MARATONA DE PROGRAMAÇÃO

InterFatecs

IV MARATONA DE PROGRAMAÇÃO INTERFATECs 2015

www.fateccruzeiro.edu.br/interfatecs

2ª. Fase - 29 de agosto de 2015

Caderno de Problemas

(Sugestões de Solução)

Este caderno contém 10 problemas – A a J, com páginas numeradas de 01 a 20.
Verifique se o caderno está completo.

Informações Gerais

A) Sobre o arquivo de solução e a submissão

1. O arquivo de solução deve ter o mesmo nome que o especificado no enunciado (logo após o título);
2. Confirme se você escolheu a linguagem correta e está com o nome de arquivo correto antes de submeter a sua solução.

B) Sobre a entrada

1. A entrada de seu programa deve ser lida da entrada padrão (não use interface gráfica);
2. A entrada é composta por vários casos de teste, especificados no enunciado;
3. Em alguns problemas, o final da entrada coincide com o final do arquivo. Em outros, o final é especificado por alguma combinação de caracteres ou dígitos.

C) Sobre a saída

1. A saída de seu programa deve ser escrita na saída padrão;
2. Não exiba qualquer mensagem não especificada no enunciado.

Promoção:

Fatec Cruzeiro Prof. Waldomiro May

Centro Paula Souza

Governo do Estado São Paulo – Secretaria de Desenvolvimento Econômico, Ciência, Tecnologia e Inovação

Apoio:

IBM, EMC², UNISAL Centro Universitário Salesiano de São Paulo, CIE Centro de Integração Empresa-Escola, MAXION Structural Components, MAXION Wheels, CENGAGE Learning, MF Soluções em Tecnologia, premium store, Camisetas evidence, TICVALE, ZNC Sistemas, LIAX

Problem A

Armstrong Numbers

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

Author: Antonio Cesar de Barros Munari (Fatec Sorocaba)

An integer number is known as an Armstrong Number when we take each one of its digits to the power of an integer *n* and with the sum of the results we get the original number. For example, 153 is an Armstrong Number with base 3, because 13 + 53 + 33 = 1 + 125 + 27 = 153. The integer 54748 is an Armstrong Number with base 5 because 55 + 45 + 75 + 45 + 85 = 3125 + 1024 + 16807 + 1024 + 32768 = 54748. Your task in this problem is to verify if an integer number is a rare Armstrong Number.

Input

The input has many integers between 2 and 1000000000, each number must be verified if it is an Armstrong Number in some base. Consider in this problem that we want to identify numbers that fits the criteria described above only on bases bigger than 1 and smaller than 10. The program should finish reading the input after a zero is read.

Output

For each test case, print a letter N (uppercase) if the integer read isn´t an Armstrong Number with base between 2 and 9. Otherwise, print the base found for the input. If it was possible to find an Armstrong Number with more than one base, print the smallest.

Examples

Input:	**Output:**
1234 153 370 371 407 201 54748 0	N 3 3 3 3 N 5

Solução

Problema fácil, com maior índice de acertos na prova, requeria que se testasse, para bases de 1 até 10, se era possível produzir o número original por meio da fórmula dada. A solução era então uma simulação, começando com base 2 e testando até que a soma dos dígitos elevados àquela potência produzisse o valor original (e nesse caso o programa deveria encerrar os testes e imprimir a base encontrada) ou que o contador de bases tivesse estourado (e nesse caso imprimir a letra N maiúscula).

Problema B

Calculadora IP

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

Autor: Julio Fernando Lieira (Fatec Lins)

Para se comunicar, um dispositivo de rede (computador, notebook, tablet, smartphone, etc.) precisa de uma identificação única. Em uma rede local ou na Internet os dispositivos são identificados através de um endereço IP. Na versão IPv4, um endereço IP é composto de 32 bits, divididos em quatro octetos (8 bits). Para facilitar a configuração dos dispositivos pelos seres humanos, criou-se a notação ponto decimal, a qual converte cada octeto em seu valor decimal e utiliza um ponto para separá-los. Por exemplo, o endereço IP 192.168.10.5 é representado internamente pelo dispositivo de rede e pelos protocolos de comunicação como sendo 11000000 10101000 00001010 00000101. Veja a correspondência a seguir e note que todos os valores decimais são representados com 8 bits na sua versão binária.

192	168	10	5
11000000	10101000	00001010	00000101

Estes 32 bits de um endereço IP são divididos em duas partes:

- Netid: o qual identifica a rede a qual este endereço pertence. Todos os dispositivos na mesma rede devem ter o mesmo netid;
- Hostid: o qual identifica um determinado dispositivo (host) da rede.

A Figura 1 mostra a distribuição de endereços IP para os dispositivos de uma rede. Note que todos os dispositivos possuem uma parte que se repete (192.168.10) a qual corresponde ao Netid (identificador da rede). Neste esquema de distribuição, o último número (octeto) corresponde ao Hostid (identificador do host), o qual deve ser diferente para identificar cada dispositivo da rede.

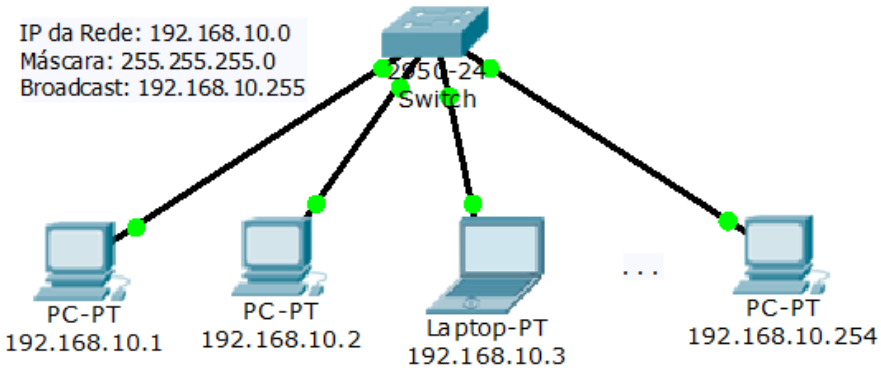


Figura 1- Distribuição de endereços IP para os dispositivos de uma rede.

Quem define qual parte é o Netid e qual parte é o Hostid é a máscara, e isso é feito pela notação binária. Usando a máscara do exemplo da Figura 1 (255.255.255.0), os bits de valor 1 (um) na máscara correspondem ao Netid e os bits com valor 0 (zero) ao Hostid. Assim, neste

exemplo, os 24 primeiros bits formam o Netid e os 8 últimos formam o Hostid, veja abaixo:

192	168	10	5
11000000	10101000	00001010	00000101
11111111	11111111	11111111	00000000
255	255	255	0
Netid			Hostid

Portanto, neste exemplo, com esta máscara é possível ter 254 máquinas na rede, que é a quantidade de combinações possíveis com 8 bits (Hostid) menos 2, pois a primeira combinação 00000000 (tudo bit zero no hostid) e a última 11111111 (tudo bit um no hostid) NÃO podem ser atribuídas às máquinas da rede, pois correspondem, respectivamente, ao IP DA REDE (192.168.10.0) e IP DE BROADCAST (192.168.10.255). Assim, a quantidade de bits no Hostid vai determinar a quantidade de máquinas que é possível ter na rede, bem como quais IPs pertencem a referida rede. Por exemplo, se a máscara fosse 255.255.255.192, os endereços 192.168.10.5 e 192.168.10.80 NÃO pertencem a mesma rede IP, pois:

192.168.10.5:	11000000 10101000 00001010 00000101	
255.255.255.192:	11111111 11111111 11111111 11000000	
192.168.10.80:	11000000 10101000 00001010 01010000	
	Netid	Hostid

Note que com esta máscara o Netid agora compreende os 26 bits da esquerda para a direita. Perceba que existe diferença no último bit da parte do Netid dos dois IPs comparados. Portanto eles NÃO pertencem à mesma rede IP, pois para pertencerem à mesma rede IP precisam ter os mesmos bits na parte do Netid.

Seu trabalho neste problema é dizer se dois IPs pertencem ou não à mesma rede IP.

Entrada

Cada linha da entrada é composta por dois endereços IP e uma máscara, todos na notação ponto decimal e separados por um espaço.

Saída

Para cada linha da entrada imprima na saída a letra S maiúsculo caso os dois IPs pertencem à mesma rede IP, ou a letra N maiúsculo caso não pertencem.

Exemplos

Entrada:	**Saída:**
192.168.10.5 192.168.10.80 255.255.255.0	S
192.168.10.5 192.168.10.80 255.255.255.192	N
192.168.10.5 192.168.10.40 255.255.255.192	S
10.10.10.1 10.20.30.40 255.0.0.0	S

Solução:

Uma possível solução para o problema é usar comparação de strings.
Primeiramente, deve-se converter os IPs (IPA, IPB) e a máscara (MASC) em binário, formando uma string de zeros e uns (strIPA, strIPB, strMASC). Considerando o primeiro exemplo teríamos:

strIPA: 11000000101010000000101000000101
strIPB: 11000000101010000000101001010000
strMASC: 11111111111111111111111111000000

É importante tomar cuidado na conversão de um número que não necessite de 8 bits. Por exemplo, o número 5, convertido em binário pelo método de divisões por 2, daria 101. Entretanto, devemos representá-lo em 8 bits. Então é preciso completar com zeros a esquerda quando os números em binário tiverem menos que 8 bits (5 = 00000101).

Após montar as strings, basta verificar quantos bits 1 tem na string da máscara (cont1MASC), e depois erificar os cont1MASC primeiros caracteres das strings strIPA e strIPB são iguais. Em Linguagem C isso pode ser feito pela função strncmp():

```
If (strncmp(strIPA,strIPB,cont1MASC)==0)
     printf(“S\n”);
else
     printf(“N\n”);
```

Problema C

Number Crush Saga

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

Autor: Anderson Viçoso de Araújo (UFMS)

A *Capivara Apps*, uma das novas startups que estão surgindo no Centro-Oeste brasileiro, está desenvolvendo um jogo similar a um dos jogos mais viciantes lançados recentemente. O novo jogo se chama *Number Crush Saga*. É um jogo relativamente simples: para cada fase do jogo o jogador deve tentar agrupar peças do mesmo tipo para fazer o maior número de pontos possíveis.

A tabela a seguir indica o número de pontos para cada uma das combinações de peças possíveis.

Número de peças combinadas (horizontal ou vertical)	3	4	>=5
Pontuação	60	120	180

Tabela 1. Pontuação de acordo com o número de peças combinadas.

A combinação de peças pode ser realizada tanto na horizontal como na vertical. O número de peças combinadas deve estar de acordo com a tabela 1 apresentada.

Quando um conjunto de peças do mesmo tipo é agrupado, as peças combinadas são retiradas do tabuleiro. Com isso, as peças que estão acima das peças removidas são deslocadas para posições inferiores no tabuleiro podendo gerar novas combinações e assim consecutivamente. As posições da parte superior do tabuleiro, nas colunas onde peças foram deslocadas para baixo, ficam vazias. No jogo não existe a entrada de novas peças no tabuleiro. As pontuações de todas as combinações geradas a partir de uma movimentação devem ser somadas para alcançar a pontuação final da jogada.

No total existem 5 tipos diferentes de peças correspondendo aos números de 1, 2, 3, 4 e 5. Desta forma, ao combinar pelo menos 3 peças com o mesmo número de forma sequencial (tanto na horizontal quanto na vertical) o jogador recebe 60 pontos, caso a remoção destas 3 peças não gere novas combinações. Movimentações que geram mais de uma combinação ao mesmo tempo, tanto na vertical quanto na horizontal, são somadas separadamente.

A movimentação das peças se dá através da troca entre duas peças vizinhas, comparando somente a peça acima, abaixo, à esquerda e à direita (não é possível trocar uma peça com a sua diagonal). Neste jogo é possível movimentar as peças sem que seja obrigatória uma combinação mínima entre as peças vizinhas. Se isso acontecer, a pontuação para a jogada é 0 (zero).

A Tabela 2 apresenta uma matriz com 5 linhas e 5 colunas que representa um tabuleiro completo de jogo, com peças numeradas distribuídas por toda a matriz.

Ao mover a peça 4x3 (linha 4 e coluna 3) para a posição 4x4 uma única combinação vai ser gerada com 3 números 3 na coluna 4. Com esta jogada, o jogador obtém 60 pontos no total. As peças nas posições 0x4 e 1x5 vão ser movimentadas para baixo e ficaram nas posições 4x4 e 3x4, respectivamente. Depois da movimentação as posições 0x4 e 1x5 ficaram vazias, como apresentado na Tabela 3.

	0	1	2	3	4
0	2	3	3	1	4
1	1	1	2	3	4
2	1	1	2	2	3
3	3	2	1	2	3
4	4	4	3	**3**	**1**

Tabela 2. Exemplo de tabuleiro de jogo com as peças numeradas **antes** da movimentação.

	0	1	2	3	4
0	2	3	3	1	
1	1	1	2	3	
2	1	1	2	2	
3	3	2	1	2	4
4	4	4	3	**1**	**4**

Tabela 3. Exemplo a partir da Tabela 2 **depois** da movimentação da peça 4x3 para a posição 4x4.

No exemplo da Tabela 4, caso o jogador mova a peça na posição 3x1 para a posição 3x2, a peça de número 1 é trocada pela peça de número 2. Com essa jogada, o jogador obtém três combinações: a combinação de 3 números 1 na coluna 1; 3 números 2 na coluna 2; e a combinação de 3 peças com o número 3 na linha 2 depois da remoção das seis peças anteriormente combinadas (3 números 1s e 3 números 2s). Após a execução da movimentação o tabuleiro resultante é apresentado na Tabela 5, Com essa jogada o jogador obtém uma pontuação de 180 (60+60+60) pontos pois conseguiu 3 combinações de 3 peças em um mesmo movimento de peças.

	0	1	2	3	4
0	2	3	3	1	4
1	1	1	2	3	4
2	1	1	2	2	3
3	3	**2**	**1**	2	3
4	4	4	3	3	1

Tabela 4. Exemplo de tabuleiro de jogo com as peças numeradas **antes** da movimentação.

	0	1	2	3	4
0				1	4
1				3	4
2				2	3
3	2			2	3
4	4	4	3	3	1

Tabela 5. Exemplo a partir da Tabela 4 **depois** da movimentação da peça 3x1 para a posição 3x2.

A *Capivara Apps* está precisando de um programador para implementar o sistema de pontuação para o seu novo jogo. Você que curte um desafio, decidiu entrar nessa. Seu trabalho é descobrir a quantidade de pontos que o jogador vai ganhar ao fazer uma movimentação de uma peça no tabuleiro.

Entrada

Serão vários casos de teste a serem testados. Inicialmente dois valores inteiros N e M são informados ($3 \leq N$ e $M \leq 100$), indicando o tamanho da matriz a ser lida que corresponde a um tabuleiro de jogo. Seguem-se *N* linhas com *M* colunas, com inteiros de 1 a 5 indicando as peças numeradas para cada posição do tabuleiro. Duas peças consecutivas na matriz são separadas por um espaço em branco e não podem existir posições no tabuleiro sem peças (número). Em seguida, são informados quatro valores inteiros I1, J1, I2 e J2 separados por um espaço ($0 \leq I1$, $I2 < N$ e $0 \leq J1$, $J2 < M$) indicando as posições das duas peças consecutivas a serem trocadas. As entradas terminam com o fim do arquivo. **IMPORTANTE**: Os tabuleiros lidos inicialmente **não possuem combinações pré-existentes** de 3 ou mais peças do mesmo tipo. As combinações só acontecem depois da movimentação das peças.

Saída

Para cada matriz lida, escrever uma linha com o número de pontos alcançados pelo jogador após executar a jogada de entrada de acordo com a tabela de pontuação apresentada, somando com a pontuação das possíveis novas combinações, caso existam, após a remoção das peças combinadas.

Exemplos

Entrada:	**Saída:**
5 5 3 2 3 1 2 3 3 1 4 1 1 2 3 4 1 1 2 2 3 3 2 1 2 3 4 4 3 3 1 3 4 2 0 2 1 1 2 3 4 1 2 3 3 2 1 2 5 3 4 2 1 1 1 1 2 3 4 1 2 3 3 2 1 2 5 5 5 4 4 4 3 2 3 3 1 4 1 1 2 3 4 1 1 2 2 3 3 2 1 2 3 4 4 3 3 1 4 4 0 1 1 1 1 3 3 4 2 2 3 3 1 3 2 2 2 3 1 1	180 120 60 60 120

Solução

Este é um problema considerado de dificuldade média e basicamente de manipulação de matrizes.

Em uma das diferentes maneiras para solucionar este problema podemos seguir os passos:

1. Ler as entradas e armazenar em uma matriz de inteiros ondes os inteiros de 1 a 5 indicam os tipos de peças;
2. Realizar a troca das peças nas posições de entrada;
3. Criar um laço para verificar se existem combinações
    a. Dentro do laço, verificar horizontalmente e verticalmente se existe combinações e quantidade de peças combinadas, para poder gerar a pontuação correspondente.
    b. Caso encontre combinações, marcá-las (por exemplo com zeros) para identificar quais colunas devem “cair”.

    c. Percorrer a matriz procurando por zeros, e verticalmente trocar todas as posições, que não contenham um zero, acima de cada zero encontrada.
    d. Reiniciar o laço;
4. Imprimir a pontuação total armazenada.

Problema D

Pacman

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

Autor: Leandro Luque (Fatec Mogi das Cruzes)

Pacman é um famoso jogo, criado pela Namco para fliperamas, que teve versões para diversos consoles, como Game Boy e Nintendo DS. O jogo consiste em um labirinto onde se encontra o jogador, representado por uma pizza sem uma fatia (essa foi a inspiração real para a personagem), fantasmas, que não podem ser tocados, e itens que devem ser coletados (círculo escuro), conforme Figura 1. Nesta figura, existem dois fantasmas e um item a ser coletado.

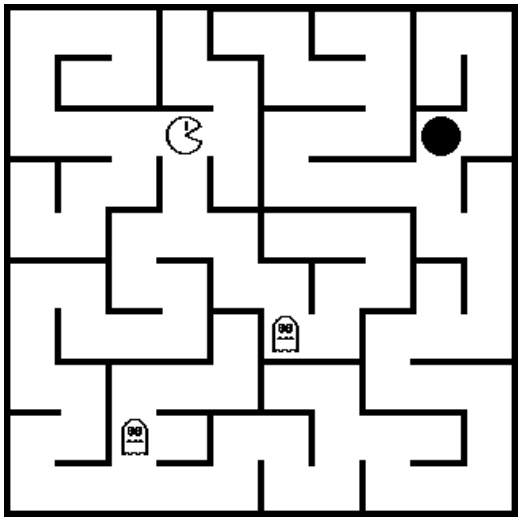


Figura 1. Exemplo de labirinto com o jogador, dois fantasmas e um item a ser coletado.

Você foi convidado para implementar uma nova versão do jogo onde os labirintos são gerados em tempo de execução (no jogo original, o labirinto era sempre o mesmo) e os fantasmas ficam parados (no jogo original, eles se moviam). Em resumo, o jogo consiste em mover o jogador pelo labirinto, sem tocar nos fantasmas, até coletar o item. Para tanto, você precisa saber se, dado um certo labirinto gerado pelo seu código, existe ao menos um caminho que leva o jogador até o item sem passar por um fantasma.

Entrada

A entrada contém vários casos de teste. A primeira linha de cada caso de teste contém um inteiro N ($7 \leq N \leq 20$), indicando o número N de linhas e colunas do labirinto (incluindo as paredes externas). As próximas N linhas contêm N caracteres C (C='*', C='I', C='T', C='#', C=' ') cada representando o labirinto e o seu conteúdo. Um caractere C='*' indica uma parede do labirinto. Um caractere C='I' indica o local onde se encontra o jogador. Um caractere C='T' indica o local onde está o item a ser coletado. Um caractere C='#' indica o local onde está um fantasma. Um espaço em branco (C=' '), por sua vez, representa um caminho livre no labirinto. Em um mesmo labirinto, existe apenas um jogador e um item a ser coletado, mas podem existir vários fantasmas. A entrada termina com o fim do arquivo.

Saída

Para cada caso de teste, imprima um caractere ‘S’ (maiúsculo e sem aspas), caso exista ao menos um caminho entre o jogador e o item a ser coletado sem passar por fantasmas. Caso contrário, imprima um caractere ‘N’ (maiúsculo e sem aspas).

Exemplos

Entrada:	**Saída:**
7 ******* * T*I* * *** * * # * * *** * * * *******	S

Solução:

Trata-se de um problema onde pode ser utilizada uma técnica de backtracking com recursão para percorrer a matriz a partir da posição do Jogador. Os seguintes passos podem orientar a implementação:

1- Fazer a leitura do Labirinto em uma Matriz;
   Durante a entrada já identificar e guardar a posição (linha,coluna) do Jogador;
2- Por recursão, localizar o Item a partir da célula onde se encontra o Jogador:

```
LocalizarItem()
    Se a célula contém o Jogador, retornar ‘S’
     Senão:
        Marcar item como visitado;
        Localizar Item na célula da direita;
        Localizar Item na célula de baixo;
        Localizar Item na célula da esquerda;
        Localizar Item na célula de cima;
   Se passou por todos e não encontrou, retornar ‘N’
```

Problema E

Escalas

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

Autor: Anderson Viçoso de Araújo (UFMS)

Uma construtora precisa de um programinha que a ajude a definir rapidamente o tamanho das suas construções baseando-se em maquetes. Para fazer isso, a empresa tem em mãos dados de maquetes correspondentes definidas através de escalas reduzidas (tanto na horizontal quanto na vertical).

A escala tem o formato: "**x:y**", onde **x**, o número antes dos dois pontos, indica o tamanho em centímetros apresentado na maquete e; **y** o valor depois dos dois pontos, corresponde ao tamanho em centímetros na construção real.

Seu trabalho é fazer um programinha que auxilie na descoberta do tamanho de cada construção real para as escalas de entrada. O programa deve retornar a **área total** a ser construída **em metros quadrados**. A área total é calculada através do somatório das áreas de todos os andares da construção. Para calcular o número de andares, deve-se verificar se a construção possui altura maior ou igual a 6 metros. Cada andar corresponde a 3 metros (sem contar os primeiros 3 metros que correspondem ao térreo). Caso o número de andares corresponda a uma fração, arredondar o valor para o primeiro valor inteiro inferior ao número.

Para a figura abaixo a escala horizontal é 1:1200 e a vertical é 1:250. Desta forma, o tamanho da construção do exemplo apresentado na figura é de 7200 metros quadrados (horizontal), 5 metros de altura (vertical) e nenhum andar (0) além do térreo. A saída do programinha para este exemplo deve ser 7200.

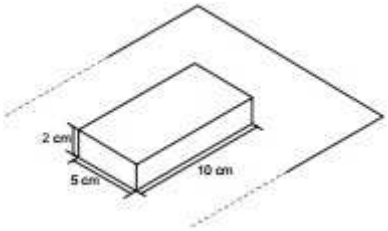


Figura 1. Exemplo de maquete com 2 cm de altura, 5 cm de largura e 10 cm de comprimento.

Entrada

Serão vários conjuntos de teste a serem testados. Para cada conjunto de testes, duas *strings* H e V correspondendo às escalas (horizontal e vertical, respectivamente) e um inteiro N ($1 \leq N \leq 100$) são informados em uma mesma linha separados por um espaço em branco. Cada *string* (H e V) contém dois valores inteiros I e J (I=1, $1 \leq Jh$ e $Jv \leq 1000$) indicando o valor para a escala correspondente em centímetros, separados por um símbolo "**:**". Seguem-se N linhas com 3 inteiros L, C e A ($1 \leq L, C \leq 1000$ e $1 \leq A \leq 100$), separados por um espaço em branco, indicando o tamanho em centímetros para a largura, comprimento e altura, respectivamente, de cada construção a ser avaliada de acordo com as escalas da maquete correspondente. As entradas terminam com o fim do arquivo.

Saída

Para cada caso de teste lido, escrever o texto "Escalas X" onde X indica o número do conjunto de teste associado a um par de escalas (horizontal e vertical). Para cada caso de teste,

escrever uma linha com o tamanho da área total da construção em metros quadrados. Caso o número seja de ponto flutuante, arredondá-lo para o maior inteiro posterior a ele. A área total da construção é considerada como sendo o somatório de todas as áreas de todos os andares (incluindo o térreo), supondo sempre que todos os andares possuem a mesma área da base da construção. Cada andar corresponde a 3 metros de altura para construções com no mínimo 6 metros de altura. Uma linha em branco deve ser deixada após cada conjunto de testes.

Exemplos

Entrada:	**Saída:**
1:1200 1:250 3 5 10 2 20 50 20 100 100 5 1:1000 1:1000 2 200 150 90 20 2 7	Escalas 1: 7200 2304000 5760000 Escalas 2: 900000000 92000

Solução

Este é um problema considerado de dificuldade baixa e usa basicamente aritmética.
Uma das diferentes maneiras para solucionar este problema pode ser:

1. Para cada dupla de escalas lidas, incrementar um contador de escalas e imprimir o rótulo das escalas;
    a. Para cada trio de inteiros com os tamanhos, calcular a área usando a escala e transformando para metros. Calcular altura da mesma forma;
        i. Para calcular o número de andares, caso a altura seja maior ou igual a 6 metros, temos que pegar o piso da divisão da altura por 3 (tamanho do andar). Caso contrário, só temos um andar.
        ii. Imprimir o teto inteiro da multiplicação da área pelo número de andares.

Problema F

O jogo das letras

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

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Zequinha ganhou de seus pais um jogo pedagógico que trabalha com letras. Nele o jogador sorteia uma palavra e precisa formar todos os conjuntos possíveis com as letras que formam aquela palavra sorteada. A pontuação depende do tamanho da palavra, da quantidade de conjuntos distintos que o jogador consegue formar com as letras daquela palavra e do tempo que levará para completar a tarefa. Zequinha não tem sido muito bom com as palavras, suas letras e as possibilidades de conjuntos (talvez seja por isso que ganhou o jogo de seus pais) e também não está se entendendo direito com a interface do jogo, pois este informa apenas quantos conjuntos estão faltando na resposta de Zequinha, mas não mostra quais ficaram faltando.

Você, como o tio computeiro preferido de Zequinha, foi convocado pelo sobrinho para escrever um programa que, dada uma palavra, informa todos os conjuntos que podem ser formados com suas letras. Lembre-se que um conjunto não possui elementos repetidos (senão seria um multiconjunto) então, apesar de as palavras informadas por Zequinha poderem ter repetição de letras, não poderá haver letras repetidas dentro dos conjuntos que seu programa vai produzir.

Entrada

Inicialmente um valor N é informado, $1 \leq N \leq 100$, indicando a quantidade de palavras a serem processadas. Seguem-se N linhas, cada uma com uma palavra composta por até 15 caracteres alfabéticos. O programa deverá considerar que as versões maiúscula e minúscula de uma letra correspondem a uma mesma letra.

Saída

Para cada caso de teste, imprima uma linha contendo apenas os conjuntos não vazios (ou seja, que possuem pelo menos uma letra). Cada conjunto deve ser expresso na forma de uma sequência de letras maiúsculas sem espaço entre si, em que as letras aparecem em ordem alfabética. Exibir os conjuntos em ordem também alfabética, como mostram os exemplos, usando um espaço em branco para separar um conjunto do conjunto seguinte.

Exemplos

Entrada:	**Saída:**
2 Fatec BaNAna	A AC ACE ACEF ACEFT ACET ACF ACFT ACT AE AEF AEFT AET AF AFT AT C CE CEF CEFT CET CF CFT CT E EF EFT ET F FT T A AB ABN AN B BN N

Solução

A solução esperada deveria inicialmente converter todas as letras para maiúsculas e eliminar as repetições de letras da palavra lida. Então, para simplificar o processamento posterior, é interessante colocar as letras utilizadas previamente em ordem ascendente, o que pode ser feito por algum método de ordenação ou por manter um vetor de letras constante e ordenado, apenas com um flag indicando para cada letra se ela é utilizada ou não. A geração dos subconjuntos pode ser realizada por meio de loopings ou chamadas recursivas, que para cada possível letra inicial, produz todas as sequências possíveis. Existem algoritmos específicos para a enumeração de todos os subconjuntos possíveis.

Problema G

Matemática é o melhor remédio

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

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Toninho está doente, abatido, macambúzio. Também está sorumbático e meditabundo com seus problemas de saúde. Ele, que é professor de Matemática, saiu da consulta com o médico bastante desanimado, cheio de remédios para tomar. O doutor que o atendeu até tentou animá-lo, sacaneando um pouco na receita com um pequeno desafio matemático ao taciturno paciente (vai que o Toninho se anima um pouco ...). A receita emitida pelo médico indica, para cada remédio, quantas vezes este deve ser ingerido por dia e também a quantidade (em mg) referente a cada dose. O médico orientou a Toninho para que tome a primeira dose de cada remédio todas juntas, e depois siga tomando a medicação segundo o horário de cada uma até que ocorra uma nova situação de coincidência total, em que ele precise novamente tomar todos os remédios juntos. Quando isso acontecer ele deve tomar a dose de cada remédio e então suspender a medicação. Repare que o médico não especificou diretamente na receita por quanto tempo ou quantas doses Toninho precisa tomar de cada remédio, ele precisará fazer as contas para saber. Esse é o singelo desafio que o dedicado doutor propôs ao moribundo como uma até certo ponto sádica forma de animá-lo. Sua tarefa neste problema será determinar por quantos dias Toninho tomará a medicação e quanto de remédio ele terá que tomar e comprar.

Entrada

A entrada possui vários casos de teste. Inicialmente um valor Q é informado, indicando a quantidade de casos a serem processados. A descrição de cada caso é iniciada por um inteiro N, que indica a quantidade de remédios distintos que Toninho precisa tomar. Seguem-se N linhas, cada uma contendo uma string de até 20 caracteres alfanuméricos que corresponde ao nome do remédio, um inteiro D que representa a quantidade (em mg) correspondente a uma dose, um inteiro I que representa de quantas em quantas horas Toninho deve tomar uma dose daquele remédio e um inteiro C que indica quantos mg existem em cada caixa do remédio. Assuma que $1 \leq Q \leq 100$; $2 \leq N \leq 200$; $1 \leq D \leq 500$; $1 \leq I \leq 10$ e $1 \leq C \leq 1000$.

Saída

Para cada caso de teste, imprima o número sequencial do caso, conforme mostra o exemplo. Em uma nova linha imprima um inteiro indicando por quantos dias vai se estender a medicação, assumindo que Toninho vai tomar a primeira dose exatamente no primeiro instante após a meia noite do dia 1 do tratamento. Em seguida, imprima N linhas, uma para cada remédio, contendo o nome do medicamento, a quantidade total em mg que Toninho precisa ingerir daquele remédio e a quantidade de caixas que precisarão ser compradas para atender à necessidade do doente. Imprima os dados dos remédios em ordem alfabética dos seus nomes. Deixe uma linha em branco após cada caso de teste.

Exemplos

Entrada:	**Saída:**
2 3 Amarazil 5 2 20 Mapetilec 1 3 8	Caso 1: Total de dias = 1 Alopram 12 1 Amarazil 65 4

Alopram 3 8 12 5 Pteridax 1 2 12 Xanxalan 1 3 5 Apurinol 3 2 20 Hemorroidol 5 3 500 Ateridan2b 10 7 12	Mapetilec 9 2 Caso 2: Total de dias = 2 Apurinol 66 4 Ateridan2b 70 6 Hemorroidol 75 1 Pteridax 22 2 Xanxalan 15 3

Solução

Um problema de lógica relativamente simples que nenhuma equipe conseguiu resolver durante a prova. O cálculo da quantidade de dias de medicação requeria que se determinasse o Mínimo Múltiplo Comum (MMC) do conjunto de valores referentes ao valor I da entrada de dados, correspondente à periodicidade em que cada medicamento deveria ser tomado. A quantidade de dias seria o quociente do MMC dividido por 24, e caso a divisão deixasse resto, acrescentar um ao quociente obtido. Algumas tentativas de resolver o problema por simulação ficaram muito lentas e receberam mensagem Time Limit Exceeded. Os dados dos medicamentos deveriam ser ordenados ascendentemente, e o cálculo das quantidades em mg e em caixas é simples, requerendo apenas que se considere que a quantidade de caixas é um inteiro.

Problema H

A conjectura de Goldbach

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

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

A conjectura de Goldbach é um famoso postulado da Teoria dos Números que ainda não foi provado como válido para todos os casos possíveis. Trata-se da afirmação de que todo número par maior que 2 pode ser expresso como sendo a soma de dois números primos, como mostram os exemplos a seguir.

4 = 2 + 2

6 = 3 + 3

8 = 3 + 5

10 = 3 + 7 ou 10 = 5 + 5

Neste problema você deverá apresentar, para um número lido da entrada, um par de primos que somados produzem o número lido originalmente. Apesar de ainda não ter sido provada como válida, a conjectura de Goldbach foi verificada como correta para uma enorme faixa de valores.

Entrada

A entrada é composta por diversos números inteiros, um em cada linha. Todos os números serão maiores do que 2 e menores do que 1000000, com exceção do último valor, que será zero e sinaliza o fim das entradas. Todos os números pares que seu programa receberá possuem uma decomposição consistente com a conjectura descrita anteriormente.

Saída

Para cada caso de teste imprima a mensagem “erro” se o número lido for ímpar. Se for par, imprima, separados por um espaço em branco, os dois números ímpares que, somados, produzem o valor lido originalmente. Como é possível que um número possa ser produzido pela soma de diversas duplas de primos, seu programa deverá imprimir apenas aquela dentre as duplas possíveis que tiver o menor primo. Ao exibir a dupla na saída, imprimir os números em ordem não decrescente de seus valores. Lembre-se que o número 1 não é primo.

Exemplos

Entrada:	**Saída:**
4 6 8 10 100 5 12 13 0	2 2 3 3 3 5 3 7 3 97 erro 5 7 erro

Solução

Segundo problema com maior número de acertos durante a prova, sua solução requeria que se identificasse dois números primos que, somados, produzissem o valor originalmente lido. Para isso bastava gerar inicialmente uma tabela de números primos menores que um milhão e depois se verificasse, para cada entrada, qual o primeiro par de primos (não necessariamente distintos) que somados geravam o valor desejado. Muitas equipes erraram por tentar gerar os primos a cada entrada lida, o que tornaria a execução muito lenta, produzindo a mensagem Time Limit Exceeded.

Problema I

Trilhos do Trem

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

Autor: Anderson Viçoso de Araújo (UFMS)

A empresa de trens e metrôs *NewTrainBraza* ganhou uma licitação para construir estações e trilhos em cidades de grande porte do Brasil com o intuito de reduzir o fluxo de carros pelas ruas e melhorar a qualidade de vida dos seus habitantes. Para isso, a empresa pediu para que seus técnicos definissem os tipos de trilhos (reto, curvo, na superfície, subterrâneo etc.) que serão instalados entre as estações das cidades.

Para tanto, os técnicos decidiram utilizar uma cadeia de caracteres que representa as estações e os tipos de trilhos existentes entre elas. O mapa dos trens da cidade *FooBar*, por exemplo, é definido por meio da cadeia de caracteres: "A 12 - ~ 7 _ B ~ 4 # C" (desconsiderando as aspas), onde todos os caracteres são separados por um espaço em branco e o significado de cada um deles é apresentado na Tabela 1. De acordo com esta cadeia de caracteres, entre a estação A e B, existem 12 trilhos retos (na superfície), 1 trilho em curva e 7 trilhos retos subterrâneos.

É importante notar que, em cada tipo de trecho, o trem desenvolve uma velocidade constante proporcional à sua velocidade de entrada. Desta forma, caso o trem saia a 100 km/h da estação A, ele percorrerá o trecho com 12 trilhos retos (na superfície) na mesma velocidade (de acordo com a Tabela 1, 100% da velocidade de entrada). Em seguida, ele percorrerá um trecho em curva com uma velocidade média de 70 km/h (70% da velocidade de entrada). E assim por diante.

Você, um especialista em computação, vai ajudar a empresa a construir um sistema para calcular o tempo que os seus trens demoram para realizar o trajeto entre uma estação e outra.

Tabela 4. Descrição das informações da cadeia de caracteres que representa o mapa dos trens.

Caracteres	Descrição	Função	Velocidade
-	Hífen	Trecho de trilho em reta na superfície	100%
_	Sublinhado (*Underline*)	Trecho de trilho em reta subterrânea	110%
~	Til	Trecho de trilho em curva	70%
#	Cerquilha	Trecho de trilho em superfície sobre ponte	80%
A-Z	Alfabeto (maiúsculas)	Indicação de uma estação	-
2-999	Número inteiro	Número de trechos com mesma característica	-

Entrada

A entrada é composta por vários conjuntos de casos de teste. A primeira linha de cada conjunto de casos de teste contém uma cadeia de caracteres *M* (entre 1 e 512 caracteres), correspondendo ao mapa dos trilhos de uma cidade qualquer. A cadeia *M* deve iniciar e terminar com uma letra maiúscula entre A e Z (que corresponde a uma estação do trem). Entre quaisquer letras maiúsculas, existe uma combinação dos caracteres descritos na Tabela 1, separados por espaços em branco. Nas combinações, é possível ter um número (entre 2 e 999) indicando o número de trilhos de mesmo tipo em sequência. A próxima linha indica três números inteiros *T*, *V* e *N* ($1 \leq T \leq 1000$, $1 \leq V \leq 1000$ e $1 \leq N \leq 1000$), separados por um espaço em branco, onde T representa o valor em metros correspondente a um trilho do percurso; V indica a velocidade

constante do trem em km/h em reta na superfície (100% na Tabela 1) e N o número de casos de teste a serem analisados. Seguem-se N linhas com dois caracteres cada, separados por um espaço em branco, *X* e *Y* (ambos com algum caractere entre A e Z), indicando a estação de partida e de chegada para as quais deve-se calcular o tempo de viagem, respectivamente. As entradas terminam com o fim do arquivo.

Saída

Para cada caso de teste lido, escreva o texto "Cidade X" onde *X* indica o número do conjunto de casos de teste associado a um mapa de trilhos. Para cada caso de teste, escreva uma linha com o tempo que o trem vai levar para chegar entre as estações definidas no teste. O formato da saída para o tempo deve estar de acordo com a *string*: **HHh MMmin SSs,** onde as maiúsculas *HH* ($00 \leq HH \leq 99$), *MM* e *SS* ($00 \leq MM$ e $SS \leq 59$) indicam os números inteiros correspondentes às horas, minutos e segundos, respectivamente. **IMPORTANTE: valores com precisão de tempo menores que segundos (centésimos) devem ser truncados**. Por fim, uma linha em branco deve ser deixada após cada conjunto de casos de teste.

Exemplos

Entrada:	**Saída:**
A 12 - ~ 7 _ B ~ 4 # C 100 72 2 A B A C A 84 ~ - # B ~ _ 26 - C ~ - 26 ~ # D - 12 ~ 12 - _ E 85 ~ 23 - F 29 - 3 _ - G - 62 ~ # 44 _ 24 ~ _ H 761 73 1 E F A 32 # - _ ~ B 93 ~ _ 24 # 45 ~ _ ~ 56 _ 59 ~ C 80 160 2 A C B C	Cidade 1: 00h 01min 38s 00h 02min 11s Cidade 2: 01h 30min 20s Cidade 3: 00h 12min 16s 00h 10min 58s

Solução

Este é um problema considerado de dificuldade média, usa física (cálculo do tempo) e manipulação de strings.

Podemos chegar a solução deste problema através dos passos:

1. Para cada string de trechos lida, incrementar um contador de cidades e imprimir o rótulo da cidade corrente;
2. Depois de ler as entradas, converter a velocidade de entrada para m/s;
   a. Para cada pesquisa de percurso entre estações, percorrer a string do caminho, retirando os espaços
   i. Procurar a letra correspondente a estação de início para iniciar o processamento dos caracteres;
   ii. Verificar se o token é um número, se sim, armazenar para multiplicação posterior;
   iii. Caso o token seja um dos caracteres indicados na tabela (-,_,~ e #) verificar a porcentagem da velocidade correspondente que deve ser considerada, multiplicar pelo número anterior (caso exista), multiplicar pela quantidade de metros por trecho e dividir pela velocidade no trecho. Somar os resultados, que vai corresponder ao tempo em segundos;
   iv. Ao encontrar a letra correspondente a estação final, parar o processamento da cadeia;
   v. Transformar o tempo em segundos para horas, minutos e segundos usando resto e divisão por 60;
   vi. Imprimir a string do tempo.

Problema J

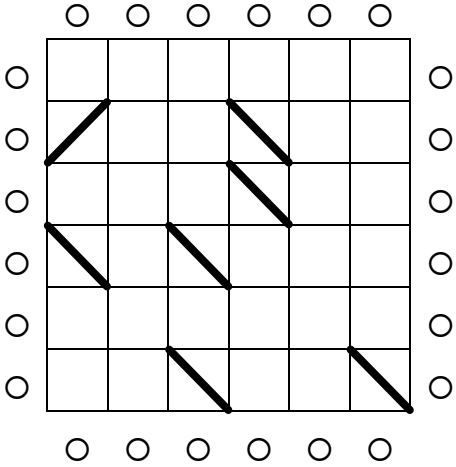
Treinamento Cerebral

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

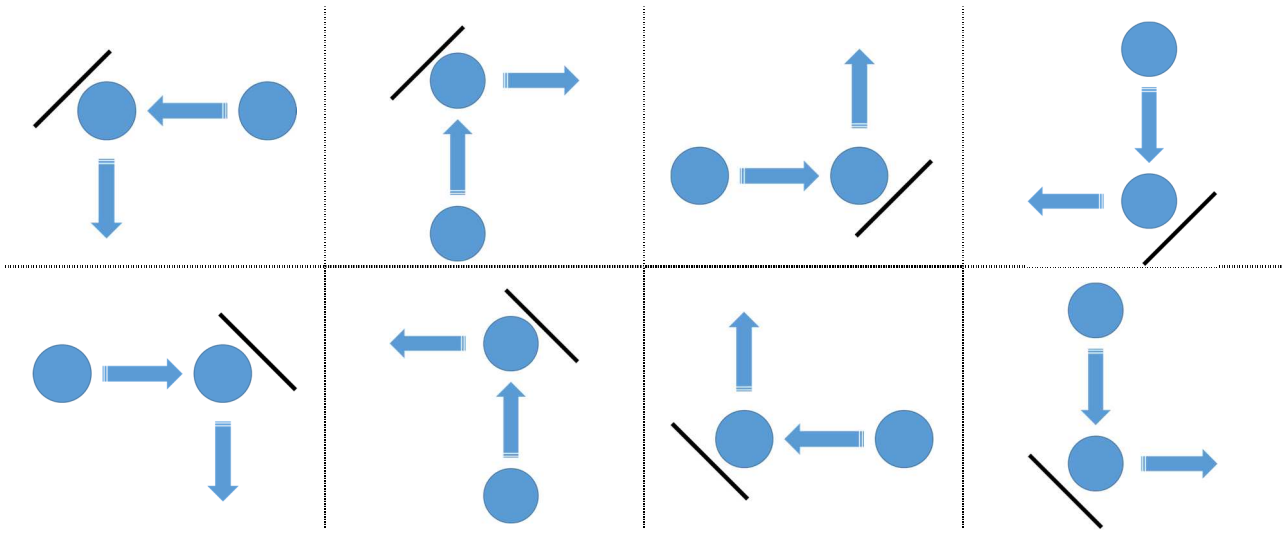
Autor: Leandro Luque (Fatec Mogi das Cruzes)

Nos últimos anos, diversos jogos para treinamento cerebral têm sido desenvolvidos e disponibilizados em diferentes plataformas. Estes jogos geralmente lidam com retenção de informações, visão periférica, atenção dividida, entre outras habilidades.

Um destes jogos, relacionado à retenção de informações, envolve o jogador se lembrar das posições de barras diretas (/) e invertidas (\) inseridas em um tabuleiro com N x N casas. A figura seguinte apresenta uma possível configuração do tabuleiro deste jogo, com 6 x 6 casas e algumas barras.

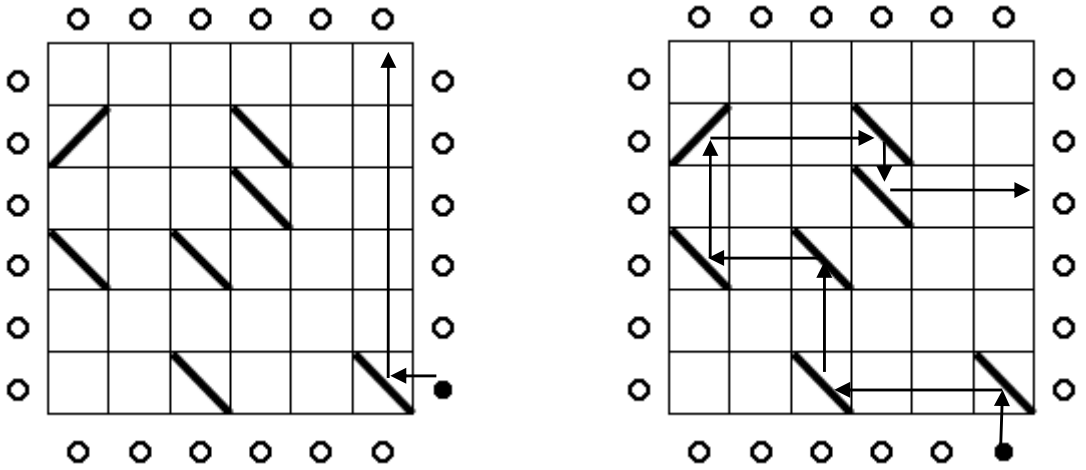


Os pontos nas laterais do tabuleiro indicam locais de onde bolas podem ser lançadas. Quando uma bola é lançada, ela segue em linha reta até encontrar uma barra. Neste caso, ela rebate na barra, conforme ilustrado nas figuras seguintes, e muda de direção.



Após apresentar as barras no tabuleiro por um curto intervalo de tempo para o jogador (geralmente 1 segundo), estas desaparecem e o jogo informa um ponto de onde uma bola será lançada. O jogador deve então especificar onde a bola irá parar: em qual linha ou coluna, e borda, ela sairá do tabuleiro.

Nas figuras seguintes, são apresentados dois exemplos de caminhos que a bola seguirá caso ela seja lançada do ponto escuro (preenchido). No primeiro caso (esquerda), a bola é lançada da 6ª linha, borda direita, rebatendo em apenas uma barra e saindo do tabuleiro na 6ª coluna, borda superior. No segundo caso (direita), a bola é lançada da 6ª coluna, borda inferior, rebatendo em sete barras e saindo do tabuleiro na 3ª linha, borda direita.



Você foi convidado para codificar esse jogo em Android. Para tanto, deve implementar uma funcionalidade que, dado um tabuleiro, suas barras e uma posição de lançamento da bola, determina em qual linha ou coluna, e borda, a bola irá sair do tabuleiro.

Entrada

A entrada contém vários casos de teste. A primeira linha de cada caso de teste contém um inteiro N ($4 \leq N \leq 50$), indicando o número N de linhas e colunas do tabuleiro. A próxima linha contém um inteiro M ($0 \leq M \leq$ N*N), indicando o número de barras que serão informadas. As próximas M linhas contêm dois inteiros L e C ($1 \leq L$ e $C \leq N$) e um caractere B (B='D' ou B='I'), separados por um espaço simples, representando respectivamente a linha e coluna do tabuleiro onde se encontra uma barra e o tipo da barra ('D' indica uma barra direta e 'I' uma barra invertida). A última linha do caso de teste contém um inteiro O ($1 \leq O \leq N$), e um caractere X (X='E' ou X='D' ou X='S' ou X='I'), indicando respectivamente o número da linha ou coluna de onde uma bola será lançada e a borda da qual ela será lançada ('E': borda esquerda; 'D': borda direita; 'S': borda superior; ou 'I': borda inferior).

Saída

Para cada caso de teste, imprima um inteiro S ($1 \leq S \leq N$), e um caractere Y (Y='E' ou Y='D' ou Y='S' ou Y='I'), indicando respectivamente o número da linha ou coluna e a borda onde a bola deixará o tabuleiro ('E': borda esquerda; 'D': borda direita; 'S': borda superior; ou 'I': borda inferior).

Exemplos

Entrada:	**Saída:**
6	3 D

7 2 1 D 2 4 I 3 4 I 4 1 I 4 3 I 6 3 I 6 6 I 6 I	

Solução:

Uma possível solução é, a partir da posição onde foi lançada a bola, ir se locomovendo pelo Tabuleiro até atingir uma extremidade. Passos:

1- Crie uma Matriz para implementar o tabuleiro e inicialize todas as células com um caractere espaço;
2- Faça a leitura das barras e coloque-as na Matriz;
3- Após a leitura da Borda, defina a direção (Bdir), linha (Blin) e coluna (Bcol) da bola, guardando-as nas respectivas variáveis (Bdir, Blin, Bcol);
4- Enquanto as coordenadas da bola não sair das dimensões da Matriz:

a. Teste o caractere da posição atual da bola (Blin,Bcol) e decida qual a nova direção e também incremente a linha ou coluna. Por exemplo, se o caractere na célula da posição atual da bola é um 'D' e se a direção atual é para cima, isso significa que a bola deve ir para a Direita, então a nova direção será Direita e deve-se incrementar a coluna atual da bola. E assim, para as demais direções, Veja:

```
Se (tabuleiro[Blin][Bcol]=='D') //Barra p/ direita
      Se (Bdir==SOBE){
       Bdir=DIR;
       Bcol++; //incrementa coluna
      }
      Senão
        Se (Bdir==DESCE){
         Bdi=ESQ;
         Bcol--; //decrementa coluna
        }
        Senão
          Se (Bdir==ESQ){
            Bdir=DESCE;
            Blin++; //incrementa linha
          }
          Senão
            Se (Bdir==DIR){
              Bdir=SOBE;
              Blin--; //decrementa linha
            }
```

Fazer o mesmo caso a posição atual seja uma Barra Invertida 'I' ou uma célula vazia.

Após cada incremento, verifique se um das novas coordenadas da bola (Blin e Bcol) saiu da Matriz. Se sim, imprima por onde saiu. Senão, continue.