipular a string de entrada e a formatação de saída poderia simplificar bastante a codificação.

Problema B

Sistema Antimísseis

Arquivo fonte: `misseis.{c | cpp | java}`

Autor: Anderson V. de Araújo (Fatec São José dos Campos)

Dois países do Oriente Médio entraram em conflito devido a uma disputa territorial. Um dos países desenvolveu um projeto para implementação de um sistema antimísseis para se defender dos ataques do país vizinho. O país já tem uma ideia de como fazer isso, eles vão usar um radar que detecta os mísseis até uma certa distância dele e, depois de detectados, eles devem ser ordenados a partir do mais próximo até o mais distante. Para implementar o sistema, o país convidou você para codificar a função de detecção do radar.

Para implementar o sistema você deve verificar qual é ordem em que os mísseis atacantes devem ser abatidos, mas isso somente para os mísseis que estiverem dentro do perímetro de alcance do sistema de defesa. As posições dos mísseis são definidas por um ponto de coordenadas inteiras X e Y para cada um dos mísseis.

A imagem abaixo apresenta um exemplo com 5 mísseis (B, C, D, E e F), onde apenas os mísseis E, F e B, nesta ordem, podem ser alcançados pelo sistema de defesa. As bordas do limite do radar também detectam os mísseis, assim como B no exemplo. Mísseis à mesma distância do radar, devem ser ordenados comparados pela ordem de leitura, sendo que o primeiro míssil lido vem primeiro na no vetor de saída.

No exemplo, o ponto A representa a posição do radar do sistema antimísseis e o raio de alcance do radar é 2.

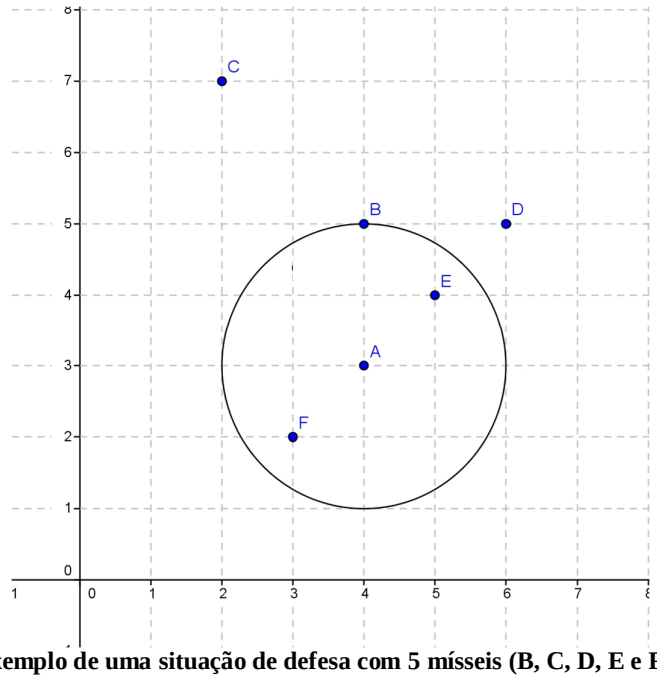


Figura 1. Exemplo de uma situação de defesa com 5 mísseis (B, C, D, E e F) e radar A.

Para a imagem apresentada os valores de entrada são iguais aos apresentados no primeiro caso de testes dos exemplos.

Entrada

A primeira informação a ser lida é N (0 < N < 100), que corresponde ao número de situações de teste de entrada. Para cada situação, devem ser lidos: a posição do radar (Xa e Ya), o raio de alcance do radar (0 < R < 100), o número de misseis (0 < M < 26) a serem verificados, e uma sequência de M posições correspondentes aos mísseis no mapa. Sendo cada informação em uma linha separada. Para cada míssil, devem ser lidos 2 inteiros, Xm e Ym, indicando a posição do mesmo. Todas as posições são inteiros positivos maiores ou iguais a zero e menores que 100. As posições estão separadas por um espaço em branco.

Saída

Para cada situação consultada deve ser impressa uma linha que representa a sequência dos mísseis a serem abatidos que estão dentro do perímetro do radar em ordem de proximidade. Os mísseis que estiverem na borda limite de detecção do radar também devem ser contados. Cada míssil deve ser identificado por uma letra maiúscula começando de B. Caso um míssil esteja a mesma distância de outro míssil, o que foi lido primeiro tem precedência na saída. Todas as letras devem ser separadas por um espaço em branco, apenas entre as letras. Caso não exista nenhum míssil no radar do sistema antimísseis deve-se imprimir exatamente o texto: OUT OF RANGE.

Exemplos

Entrada:	**Saída:**
2 4 3 2 5 4 5 2 7 6 5 5 4 3 2 1 1 1 3 2 2 3 1 0 2	E F B OUT OF RANGE

Solução

Uma possível solução do problema seria:

Após fazer as leituras das entradas, calcular a distância de cada míssil para o radar. Se a distância for menor que o raio de alcance do radar de entrada, armazenar a distância e a letra correspondente em um mapa (chave:letra; valor:distância).

Criar uma lista ordenada das distâncias. Para cada posição da lista, buscar no mapa as letras correspondentes a distância e armazene em uma outra lista, de letras. Remover do mapa a primeira letra da lista de letras e exibir na saída padrão.

Caso a lista esteja vazia, nenhum míssil dentro do alcance do radar, escreva “OUT OF RANGE”.

Problem C

Koch Snowflake

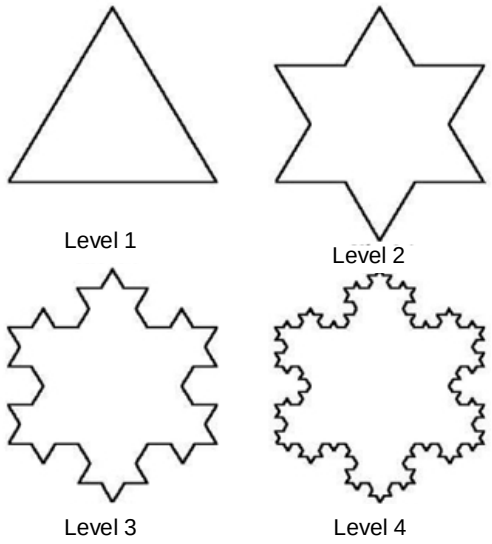
Source file: `koch.{c | cpp | java}`

Author: Eduardo Sakaue (Fatec São José dos Campos)

The Koch snowflake is a mathematical curve and one of the earliest fractal curves to have been described. It is based on the Koch curve, which appeared in a 1904 paper entitled:*"Sur une courbe continue sans tangente, obtenue par une construction géométrique élémentaire"*, by the Swedish mathematician **Helge von Koch.**

The Koch Snowflake can be constructed by starting with an equilateral triangle, and then recursively altering each line segment *s* by dividing it in three segments of size s/3, forming a new triangle. Then, each segment can be divided infinitely.

The following figure shows the first four levels of the Snowflake.



Helge von Koch, in his epoch, could not create a machine able calculate the total area of the Snowflake, but now you have all the necessary technology.

To calculate the area of the equilateral triangle, use the formula:

$$\frac{s^2\sqrt{3}}{4}$$

For all the calculations, consider using floating-point number with double precision.

Input

The input consists of several test cases. Each line has a number *S* ($0 < S < 100$), a floating-point number with double precision that corresponds to the size of each segment of the triangle and the number of levels *N* ($0 < N < 16$) for which the Snowflake must evolve.

The final input case is defined by *S* and *N* equal to zero (0 0).

Output

Print a line for each input with the total area of the Snowflake obtained considering the precision of 8 numbers after the decimal point.

Examples

Input:	**Output:**
1 3 15.5 6 2.4827 11 10 10 30.627263 8 0 0	0.64150030 165.36764545 4.26992383 69.26445269 649.05090453

Solução

1. Calcular a área de 1 triângulo
2. Cada segmento de reta com tamanho *s* é dividido em 4 segmentos de tamanho *s*/3. Em cada um deste segmentos nasce um novo triângulo.
3. Calcular a área de 1 novo triângulo.
4. Calcular a quantidade de novos triângulos por iteração.
5. Somar no total
6. Exibir o total formatado %.8f

Problema D

Robótica Móvel

Arquivo fonte: `robotica.{c | cpp | java}`

Autor: Julio Fernando Lieira (Fatec Lins)

Robótica móvel estuda técnicas para permitir a locomoção de veículos autônomos em ambientes que podem, ou não, ser conhecidos a priori e que contenham obstáculos estáticos ou móveis. Para operar neste tipo de ambiente o robô deve ser capaz de adquirir e utilizar conhecimento sobre o ambiente, estimar sua posição dentro da área a ser explorada, reconhecer obstáculos e, caso o ambiente seja dinâmico, responder em tempo real às situações que possam ocorrer.

Dentre as várias técnicas existentes, a transformada de distância é uma técnica tradicionalmente utilizada para gerar rotas para robôs móveis em ambientes pré-mapeados. Esta técnica utiliza uma matriz para mapeamento do ambiente a ser explorado, onde cada célula é identificada como sendo uma região livre ou ocupada (obstáculo). As células livres serão as utilizadas para determinação da rota.

A Figura 1(a) mostra um ambiente mapeado em uma matriz, onde os obstáculos são representados por células pintadas com cinza e as células não preenchidas são células livres (por onde o robô pode se locomover). A Figura 1(b) mostra a Matriz Transformada de Distância gerada a partir do ponto destino com valor **0** (valor zero, onde se deseja chegar), de coordenadas (3,8). O valor em cada célula desta Matriz representa o custo para se chegar até a célula destino e reflete, neste exemplo, exatamente a distância, em células, para se atingir a célula destino. Estando em qualquer célula, o robô pode orientar sua trajetória até o ponto destino apenas escolhendo a célula vizinha com menor custo (valor). O conceito de vizinhança de uma célula K é ilustrado na Figura 2.

Como exemplo, para se chegar ao destino (3,8), considerando que o robô inicialmente esteja na célula de linha 7 e coluna 2 (7,2), primeiro ele deve se deslocar para a célula vizinha de menor custo seguindo em frente para a célula na linha 7 e coluna 3 (7,3), ou pela diagonal para a célula (6,3), ambas de custo 5. Supondo que tenha se deslocado em frente para a célula (7,3), agora ele deve se locomover na diagonal para a célula (6,4) que é a célula vizinha de menor custo (4). Estando na célula (6,4) ele deve se locomover novamente pela diagonal para a célula (5,5) de custo 3. Depois deve ir para a célula (4,6) de custo 2. Nesta célula tem-se duas opções: à direita para a célula (4,7) ou pela diagonal para (3,7), ambas com custo 1. Estando em uma célula de custo 1, o próximo deslocamento atingirá o destino.

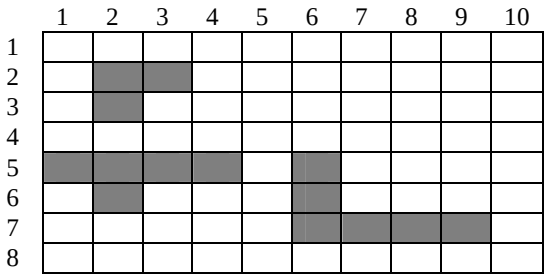


Figura 1(a) Ambiente mapeado em uma matriz.

	1	2	3	4	5	6	7	8	9	10
1	7	6	5	4	3	2	2	2	2	2
2	7			4	3	2	1	1	1	2
3	7		5	4	3	2	1	**0**	1	2
4	7	6	5	4	3	2	1	1	1	2
5					3		2	2	2	2
6	7		5	4	4		3	3	3	3
7	7	6	5	5	5					4
8	7	6	6	6	6	6	7	6	5	5

Figura 1(b) Matriz Transformada de Distância para o destino (3,8).

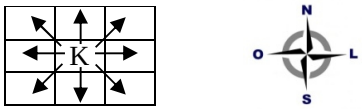


Figura 2. Vizinhança de uma célula

Sua missão é ajudar o robô a chegar ao destino, dada a Matriz Transformada de Distância e as coordenadas de uma célula origem. Por ainda estar em fase de testes, o robô possui algumas limitações que podem influenciar na escolha do melhor caminho e que devem ser consideradas:

1. O robô **não** consegue se deslocar de uma célula para outra pela diagonal;
2. Quando se depara com uma situação em que deve escolher entre duas células vizinhas de mesmo custo, ele deve escolher a célula no mesmo sentido de deslocamento, evitando manobras de 90 graus;
3. O sentido inicial do robô (Norte, Sul, Leste, Oeste) será informado na entrada de dados.

Com tais considerações, o exemplo anterior para se chegar ao destino (3,8), considerando que o robô inicialmente esteja na célula (7,2) e o sentido inicial **Leste**, seria:

(7,2) - (7,3) - (7,4) - (6,4) - (6,5) - (5,5) - (4,5) - (4,6) - (4,7) - (4,8) - (3,8)

Entrada

A entrada inicia com dois valores inteiros L e C separados por um espaço, que correspondem respectivamente ao número de linhas e colunas da Matriz Transformada de Distância. Em seguida, é dada a Matriz Transformada de Distância, composta por L linhas, cada linha contém C valores inteiros V separados por um espaço. Cada valor V corresponde ao custo da célula para se chegar ao destino, ou o valor -9 que representa um obstáculo, ou o valor 0 (zero) que representa a célula destino. Após as linhas da matriz, cada linha seguinte é composta por dois valores inteiros I e J e um caractere S separados por um espaço em branco que correspondem respectivamente à linha (I) e coluna (J) da célula origem na qual o robô se encontra e o sentido (S) inicial do robô (N para Norte, S para Sul, L para Leste e O para Oeste). A entrada termina com o fim do arquivo de entrada.

Restrições

$3 \leq L,C \leq 20$
$1 \leq I,J \leq 20$
$0 \leq V \leq 1000$
$S \in \{N,S,L,O\}$

Saída

Para cada célula de origem, imprima na saída uma linha que representa a soma do **valor da coordenada da coluna** (não o valor do custo) de cada célula percorrida pelo robô desde a origem até o destino (inclusive as colunas das células origem e destino) respeitando as limitações do robô descritas no enunciado.

Exemplos

Entrada:	**Saída:**
8 10 7 6 5 4 3 2 2 2 2 2 7 -9 -9 4 3 2 1 1 1 2 7 -9 5 4 3 2 1 0 1 2 7 6 5 4 3 2 1 1 1 2 -9 -9 -9 -9 3 -9 2 2 2 2 7 -9 5 4 4 -9 3 3 3 3 7 6 5 5 5 -9 -9 -9 -9 4 7 6 6 6 6 6 7 6 5 5 7 2 L 8 7 O 3 1 N	57 67 52

Solução

Uma possível solução para o problema seria:

1- Colocar os dados da Matriz Transformada em uma matriz de inteiros;
   Nesta entrada, já identificar o Destino (valor 0), guardando suas coordenadas linha e coluna em variáveis;
2- Para cada entrada das coordenadas de Origem, iniciando nesta célula Origem, percorrer a matriz até chegar ao Destino.
3- Para cada célula para onde se desloca, procurar nas células vizinhas qual célula tem o menor custo (valor).
4- Note que, ao estar em uma célula onde existe mais de um vizinho com mesmo custo, considerar o vizinho que está no mesmo sentido do deslocamento.
   Para tanto, deve-se manter uma variável que guarda o sentido inicial e que muda a cada célula percorrida.
5- Note que, embora o sentido original mande ir para um determinado sentido, ele só deve ser respeitado se houver empate no menor custo das células vizinhas. Ou seja, ir para a célula vizinha com menor custo mesmo que não esteja no mesmo sentido indicado.
6- Ao se deslocar para uma nova célula, atualizar a variável que guarda o sentido com o novo sentido deste deslocamento.
7- Note que deve-se somar o valor da coluna de cada célula no trajeto, e não o custo da célula.

Problema E

Espionagem

Arquivo fonte: `espionagem.{c | cpp | java}`

Autor: Leandro Luque (Fatec Mogi das Cruzes)

Após a divulgação das denúncias feitas pelo analista de sistemas Edward Snowden, ex-funcionário da CIA e da NSA, sobre o programa de espionagem do governo estadunidense, o governo brasileiro resolveu se movimentar e investir em segurança digital e um programa de espionagem próprio.

Entre os primeiros objetivos do programa está o reconhecimento do possível conteúdo de um conjunto de dados criptografados obtidos pelo governo. Os alvos da investigação, que não podem ser citados no enunciado por questões de segurança dos elaboradores da prova, usaram uma estratégia para dificultar a espionagem:

- As mensagens trocadas por eles eram escritas em fitas circulares – de tal forma que o final da mensagem estava diretamente conectado ao início dela;
- O conteúdo da mensagem foi criptografado com a Cifra de César (não é o de Sorocaba), um tipo de criptografia de substituição na qual cada letra do texto original é substituída por outra, que se apresenta C posições à frente desta – o número C é conhecido como chave. Assim, por exemplo, caso seja adotada uma chave igual a três (C=3), as ocorrências de 'A' no texto serão substituídas por 'D', as de 'B' serão substituídas por 'E', e assim por diante (ver Figura 1).

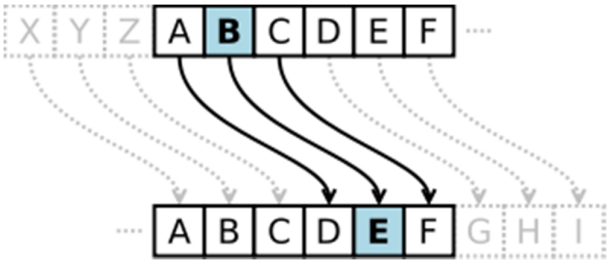


Figura 1: Exemplo de criptografia com a Cifra de César (C=3). Fonte: Wikipédia.

Desta forma, uma frase como "O PROBLEMA C DA MARATONA" seria lida como "R SUREOHPD F GD PDUDWRQD", caso a chave C fosse igual a 3 e, por sorte, a mensagem criptografada fosse lida a partir da posição correta da fita. Porém, se a fita fosse lida a partir de outra posição, como a partir da posição 10, o conteúdo dela seria "D F GD PDUDWRQDR SUREOHP".

Um erro cometido pelos alvos da investigação foi que eles usaram nas mensagens apenas letras maiúsculas e espaços (e os espaços não foram criptografados), facilitando a identificação do possível conteúdo da mensagem.

Você foi contratado pela ABIN para, dada uma frase procurada e uma lista de mensagens criptografadas, definir se alguma delas pode conter a frase. É importante notar que você não sabe a partir de que posição a fita foi lida para gerar a mensagem criptografa em questão e nem sabe a chave que foi utilizada para criptografá-la.

Entrada

A entrada é composta por vários casos de teste. A primeira linha de cada caso de teste contém uma frase que deve ser buscada pelo algoritmo com, no máximo, 100 caracteres (entre letras maiúsculas e espaços). A próxima linha contém um número inteiro N ($0 < N < 100$) com o número de mensagens criptografadas onde a frase deverá ser procurada. As N linhas seguintes contém uma mensagem criptografada cada com, no máximo, 100 caracteres (entre letras maiúsculas e espaços).

Saída

Para cada caso de teste, você deve imprimir um caractere 'S', sem aspas e maiúsculo, caso alguma das mensagens criptografadas puder conter a frase procurada. Imprima um caractere 'N', sem aspas e maiúsculo, caso contrário.

Exemplos

Entrada:	**Saída:**
O PROBLEMA C DA MARATONA 3 P QSPCSFNB D EP NBSBEPOB T UWTGWJRF H IF RFWFYTSF D F GD PDUDWRQDR SUREOHP MARATONA 3 P QSPCSFNB D EP NBSBEPOB T UWTGWJRF H IF RFWFYTSF D F GD PDUDWRQDR SUREOHP	S S

Explicação do Primeiro Caso de Teste

No primeiro caso de teste, existem 3 mensagens. A primeira delas envolveu a criptografia da frase “O PROB**R**EMA C D**O** MARA**D**ONA” com chave C=1. A segunda envolveu a criptografia da frase “O PROB**R**EMA C DA MARATONA” com chave C=5. Claramente, nenhuma das duas contêm a frase “O PROBLEMA C DA MARATONA” – veja as diferenças nos caracteres destacados. A terceira mensagem, no entanto, envolve a criptografia da frase que estamos procurando com chave C=3, resultando na mensagem “R SUREOHPD F GD PDUDWRQD”. Porém, a fita começou a ser lida a partir do 10º caractere e, portanto, ficou como “D F GD PDUDWRQDR SUREOHP”.

O seu algoritmo não precisa descobrir a frase que existia originalmente em cada mensagem criptografada, mas apenas se esta mensagem pode conter a frase procurada.

Solução

O problema "Espionagem" pode ser resolvido de diferentes formas. Uma delas envolve encriptar a frase com todas as chaves possíveis (são 26, de 0 a 25) e comparar a frase encriptada com as mensagens fornecidas. Como não se sabe onde a fita inicia em cada mensagem e para não ter que testar todas as possibilidades de posição de início da fita, pode-se simplesmente concatenar cada mensagem com ela mesma antes de procurar a frase encriptada dentro da mensagem. Para a comparação, um String.indexOf (Java), strstr (C++), ou outro equivalente, pode ser utilizado.

Outra solução possível seria gerar, para a frase e para cada mensagem, uma sequência representativa da distância entre os caracteres consecutivos e procurar a sequência da frase dentro da sequência de cada mensagem. Nesta solução, não é necessário encriptar a frase com todas as chaves possíveis. Caso você deseje estudar problemas similares, veja o problema "Plagio Musical" da 1a fase da maratona brasileira de 2010.

Problema F

Números Triangulares

Arquivo fonte: `triang.{c | cpp | java}`

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Ovídio é um administrador de redes fissurado por segurança. As redes por ele administradas são muito seguras, pelo menos tão seguras quanto a tecnologia de redes permite, embora muito chatas de usar: as senhas precisam ter no mínimo 10 caracteres de comprimento, precisam combinar letras maiúsculas, minúsculas e dígitos numéricos (pelo menos um de cada) e, para desespero de seus usuários, precisam ser trocadas a cada 15 dias, sendo que a nova senha não pode ser igual a nenhuma das últimas 50 senhas utilizadas. "É para garantir", diz ele. Não sabemos bem o que fica garantido nessa paranoia toda, mas enfim, a rede é segura. Ovídio também faz uso de diversos outros sistemas, na condição de usuário comum, portanto, e geralmente fica decepcionado com a liberalidade com que a composição das senhas é tratada. Metódico, ele estipulou para si mesmo que suas senhas sempre terão no mínimo dois e no máximo três números triangulares. Um número triangular é aquele que pode ser composto na forma de um triângulo, como por exemplo o número 10 (suponha que cada '*' representa a unidade):

```
    *
   * *
  * * *
 * * * *
```

O número decimal 10 possui dez unidades, que podem ser agrupadas da forma acima, compondo um triângulo (viu que temos 10 '*' na figura e que ela forma um triângulo?). A ideia dos números triangulares é atribuída ao famoso matemático Gauss, que a concebeu quando ainda era criança. Os primeiros cinco números triangulares são mostrados na figura abaixo. Observe que o topo sempre possui uma unidade, o nível imediatamente abaixo possui duas unidades, o próximo nível possui três unidades, e assim por diante. Se não for possível construir um arranjo nesses termos com as unidades que formam o número em questão, então concluímos que esse número não é triangular. O número 8, por exemplo, não é triangular.

```
1     3     6      10       15
*     *     *      *        *
     * *   * *    * *      * *
          * * *   * * *    * * *
                 * * * *  * * * *
                          * * * * *
```

Sua tarefa é, dado um inteiro natural *N*, determinar o *N*-ésimo número triangular, para ajudar Ovídio a manter seu padrão de senhas numéricas.

Entrada

A entrada possui vários casos de teste, cada um composto por um valor inteiro *N* (0<*N*<=45000). As entradas são encerradas com um valor *N* = 0, que não deverá ser processado.

Saída

Para cada caso de teste imprima uma linha contendo o *N*-ésimo número triangular.

Exemplos

Entrada:	**Saída:**
1 2 3 4 10 0	1 3 6 10 55

Solução

O problema mais fácil da prova, podia ser resolvido por simulação direta, gerando os números triangulares até atingir o *N*-ésimo valor pretendido, ou então bastava aplicar a fórmula (*N**(*N*+1))/2, onde *N* indica qual a ordem do valor buscado (1 para primeiro, 2 para segundo, etc), que era exatamente o valor desejado.

Problema G

Distância dos pontos

Arquivo fonte: `pontos.{c | cpp | java}`

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

O professor de Matemática do Zequinha propôs um desafio à classe. Entregou um papel quadriculado com vários pontos marcados e agora os alunos precisam interligar todos eles com segmentos de reta, desde que não produzam ciclos, ou seja, não podem formar polígonos fechados com os segmentos de reta. Ganha o desafio aquele que interligar todos os pontos perfazendo o menor comprimento total das retas.

Você, tio do Zequinha, foi então desafiado pelo próprio sobrinho a calcular o menor valor e assim ajudá-lo a vencer o desafio.

Entrada

Um valor N ($0<=N<1000$) é informado inicialmente, indicando a quantidade de casos de teste a serem avaliados. Cada caso de teste inicia-se com um inteiro P ($2<=P<=50$) representando a quantidade de pontos marcados no plano cartesiano. Em seguida, tem-se P linhas contendo dois números inteiros Xa e Ya ($-1000<=Xa, Ya <= 1000$) descrevendo a localização de um ponto em termos de suas coordenadas, onde Xa é a abscissa (eixo x) e Ya é a ordenada (eixo y). Um espaço em branco separa dois inteiros na mesma linha da entrada.

Saída

Para cada caso de teste imprima a distância mínima com que todos os pontos podem ser interligados sem formar ciclos. Use tipo real de precisão dupla para tratar a distância entre os pontos. A distância total deverá ser exibida com 4 casas depois da vírgula.

Exemplos

Entrada:	**Saída:**
2 7 1 1 2 2 2 4 1 0 -1 3 3 4 5 0 10 4 1 0 -4 6 0 8 6 3 6 6 3 7 7 1 0 6 8 2 3	11.8482 24.7746

Solução

Trata-se de um problema básico de programação com grafos. Receber as coordenadas dos pontos e armazena-las para depois montar uma matriz ou lista de adjacências com as distâncias entre os pontos, que pode ser obtida pela simples aplicação da equação da distância entre dois pontos no plano cartesiano. Com isso teremos um grafo totalmente conexo, com pesos nas arestas dado pela distância calculada anteriormente, tomando-se o cuidado de armazena-la como dado do tipo real de precisão dupla. Feito isso, bastava determinar a árvore geradora mínima do grafo, podendo usar qualquer algoritmo clássico (Prim, Kruskal, Boruvka). O valor a ser impresso para cada caso de teste corresponde à soma dos pesos das arestas contidas na árvore geradora mínima produzida.

Problema H

Resident Evil

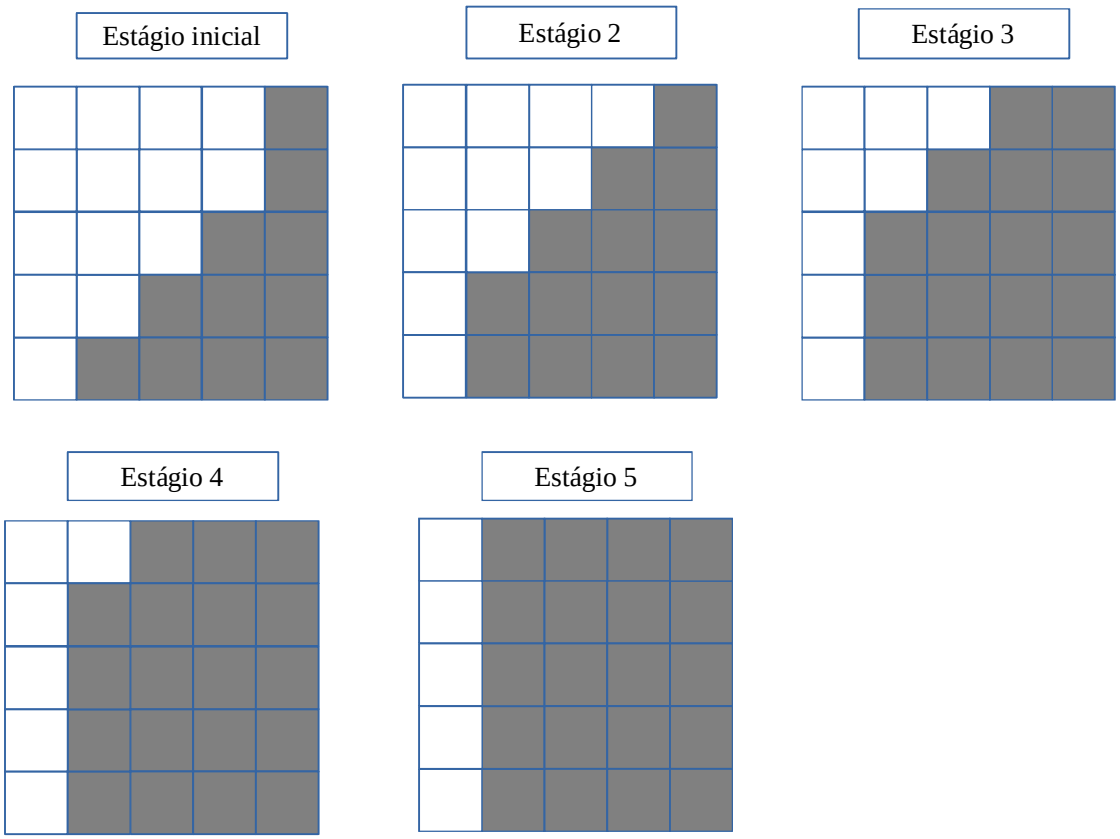
Arquivo fonte: `resident.{c | cpp | java}`

Autor: Reinaldo Arakaki (Fatec São José dos Campos)

Um vírus letal da *Umbrella Corporation* foi liberado acidentalmente, tornando mortos-vivos (zumbis) todos os seres humanos que tiveram contato com o vírus. O simples contato entre seres humanos faz com que o vírus letal se espalhe. Uma contaminação pode ser simulada através de um tabuleiro 5 x 5 (5 linhas e 5 colunas), formado por espaços de 1 cm de lado. As regras de contaminação são descritas a seguir:

- Espaços contaminados, indicados em cinza, permanecem contaminados no estágio seguinte;
- Um espaço não contaminado, indicado em branco, torna-se contaminado no estágio seguinte quando tem pelo menos dois lados comuns contaminados. Caso contrário, permanece não contaminado. Um lado comum considera apenas os vizinhos de cima, baixo, esquerda e direita, e não as suas diagonais;
- A contaminação acaba quando não é possível contaminar novos espaços.

Exemplo de uma simulação do início de contágio:



Entrada

A entrada é composta de uma sequência de matrizes de **5 linhas e 5 colunas** contendo apenas valores 0 e 1. O valor 0 indica espaço vazio e o valor 1 indica espaço contaminado. **Não existe separação** entre os valores e **nem entre as matrizes** de teste.

Saída

Para cada grupo de consultas deve ser impressa a matriz final, também com **5 linhas e 5 colunas**, indicando todos os espaços contaminados **sem espaços entre os valores**. Assim como na entrada, as matrizes de saída **não possuem separação entre elas**.

Exemplos

Entrada:	**Saída:**
00001 00001 00011 00111 01111 00110 00110 11010 00110 01000	01111 01111 01111 01111 01111 11110 11110 11110 11110 11110

Solução

Uma possível solução para o problema seria:

Ler as entradas, percorrer a matriz armazenando e contando o número de blocos que tem mais de dois vizinhos com o valor 1. Se o número de blocos contaminados for maior que 0, pintar todos os contaminados e repetir a operação até não existirem mais possibilidades para a contaminação.

Problema I

Inteligência Artificial em Jogos

Arquivo fonte: `inteligencia.{c | cpp | java}`

Autor: Leandro Luque (Fatec Mogi das Cruzes)

Joãozinho está fazendo um curso de extensão em programação de jogos na Fatec Mogi, que tem como trabalho final o projeto e a implementação de um jogo qualquer. Para matar a saudade dos primeiros jogos tridimensionais que jogou, ele optou por criar uma réplica do jogo Wolfenstein 3D.

Este é um jogo de tiro em primeira pessoa no qual o jogador anda e corre por um cenário, composto principalmente por corredores, e tem que eliminar os inimigos que encontra pelo caminho. Após implementar a parte gráfica e de interação do jogo, Joãozinho deseja criar um algoritmo para que os inimigos que estejam dentro de um raio máximo de distância do jogador, quando este atirar, se dirijam até a posição onde ele se encontra para interceptá-lo.

A Figura 1 ilustra uma fase do jogo, onde um 'X' representa uma parede, o sinal ':' representa o jogador, um '*' representa um inimigo, e um espaço vazio ' ' representa um corredor por onde o jogador ou os inimigos podem passar.

```
7 X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X
6 X     :         X   *                                       X
5 X X X   X X X   X X       X X X X X X X X X X X X X X X X X X
4 X X X   X X X   X X    *                    X X X X    X X X X
3 X X X           X X X X X X X X X X X X X      X X X      X X X
2 X X X X X X X   X X X X X X X X X X X X X      X    *        X X
1 X X X X X X X             *                    X X X X X X X X
0 X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
```

Figura 1: Exemplo de fase do jogo com o jogador e quatro inimigos.

Na implementação feita por Joãozinho, a fase é vista como uma parte do quadrante superior direito de um plano cartesiano, com o canto inferior esquerdo em (0,0). Assim, no exemplo apresentado, o jogador está em (3,6) e os inimigos estão em (10,6), (11,4), (12,1) e (24,2). Antes de definir o caminho que o inimigo seguirá para chegar ao jogador, Joãozinho precisa da sua ajuda para definir se há algum caminho entre o jogador e qualquer inimigo que esteja no máximo a uma distância R dele.

Entrada

A entrada é composta por vários casos de teste. A primeira linha de cada caso de teste define dois números inteiros L ($0 < L < 50$) e C ($0 < C < 50$), separados por um espaço em branco, representando o número de linhas e colunas da fase do jogo respectivamente. As próximas L linhas contêm C colunas com um dos seguintes caracteres cada: 'X' (parede), ' ' (espaço livre), ':' (jogador) ou '*' (inimigo). A última linha do caso de teste informa um raio R inteiro ($R > 0$), representado em unidades do plano cartesiano, dentro do qual um tiro do jogador é ouvido pelos inimigos. Em outras palavras, caso a distância no plano entre o jogador e o inimigo seja menor ou igual a R, este pode ouvir um tiro dado pelo jogador.

Saída

Para cada caso de teste, você deve imprimir um caractere 'S', sem aspas e maiúsculo, caso algum inimigo dentro do raio R especificado (<= R) consiga chegar até o jogador por meio de um corredor. Caso não seja possível, imprima 'N', sem aspas e maiúsculo. É importante notar que um inimigo pode passar por qualquer caminho que não seja uma parede, inclusive por um caminho onde se encontra outro inimigo, mas ela só caminha na horizontal e na vertical, nunca na diagonal. Além disso, inimigos podem estar em corredores sem saída.

Para o exemplo apresentado anteriormente, caso o raio R fosse igual a 7 (R = 7), o inimigo da casa (10,6) ouviria o tiro. Como existe um caminho que o leva até o jogador – descer até (10,4), ir para a direita até (20,4), descer até (20,1), ir para a esquerda até (7,1), subir até (7, 6), ir para a esquerda até (3,6) -, a resposta seria 'S'.

Exemplos

Entrada:	**Saída:**
8 30 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX X : X * X XXX XXX XX XXXXXXXXXXXXXXXXXX XXX XXX XX * XXXX XXX XXX XXXXXXXXXXXX XXX XXX XXXXXXX XXXXXXXXXXXX X * XX XXXXXXX * XXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 7 8 30 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX X X * X XXX XXX XX XXXXXXXXXXXXXXXXXX XXX XXX XX * XXXX XXX XXX XXXXXXXXXXXX XXX XXX XXXXXXX XXXXXXXXXXXX X * XX XXXXXXX * :XXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 5	S N

Solução

Uma possível solução para o problema "Inteligência Artificial" envolve os seguintes passos:

1) Identificar quais casas (corredor ou que contêm inimigos) que podem ser alcançadas pelo jogador. Isso pode ser feito por meio de um algoritmo de busca em largura ou profundidade, partindo da casa do jogador e visitando as casas acima, abaixo, à esquerda e à direita dele que não sejam parede. Pode-se utilizar uma matriz booleana para armazenar as casas alcançáveis por ele.

2) Em seguida, para cada inimigo, deve-se calcular a distância em relação ao jogador no plano cartesiano, o que pode ser feito pelo Teorema de Pitágoras: distancia = sqrt((x2-x1)^2 + (y2-y1)^2) - independentemente deles estarem na mesma linha ou coluna da fase, a fórmula é a mesma.

3) Por fim, para os inimigos que estiverem a uma distância igual ou inferior a R, deve-se verificar se a posição dele é alcançável pelo jogador (passo 1), que é o mesmo que verificar se ele pode alcançar o jogador.

Obs.: Entre as otimizações que podem ser feitas na solução citada está a verificação da distância do jogador para os inimigos já na leitura dos dados e o descarte dos inimigos que estejam a uma distância superior a R.

Problema J

Cegonha

Arquivo fonte: `cegonha.{c | cpp | java}`

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Dr. Obsvaldo é um dedicado obstetra que atende a pessoas carentes na rede pública de Pindorama do Sul. Dentro das dificuldades técnicas que sua área apresenta e o baixo padrão de vida da população por ele atendida, o médico faz o melhor que pode para garantir uma boa saúde para suas pacientes e seus bebês. Ultimamente ele vem percebendo um agravamento da questão do ganho de peso das futuras mamães durante a gravidez, que ele atribui em parte à falta de orientação nutricional daquela população e em parte ao folclore de que gestante deve engordar mesmo e comer à vontade. A questão é que um ganho de peso exagerado compromete o processo de gestação, pode aumentar os riscos na hora do parto e também dar origem a problemas de saúde para a mãe e seu filho. O ganho de peso considerado aceitável varia conforme o estado da mulher antes de ficar grávida: mulheres mais magras podem engordar um pouco mais durante a gravidez do que mulheres com problemas de peso. Para orientar o médico no acompanhamento da gestação existe uma tabela de referência, elaborada pelo *Institute of Medicine*, que associa o ganho de peso tolerado conforme a situação de peso da mulher antes da gravidez. Essa tabela é apresentada na Figura 1.

IMC antes do início da gravidez	< 18.5	>= 18.5 e < 25	>= 25 e < 30	>= 30
Ganho de peso esperado durante a gravidez	12.5 a 18 kg	11 a 16 kg	7 a 11.5 kg	5 a 9 kg

Figura 1: Tabela de referência para ganho de peso (*Institute of Medicine*, 2009)

Para evitar dúvidas sobre essa tabela devemos lembrar que o IMC de uma pessoa é um índice de uso muito comum para estimar o seu grau de obesidade, e é calculado dividindo-se o peso da pessoa (medido em quilogramas) pelo quadrado da sua altura (medida em metros). Sua escola se propôs a elaborar um programa para o Dr. Obsvaldo, em que o médico informa a altura e o peso apresentado pela gestante antes da gravidez e também o seu peso atual. O programa então responde ao médico se o ganho de peso daquela gestante está dentro do tolerado ou não, segundo a tabela de referência. Como sua equipe é composta pelos melhores programadores da escola, a tarefa sobrou para vocês.

Entrada

A entrada possui vários casos de teste. Inicialmente um valor *N* é informado, indicando a quantidade de gestantes a serem avaliadas. Seguem-se *N* linhas, cada uma contendo três números reais positivos H, *O* e *A* separados por um espaço em branco representando, respectivamente, a altura da gestante, o seu peso antes da gravidez e o seu peso atual. Um ponto decimal separa a parte inteira da parte fracionária de cada número.

Saída

Para cada caso de teste imprima a mensagem ‘GANHO INSUFICIENTE’ se o peso atual da gestante for inferior ao esperado, ‘GANHO NORMAL’ se estiver dentro da faixa esperada e ‘GANHO EXAGERADO’ se o ganho de peso da paciente for maior do que o esperado para a sua condição. Utilize nos cálculos dados do tipo real de precisão simples (tipo `float`) e escreva as mensagens de resposta em letras maiúsculas.

Exemplos

Entrada:	**Saída:**
5 1.51 50.5 65.4 1.64 40.3 58.9 1.42 60.3 65.3 1.42 60.3 68.0 1.42 60.3 62.4	GANHO NORMAL GANHO EXAGERADO GANHO INSUFICIENTE GANHO NORMAL GANHO INSUFICIENTE

Solução

Um dos problemas mais fáceis da prova, requeria receber os dados, determinar o IMC inicial da gestante (dividindo-se o peso da pessoa pelo quadrado da sua altura), o ganho de peso verificado e decidir, com base nesses dois valores e na tabela fornecida no enunciado, qual a mensagem a imprimir.