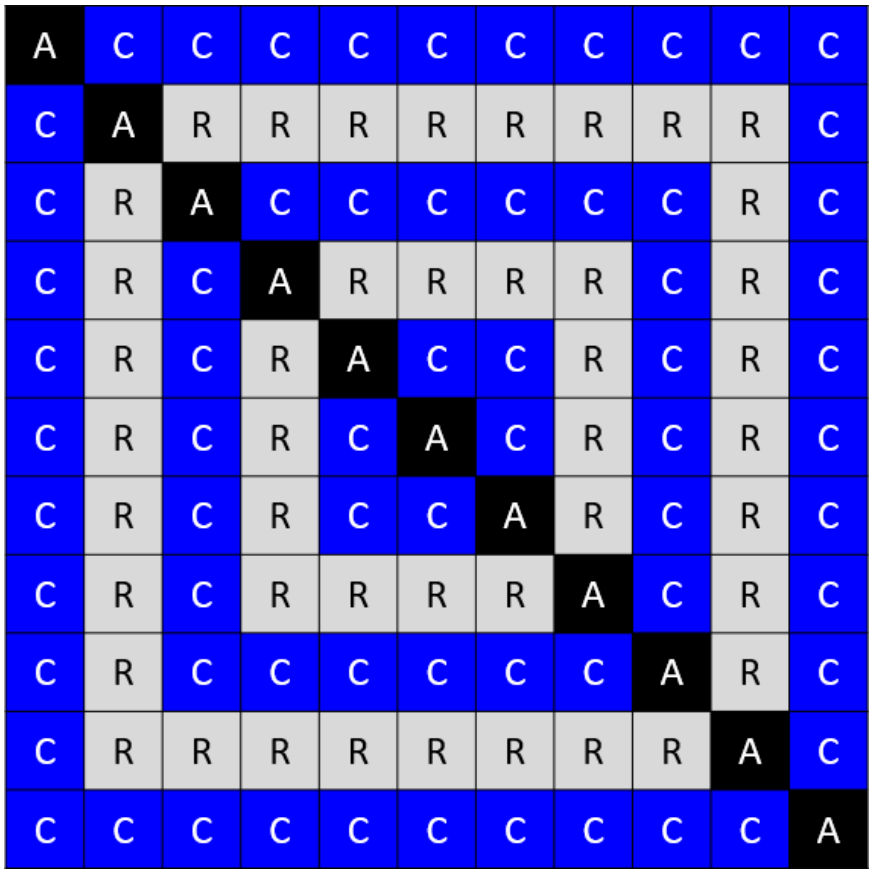
Problema A

SquareCity

Arquivo fonte: square.{ c | cpp | java | py }
Autor: Rodrigo Plotze (FATEC Ribeirão Preto)

Stanley is an architect and is working on the urban design of a new city called *SquareCity*. The city was entirely planned from concentric squares and formed exclusively by houses, streets and avenues. In the project, the houses and streets are arranged concentrically up to the outer limits of the city. A single avenue cuts the main diagonal of the city, and streets separate the blocks of houses.

To help the project visualization, Stanley used graph paper and observed the layout of the houses, identified by the letter *C*, of the streets represented by the letter *R*, and also of the avenue indicated by the letter *A*.



The main difficulty faced by Stanley is to quickly and practically determine the number of houses that can be built in the city. For that, you will have to elaborate a computational solution capable of performing this calculation.

Entrada

The input is composed of a single line containing an odd integer value named x such that $1 \leq x \leq 101$. The value of x represents the number of rows and columns used in the graph paper.

Saída

A single integer value indicating the number of houses that can be built for a city of size x, followed by a line break.

Exemplo de Entrada 1	Exemplo de Saída 1
1	0

Exemplo de Entrada 2	Exemplo de Saída 2
3	6

Exemplo de Entrada 3	Exemplo de Saída 3
9	28

Problema B

Pirâmide Alfabética

Arquivo fonte: piramide.{ c | cpp | java | py }

Autor: Prof. Me. Lucio Nunes de Lira (Fatec Ferraz de Vasconcelos)

Um famoso personagem, de um igualmente conhecido filme, disse a seguinte frase: "*Palavras são, na minha nada humilde opinião, nossa inesgotável fonte de magia [...]*". É claro que, para formar palavras, precisamos que exista um alfabeto com símbolos que permitam construí-las. Um exemplo é o nosso alfabeto latino.

O alfabeto latino contém vinte e seis letras, iniciando com o caractere 'A' e encerrando em 'Z', se desconsiderarmos as acentuações e as diferenças entre letras maiúsculas e minúsculas.

Hermione, uma garota muita estudiosa, percebeu que é possível desenhar usando letras do alfabeto latino. Em uma folha quadriculada de vinte e seis colunas, Hermione escreveu na 1ª linha e 26ª coluna o primeiro caractere do alfabeto. Na 2ª linha escreveu o primeiro e o segundo do alfabeto, ocupando a 25ª e a 26ª colunas, respectivamente. Na 3ª linha escreveu do 1º ao 3º caractere, preenchendo da 24ª à 26ª coluna. Com este procedimento, foi possível preencher a 26ª linha com todos os caracteres do alfabeto, em que 'A' ocupou a 1ª coluna e 'Z' ocupou a 26ª. Assim, formou-se uma "pirâmide alfabética", semelhante a um triângulo retângulo, como pode ser visualizado na Figura 1, supondo que o desenho parasse na 8ª linha.

Figura C.1: Exemplo de pirâmide alfabética com oito linhas.

Hermione está ocupada estudando para uma prova de programação (que ela também considera como um tipo de magia!) e pediu sua ajuda para automatizar os desenhos das "pirâmides alfabéticas".

Entrada

Um número natural N ($1 \leqslant N \leqslant 26$) e uma *string* P ($P \in \{maiusculas, minusculas\}$) (sem acentuação e em minúsculas), indicando se a pirâmide será composta só de letras maiúsculas ou minúsculas, respectivamente.

Saída

Uma pirâmide alfabética com exatas N linhas e com letras maiúsculas ou minúsculas, conforme P , seguindo a mesma estratégia descrita no texto e ilustrada na figura e casos de teste de exemplo. Note que: (a) não há espaços entre os caracteres; (b) qualquer uma das vinte e seis colunas não ocupada por uma letra, será

preenchida com um ponto '.' (sem apóstrofos) e; (c) toda linha é encerrada com uma quebra de linha.

Exemplo de Entrada 1	**Exemplo de Saída 1**
8 maiusculas	.......A ......AB .....ABC ....ABCD ...ABCDE ..ABCDEF .ABCDEFG ABCDEFGH

Exemplo de Entrada 2	**Exemplo de Saída 2**
26 minusculas	.....a ....ab ...abc ..abcd .abcde abcdef .abcdefg ..abcdefgh ...abcdefghi ....abcdefghij .....abcdefghijk ......abcdefghijkl .....abcdefghijklm ....abcdefghijklmn ...abcdefghijklmno ..abcdefghijklmnop .abcdefghijklmnopq abcdefghijklmnopqr .abcdefghijklmnopqrs ..abcdefghijklmnopqrst ...abcdefghijklmnopqrstu ....abcdefghijklmnopqrstuv .....abcdefghijklmnopqrstuvw ......abcdefghijklmnopqrstuvwx .....abcdefghijklmnopqrstuvwxy ....abcdefghijklmnopqrstuvwxyz

Problema C

Crausio

Arquivo fonte: crausio.{ c | cpp | java | py }

Autor: Prof. Dr. Leandro Luque (Fatec Mogi das Cruzes)

Nos últimos anos, o uso de robôs limpadores tem aumentado nas residências brasileiras. Munarinho, um estudante de engenharia, aderiu recentemente a essa onda, mas, para economizar, decidiu criar o seu próprio robô: o Cráusio.

Como a grana para o projeto estava curta, ele criou um robô para o qual o usuário tem que configurar manualmente a rotina de movimentos. Para essa configuração, o local de limpeza é representado por uma matriz de $L \times C$ células. Cada célula representa um lugar onde Cráusio pode estar. A configuração inicial do Cráusio inclui a célula de partida (X, Y) e uma sequência de comandos: cima (C), baixo (B), esquerda (E) e direita (D).

No entanto, logo na primeira limpeza, Munarinho percebeu que subdimensionou a bateria de Cráusio e ele não consegue seguir muitos dos comandos especificados na sua rotina. Se Cráusio tem B de bateria, consegue executar B comandos. Outra coisa que Munarinho percebeu é que Cráusio não é muito resistente e, quando ele dá comandos errados e o Cráusio fica batendo nas paredes da casa, sua pintura se desgasta facilmente.

Ajude Munarinho a determinar qual é a última posição que Cráusio consegue alcançar em um dia, dada sua rotina, posição e bateria inicial. Ainda, determine quantas vezes Cráusio bateu nas paredes da casa, representadas pelos limites da matriz.

Entrada

A primeira linha contém três inteiros L, C e B ($1 \leq L, C \leq 100$; $0 \leq B \leq 10000$), representando o número de linhas e colunas da casa de Munarinho e a carga inicial de bateria do Cráusio, respectivamente.

A segunda linha contém dois inteiros X e Y ($1 \leq X \leq L$; $1 \leq Y \leq C$), representando a posição inicial do Cráusio na casa, sendo $(1, 1)$ o canto superior esquerdo.

A terceira linha contém uma string com R caracteres I ($1 \leq R \leq 10000$; $I \in \{C, B, E, D\}$), representando a rotina diária de Cráusio.

Saída

Imprima três inteiros, representando a última posição que o Cráusio conseguirá alcançar (sendo 1,1 o canto superior esquerdo) e a quantidade de vezes que ele bateu nas paredes da casa.

Exemplo de Entrada 1	Exemplo de Saída 1
5 5 10 3 3 CDCBEDBECB	3 3 0

Exemplo de Entrada 2	Exemplo de Saída 2
5 5 10 3 3 EEEEEEDDDDDD	3 5 4

Problema D

Injeção de dependências

Arquivo fonte: injecao.{ c | cpp | java | py }

Autor: Prof. Dr. Leandro Luque (Fatec Mogi das Cruzes)

A gestão eficiente de dependências é crucial para reduzir o acoplamento em sistemas de software, promovendo a modularidade e a reusabilidade do código. Dentre as técnicas para gerenciar dependências, a Injeção de Dependência e o padrão *Service Locator* são amplamente reconhecidos. Na Injeção de Dependência, uma classe não instancia diretamente os objetos das classes das quais depende; ao invés disso, ela os recebe através do construtor, de atributos ou de métodos de acesso.

Desafios surgem quando existem dependências cíclicas, como no caso de A -> B, B -> C e C -> A. Para instanciar A, é necessário instanciar e injetar B, que por sua vez depende de C, que depende de A, formando um ciclo. Uma solução é empregar a "injeção tardia", onde A pode ser criado sem a imediata injeção de B, que ocorrerá após a criação de todas as dependências.

Munarinho está elaborando um *framework* de injeção de dependências para um novo projeto de software e precisa que este identifique automaticamente dependências cíclicas entre os módulos do projeto.

Entrada

A primeira linha da entrada contém um inteiro n ($1 \leq n \leq 500$), representando o número de dependências no projeto. As próximas n linhas contêm dois caracteres maiúsculos a e b indicando que o módulo a depende do módulo b.

Saída

A saída deve apresentar a string "usar injecao tardia"se for identificada uma dependência cíclica no projeto; caso contrário, deve apresentar "ok".

Exemplo de Entrada 1	Exemplo de Saída 1
3 A B B C C A	usar injecao tardia

Exemplo de Entrada 2	Exemplo de Saída 2
2 A B C D	ok

Problema E

Torres de Hanoi

Arquivo fonte: torres.{ c | cpp | java | py }

Autor: Prof. Antonio Cesar de Barros Munari (Fatec Sorocaba)

O jogo Torres de Hanoi é bastante conhecido do pessoal que estuda computação. Em geral, é um dos exemplos clássicos utilizados nas aulas sobre recursividade em programação, já que um elegante algoritmo recursivo é capaz de resolvê-lo sem grande dificuldade. Para quem não está associando o nome à pessoa, Torres de Hanoi possui 3 postes fixos, geralmente chamados de A, B e C, e uma certa quantidade finita de discos. Cada disco possui um diâmetro diferente dos demais, e apresenta um furo em seu centro, de maneira que é possível encaixá-lo em qualquer um dos postes que se desejar. Na configuração inicial todos os discos encontram-se encaixados em um dos postes, em ordem de tamanho, com o disco menor em cima e o disco maior embaixo, conforme ilustra a figura 1, onde a configuração dispõe de 5 discos.

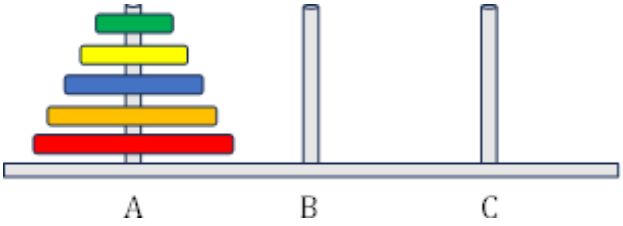


Figura E.1: Estado inicial de uma Torre de Hanoi com 5 discos.

O objetivo do jogo é transferir todos os discos para um poste indicado como destino, sendo permitido mover apenas um disco por vez, utilizando o poste restante como auxiliar para armazenamento e não podendo colocar, em qualquer momento, um disco maior sobre um disco menor no mesmo poste. Existe sempre uma sequência ótima de movimentos para atingir esse objetivo, como mostra a figura 2 onde, para um arranjo de três discos, são necessários sete movimentos. Essa quantidade ótima de movimentos é, como você já deve desconfiar, proporcional à quantidade de discos disponíveis no jogo: para um disco, um movimento; para dois discos, três movimentos; para quatro discos, quinze movimentos e assim por diante. Após cada movimento válido, o jogo assume um novo estado válido, e no processo transitamos de um estado inicial válido para um estado final também válido. É possível descrever cada estado válido do jogo por meio de uma sequência única e específica de zeros e uns, onde o comprimento dessa sequência é igual à quantidade de discos. O primeiro dígito (aquele mais à esquerda) corresponde ao maior disco, o último dígito (aquele mais à direita) representa o menor disco, e os outros dígitos seguem a mesma lógica. Por exemplo, o estado inicial válido para o jogo com 3 discos será 000; se forem 4 discos, será 0000, etc. Considerando o jogo expresso na figura 2, teremos por definição o estado inicial 000; depois do primeiro movimento, teremos 001, que significa que os dois discos maiores estão no mesmo poste e o disco menor está em um outro poste. O segundo movimento levará o jogo para uma configuração 010 (maior disco em um poste, disco intermediário em um poste diferente daquele onde está o maior disco e o disco menor estando em um poste diferente daquele onde está o disco intermediário). O terceiro movimento produzirá o estado 011 (disco maior em um poste, os outros dois discos empilhados em um mesmo outro poste, distinto daquele onde está o maior). O quarto movimento levará a 100 (e aqui perceba um padrão: o disco recém movimentado terá sempre o dígito ‘1’); após o quinto movimento teremos 101; o sexto movimento nos deixará no estado 110 e, finalmente, com o sétimo movimento, encontraremos o estado 111 (todos os discos em um mesmo poste).

Sua tarefa neste problema é bem simples: dada uma sequência de zeros e uns que expressa um estado válido

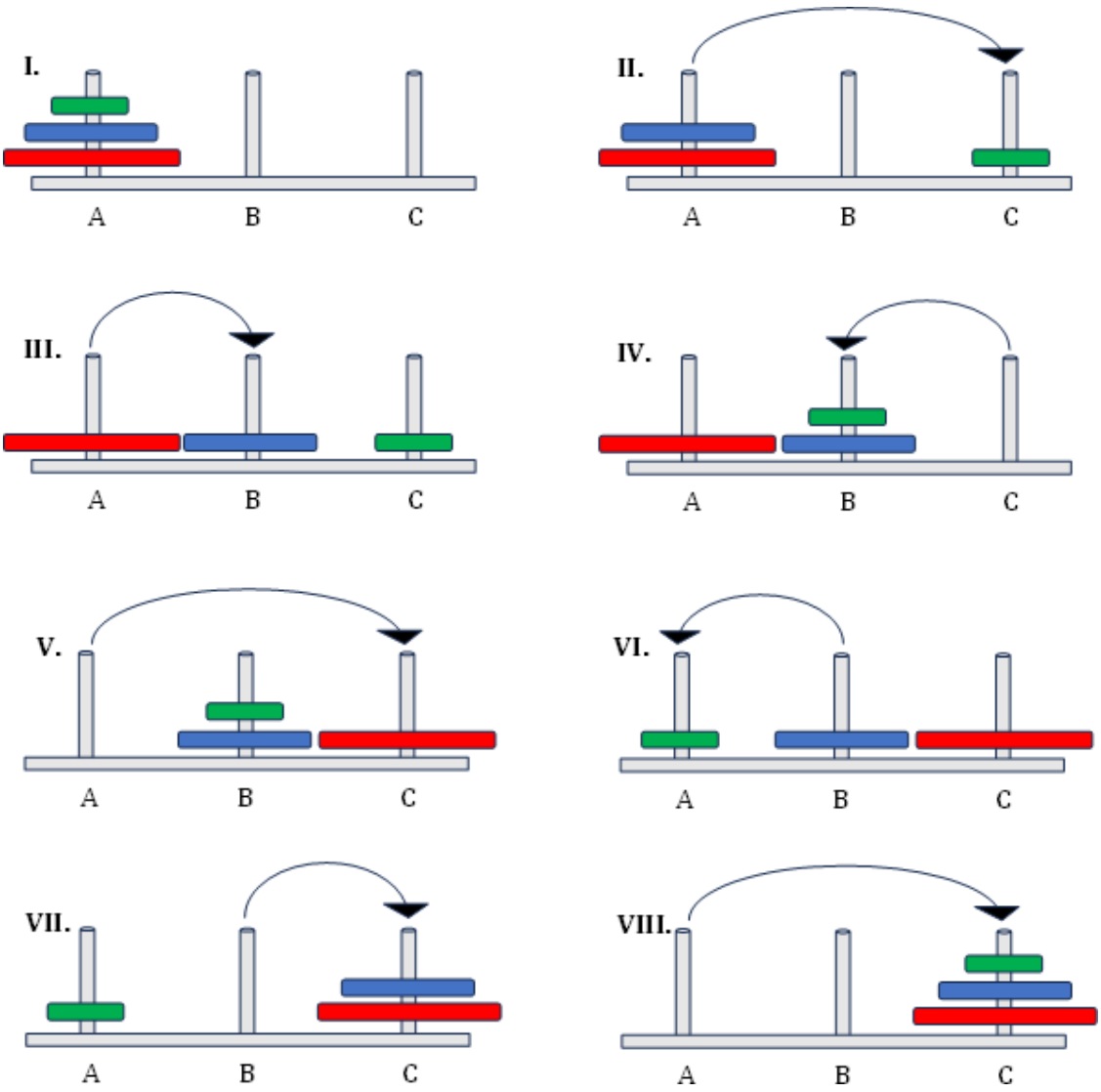


Figura E.2: Solução de uma Torre de Hanoi com 3 discos.

do jogo, indicar quantos movimentos válidos ainda faltam para finalizá-lo, de maneira ótima.

Entrada

Cada arquivo de entrada representa um único caso de teste. Na primeira linha teremos um inteiro N ($0 \leq N \leq 30$) indicando a quantidade de discos. Na segunda linha teremos uma sequência de N dígitos d ($d \in \{0, 1\}$) separados entre si por espaços em branco.

Saída

Imprimir um inteiro X que indica a menor quantidade de movimentos válidos que faltam para o jogo ser finalizado. Não esqueça de deixar uma quebra de linha após o valor impresso.

Exemplo de Entrada 1

```
3
0 1 0
```

Exemplo de Saída 1

```
5
```

Exemplo de Entrada 2

```
5
1 0 0 0 1
```

Exemplo de Saída 2

```
14
```

Exemplo de Entrada 3

```
6
1 0 0 1 1 1
```

Exemplo de Saída 3

```
24
```

Problema F

Fácil demais!

Arquivo fonte: facil.{ c | cpp | java | py }

Autor: Prof. Me. Lucio Nunes de Lira (Fatec Diadema e Fatec Ferraz de Vasconcelos)

Este problema é fácil demais! Dados números naturais positivos, que representam as posições de números primos na sequência infinita de primos, exiba uma mensagem indicando corretamente se a soma desses primos é par ou ímpar. Não entendeu? É tão simples! Por exemplo, se as posições dadas forem 1, 2, 3 e 5, simbolizando o 1º, 2º, 3º e 5º números naturais primos, respectivamente 2, 3, 5 e 11, a soma é 21, ímpar!

Entrada

Um número natural N ($1 \leqslant N \leqslant 10000$) que representa a quantidade de posições de primos que serão dadas. Em cada uma das próximas N linhas, haverá um número natural P ($0 < P < 2^{64}$) que será a posição do número primo na sequência infinita de primos, note que há possibilidade de posições repetidas.

Saída

Apenas a palavra ’par’ (em minúsculo e sem apóstrofos), indicando que a soma dos N números primos é par, ou ’impar’ (em minúsculo, sem acentuação e sem apóstrofos), caso contrário.

Exemplo de Entrada 1	Exemplo de Saída 1
4 1 2 3 5	impar

Exemplo de Entrada 2	Exemplo de Saída 2
1 1	par

Exemplo de Entrada 3	Exemplo de Saída 3
3 1 4 2	par

Exemplo de Entrada 4	Exemplo de Saída 4
3 18446744073709551615 5684 245672783187735	impar

Problema G

Processo Seletivo Simplificado

Arquivo fonte: processo.{ c | cpp | java | py }

Autor: Prof. Antonio Cesar de Barros Munari (Fatec Sorocaba)

Uma das formas de contratação de professores para as unidades do Centro Paula Souza é por meio dos chamados PSSs (Processos Seletivos Simplificados). Nessa modalidade um professor é contratado por tempo determinado e poderá ministrar aulas diversas dentro de sua área de formação nas diversas unidades da nossa autarquia. Quando um PSS é lançado para uma disciplina, os interessados se inscrevem e apresentam a documentação exigida para comprovar sua qualificação para o posto e uma comissão analisa todas as inscrições e emite o resultado final daquele processo. Um dos itens que precisa ser verificado trata da titulação e tempo de experiência profissional do candidato. Esses requisitos são expostos no edital do PSS e estão descritos a seguir, no caso para disciplinas do tipo "profissionalizante".

"ANEXO II – REQUISITOS DA FUNÇÃO E DE TITULAÇÃO

Possuir, na data da inscrição: PARA DISCIPLINAS PROFISSIONALIZANTES

1. Ser graduado e titulado em programa de mestrado ou doutorado reconhecido ou recomendado na forma da lei, sendo a graduação ou a titulação em uma das áreas da disciplina, conforme edital de abertura do certame, bem como possuir experiência profissional relevante de pelo menos 03 (três) anos na área da disciplina, após a obtenção de grau acadêmico (graduação) ou da titulação (mestrado ou doutorado) na área objeto do certame; ou

2. Ser graduado em uma das áreas da disciplina, conforme edital de abertura do certame, e possuir especialização em nível de pós-graduação na mesma área da graduação, bem como experiência profissional relevante de pelo menos 05 (cinco) anos na área da disciplina, após a obtenção de grau acadêmico na área objeto do certame."

O coordenador do seu curso, que anda às voltas com alguns PSSs de disciplinas profissionalizantes, está precisando de um programa que, para um candidato, indique qual o tempo de experiência mínima será preciso comprovar caso este seja considerado apto a concorrer àquela vaga objeto do processo. A ideia é que esse programa ajude a evitar erros na interpretação desse critério de avaliação dos candidatos. O programa receberá um resumo dos dados básicos dos candidatos, elaborado pela comissão avaliadora e informará, para cada candidato, se ele foi desclassificado ou qual o tempo mínimo de atuação profissional que precisa ser comprovado.

Entrada

O arquivo de entradas se inicia com um inteiro N ($1 \leq N \leq 100$) indicando a quantidade de candidatos a serem analisados pelo programa. Seguem-se então N linhas, cada uma consistindo dos inteiros binários G, E, A e D, que indicam, respectivamente, se o candidato informa ter graduação na área, especialização na área, mestrado/doutorado na área e, finalmente, mestrado ou doutorado fora da área da disciplina objeto do oferecimento. Um valor 0 para um desses indicadores significa que o candidato não atende àquele requisito, enquanto que um valor 1 significa que o requisito é atendido. Cada linha termina com um inteiro T ($T \in \{2, 3, 5\}$), que indica o tempo mínimo de experiência profissional informado pelo candidato na sua ficha de inscrição: 2 significa "menos de 3 anos de experiência"; 3 significa "pelo menos 3 anos de experiência" e 5 significa "pelo menos 5 anos de experiência".

Saída

Para cada caso de teste o programa deverá imprimir uma mensagem informando o resultado preliminar da avaliação. Assim, caso o candidato atenda os requisitos, o texto deverá ser "Cand. 999: deferido (comprovar 99 anos)", onde 999 indica o número do candidato (primeiro candidato é 1, segundo candidato é 2, etc) e 99 indica se precisará comprovar 3 ou 5 anos de experiência, conforme os termos apresentados anteriormente. Se o candidato não comprovar os requisitos de titulação mínima, imprimir a mensagem "Cand. 999: INDEFERIDO (acad)"; se a titulação mínima for atendida mas o problema for tempo de experiência profissional insuficiente para a formação acadêmica informada, a mensagem deverá ser, então, "Cand. 999: INDEFERIDO (exp)". Observe o uso de maiúsculas e minúsculas nas mensagens, conforme os exemplos e não se esqueça de imprimir a resposta com uma quebra de linha no final.

Exemplo de Entrada 1	**Exemplo de Saída 1**
4 1 0 1 1 5 0 0 0 0 5 1 1 0 0 5 1 1 1 1 2	Cand. 1: deferido (comprovar 3 anos) Cand. 2: INDEFERIDO (acad) Cand. 3: deferido (comprovar 5 anos) Cand. 4: INDEFERIDO (exp)

Problema H

Power Up Battle (Season #1)

Arquivo fonte: power.{ c | cpp | java | py }

Autor: Fabrício Gustavo Henrique (FATEC Ribeirão Preto)

Power Up Battle é um jogo de cartas que envolve várias magias para aumentar a força do seu exército. Claro, o objetivo é reduzir os pontos de vida do seu oponente a zero. Para medir e treinar o nível dos jogadores surgiu a ideia de desenvolver um "Companion App" que ajuda a identificar qual a maior quantidade de dano possível de ser aplicada em uma jogada. Caso esse jogador atinja essa quantidade máxima de forma constante, certamente isso faz dele um jogador mais experiente.

Essa quantidade ótima de dano é calculada com base na quantidade de cavaleiros aliados no campo de batalha, cada um deles com um poder de ataque inicial de 1 (um), e as cartas/magias que o jogador possui em mãos. Assim, o desafio é identificar qual será a melhor combinação de jogada (ordem em que as cartas/magias são lançadas) de modo a maximizar a quantidade de dano causado pelos cavaleiros?

Nesta temporada, o jogo possui as seguintes cartas:

- 'T' (Treinamento). Cada cavaleiro aliado no campo de batalha recebe +1 de ataque.
- 'R' (Reunir). Cada cavaleiro aliado no campo de batalha recebe +N/2 de ataque, sendo N a quantidade de aliados em campo.
- 'S' (Sacrificar). Sacrifique um cavaleiro. Cada cavaleiro aliado restante recebe +A/2, sendo A o poder de ataque da criatura sacrificada.

A quantidade de cavaleiros no campo de batalha é definida com três dados D20 e a quantidade de cartas compradas do grimório de cartas é definida por um D8.

Dessa forma, deve ser informado para o "Companion App" a quantidade 1 <= N <= 60 de cavaleiros aliados no campo de batalha e a quantidade 1 <= C <= 8 de cartas em mãos. Em seguida, a lista de cada uma das cartas. Como resultado, o "Companion App" deve apresentar o máximo poder de ataque, ou seja, a soma do poder de ataque de cada cavaleiro depois de aplicar as cartas/magias em mãos. Lembrando que não é obrigatório utilizar todas as cartas que estão em mãos, uma vez que nem sempre isso é garantia de maximizar o ataque.

Abaixo segue um exemplo dos dados informados para o "Companion App":

4 3 R T T

Com base nos dados acima temos quatro cavaleiros, todos com o poder inicial de ataque 1 (um), ou seja, termos no campo de batalha "1 1 1 1". Após lançar a primeira carta, 'R', cada cavaleiro recebe +4/2 de poder, resultando no seguinte estado do campo de batalha "3 3 3 3". Ao aplicar a próxima carta 'T', teremos "4 4 4 4" e em seguida com outra carta 'T', "5 5 5 5". Somando o poder de todos os cavaleiros teremos um dano máximo de 20, resultado que deve ser apresentado pelo "Companion App". Vale lembrar que as cartas devem ser utilizadas na melhor ordem possível, não necessariamente na apresentada na entrada.

Exemplo de Entrada 1	**Exemplo de Saída 1**
4 3 R T T	20

Exemplo de Entrada 2	Exemplo de Saída 2
5 3 R S T	24

Exemplo de Entrada 3	Exemplo de Saída 3
5 2 S S	5

Exemplo de Entrada 4	Exemplo de Saída 4
21 5 S T S R S	720

Exemplo de Entrada 5	Exemplo de Saída 5
33 5 T S T T R	960

Problema I

Industria Mágica

Arquivo fonte: industria.{ c | cpp | java | py }

Autor: Lucas Baggio Figueira (FATEC Ribeirão Preto)

Uma determinada indústria dos tempos de outrora deve fazer a separação dos frascos de poção que possuem cores diferentes devido sua finalidade. Um dos gnomos operários dessa indústria deve fazer a separação destes frascos para encaixotá-los, e, como todo o bom gnomo operário ele gosta de diferentes poções para tornar seu trabalho menos entediante, sendo assim ele utiliza uma poção que separa os frascos a cada N sistematicamente até que todos estejam separados. Por exemplo, caso a fila tenha 6 frascos os quais são tirados de 2 em dois até que o frasco 7 seja retirado por último, veja,

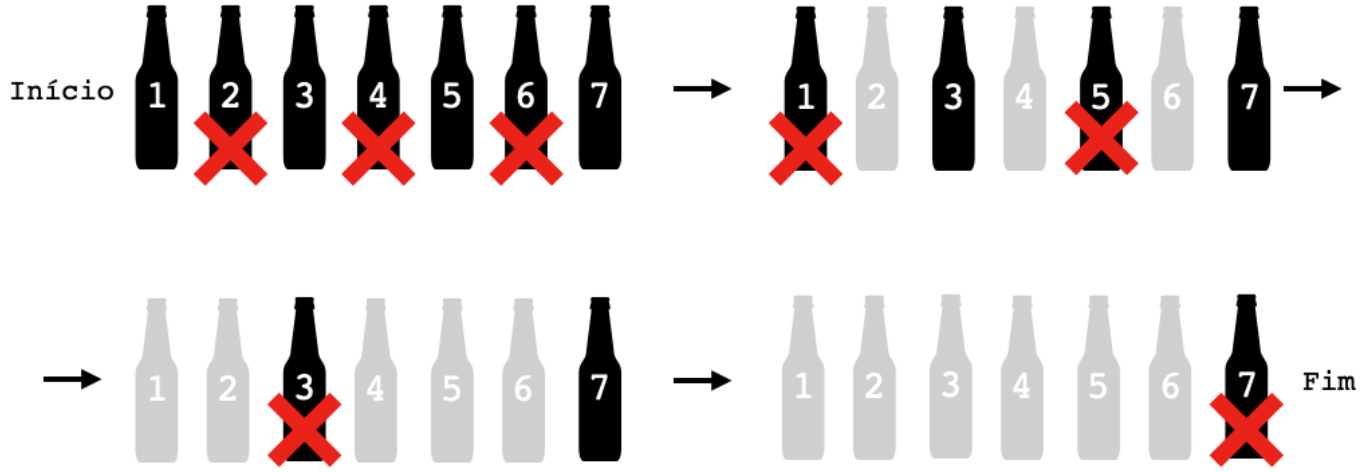


Figura A.1: Sequencia de encaixotamento dos frascos

Entrada

A entrada contém vários casos de teste, cada caso é expresso com 2 inteiros A ($1 < A < 50000$) e B ($1 < B < 5000$), onde A indica a quantidade de garrafas numeradas de forma crescente a partir de 1, e B o tamanho do passo utilizado na remoção sistemática de frascos. A entrada termina com EOF.

Saída

A saída deve, em cada linha, conter o número do último frasco para sua respectiva entrada.

Exemplo de Entrada 1	Exemplo de Saída 1
10 2 6 4 4567 123	5 5 60

Problema J

Nakitomon Cards

Arquivo fonte: nakitomon.{ c | cpp | java | py }
Autor: Lucio Nunes de Lira (Fatec São Paulo)

Danylo Danette e Felipe Silvio são pessoas extremamente ocupadas, de segunda à sexta investem a maior parte do tempo desempenhando seus empregos. Porém, aos sábados, gostam de conversar e jogar cartas juntos, mais precisamente um jogo em que cada carta representa um monstro e seus respectivos pontos em características específicas, o Nakitomon Cards.

Como mencionado, cada carta simboliza um monstro, que possui quatro características pontuadas: (a) força; (b) ataque; (c) defesa e; (d) agilidade. O jogo funciona da seguinte forma, (I) é definida a quantidade de cartas que ambos poderão usar na partida; (II) os jogadores organizam suas próprias cartas em sequência, considerando a maior pontuação de força, seguida pela maior de ataque, em terceiro pela maior de defesa e, por último, pela maior agilidade; (III) a cada rodada os adversários exibem a primeira carta da sequência e verificam quem ganhou com base na maior pontuação (a carta com maior força vence, se empatado verifica-se a próxima característica, o ataque, e assim por diante), caso as cartas empatem em todas as características, nenhum dos jogadores vence, contando como empate. Toda carta usada em uma rodada deve ser descartada em seguida.

Por serem tão ocupados, Danette e Silvio precisam da sua ajuda para criar um programa que automatize as partidas. O programa deverá ler a quantidade de cartas permitidas, os valores das características de cada carta, organizá-las e informar quantas rodadas cada jogador venceu e quantas resultaram em empate.

Entrada

Na primeira linha um número natural n ($1 \leq n \leq 30000$) com a quantidade de cartas de cada jogador; nas n linhas seguintes, n cartas de Danette, uma por linha, com quatro números naturais menores que 1000 representando força, ataque, defesa e agilidade, respectivamente, do monstro; nas próximas n linhas, n cartas de Silvio, no mesmo padrão das de Danette.

Saída

Na primeira linha, a frase *'danette venceu: X'* (sem aspas, em minúsculo e com *X* substituído pelo número de rodadas vencidas por Danette); na segunda linha, a frase *'silvio venceu: Y'* (sem aspas, em minúsculo e com *Y* substituído pelo número de rodadas vencidas por Silvio); na terceira linha a frase *'empates: Z'* (sem aspas, em minúsculo e com *Z* substituído pelo número de rodadas empatadas). Finalize com uma quebra de linha.

Exemplo de Entrada 1	**Exemplo de Saída 1**
2 3 8 0 3 0 8 0 2 0 6 8 3 4 4 6 8	danette venceu: 1 silvio venceu: 1 empates: 0

Exemplo de Entrada 2

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

Exemplo de Saída 2

```
danette venceu: 4
silvio venceu: 3
empates: 0
```

Exemplo de Entrada 3

```
3
10 10 10 10
10 10 10 10
10 10 10 10
10 10 10 10
10 10 10 10
10 10 10 10
```

Exemplo de Saída 3

```
danette venceu: 0
silvio venceu: 0
empates: 3
```

Exemplo de Entrada 4

```
1
10 20 30 40
10 20 30 50
```

Exemplo de Saída 4

```
danette venceu: 0
silvio venceu: 1
empates: 0
```

Problema L

Taylor

Arquivo fonte: taylor.{ c | cpp | java | py }

Autor: Julio Lieira (Fatec Lins)

Taylor, seu colega de grupo de trabalho, diz ter descoberto uma fórmula que calcula a trajetória dos insetos voadores que atacam a nave do jogador em um Jogo estilo Galaga que vocês estão trabalhando. Segundo Taylor, dado o valor (X) do ângulo em graus, deve-se converter esse valor em radiano (R) e aplicar a seguinte fórmula:

$$taylor(R) = \sum_{n=0}^{5} (-1)^n \frac{(R)^{2n}}{(2n)!}$$

Note que o ponto de exclamação (!) na fórmula significa a operação de fatorial. Também note que o cálculo é feito usando R que é o valor de X, dado em graus na entrada, convertido para radianos pela seguinte fórmula:

$$R = X * \pi/180$$

Coube a você implementar o programa que realiza o cálculo. Considere π = 3,1415

Entrada

A entrada consiste de um único valor inteiro X ($0 \leq X \leq 90$).

Saída

Imprima na saída o valor calculado pela fórmula de Taylor. Como se trata de um valor com várias casas após o ponto, este valor deverá ser impresso com três casas após o ponto, mesmo quando forem zero. Porém, aqui deve-se aplicar a seguinte regra de arredondamento, ou aproximação:

- Caso a quarta casa após o ponto seja um número entre 0 e 6, nada se faz na terceira casa. Por exemplo, o valor 0.996195 deverá ser impresso na saída como 0.996. O valor 0.544663 deverá ser impresso na saída como 0.544
- Caso a quarta casa após o ponto seja 7, 8 ou 9, deve-se somar 1 ao valor da terceira casa. Note que isso pode afetar também o número da segunda e primeira casa decimal, bem como da parte inteira. Por exemplo, o valor 0.984808 deverá ser impresso como 0.985. Já o valor 0.529944 deverá ser impresso como 0.530

Finalize a saída com uma quebra de linha.

Exemplo de Entrada 1	Exemplo de Saída 1
5	0.996

Exemplo de Entrada 2	Exemplo de Saída 2
57	0.544

Exemplo de Entrada 3	Exemplo de Saída 3
10	0.985

Exemplo de Entrada 4	Exemplo de Saída 4
58	0.530

Exemplo de Entrada 5	Exemplo de Saída 5
0	1.000