Universidade Federal de Goiás – UFG
Instituto de Informática – INF
Bacharelados (Núcleo Básico Comum)

Algoritmos e Estruturas de Dados 1 – 2023/1

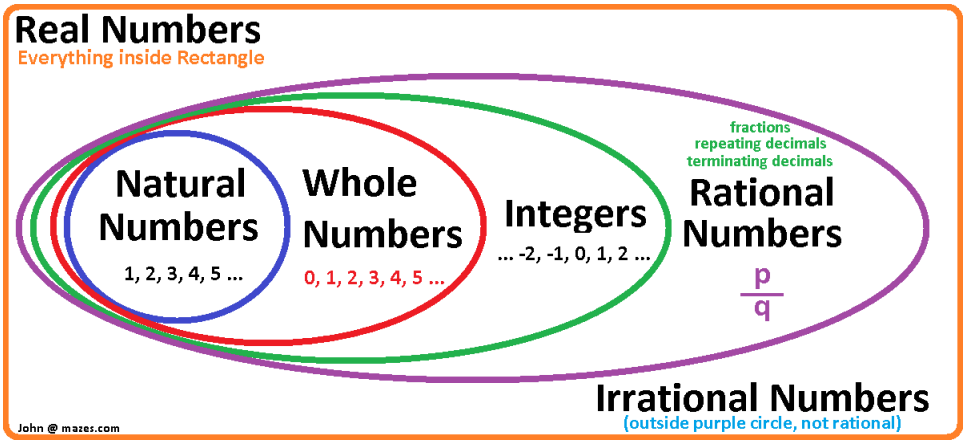
Lista de Exercícios nº 02 – Recursividade

(Turmas: INF0286 – Profa. Sofia Costa Paiva)

Sumário

1 Imprimindo números naturais recursivamente

 (+)



Os *números naturais* são os números utilizados ordinariamente para contagem:

$$\mathbb{N}^* = \{1, 2, 3, \ldots\}$$

e, por isso, às vezes são chamados de *números de contagem*. Eles são ditos *naturais* devido à nossa experiência natural, geralmente na infância, em que apenas manipulamos quantidades discretas de objetos: uma balinha, dois chiquetes, um pedaço de bolo, e certa quantidade de outras guloseimas. Ou, ainda, quando reclamávamos de ter "*muitas tarefas*" que a professora havia "*passado*" para casa – em verdade eram apenas três pequenos exercícios!
Ao matemático alemão Leopold Kronecker (1823 – 1891) está associada a seguinte frase:
"Deus criou os números naturais; o resto é obra do homem.".
Por algum tempo houve polêmica quanto ao numeral 0 (zero) pertencer, ou não, aos números naturais, já que, habitualmente, não se inicia uma contagem pelo valor "zero". Entretanto ele representa um conceito importante: a ausência de elementos num conjunto, seja ele abstrato ou concreto.
A Matemática contemporânea representa o conjunto destes números por meio do símbolo $\mathbb{N}$, incluindo o 0 (zero). Para excluí-lo utiliza-se o asterisco como expoente: $\mathbb{N}^*$, como feito no exemplo inicial desta questão.
A partir deste conceito inicial a respeito dos números naturais, deseja-se que você escreva um programa, em $\mathbb{C}$, para imprimir os n primeiros números naturais usando o conceito de recursividade, que os define da seguinte maneira:

$$n_0 = 0$$
$$n_{i+1} = n_i + 1, i \in \{0, 1, 2, \ldots\}$$

Entrada

A única linha da entrada contém um único natural n, indicando que se deseja imprimir os n primeiros números naturais, sendo que $n \in \mathbb{N}^*$ e $n \leq 5000$.

Saída

Seu programa deve imprimir uma única linha, contendo os n primeiros números naturais separados por um único espaço em branco entre eles.

Exemplos

Entrada	Saída
37	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 30 31 32 33 34 35 36 37

Entrada	Saída
50	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 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

Observação: Nos exemplos anteriores a saída *parece* ocupar mais de uma linha devido à restrição de largura da página impressa. Apesar disso, considere que todos os números apresentados na saída estão *numa única linha* da saída.

2 Função de Ackermann

 (+)

Na teoria da computabilidade, a *Função de Ackermann* (f_{ack}), nomeada por Wilhelm Friedrich Ackermann (1896 – 1962), é um dos mais simples exemplos de uma função computável que não é função recursiva primitiva. Todas as funções recursivas primitivas são totais e computáveis, mas a *Função de Ackermann* mostra que nem toda função total-computável é recursiva primitiva.
Depois que Ackermann publicou sua função (que continha três números naturais como argumentos), vários autores a modificaram para atender a diversas finalidades. Então, a f_{ack} pode ser referenciada a uma de suas várias formas da função original.
Uma das versões mais comuns, a *Função de Ackermann-Péter*, que possui apenas dois argumentos, é definida a seguir para números naturais m e n:

$$f_{ack}(m,n) = \begin{cases} (n+1), & \text{se } m = 0 \\ f_{ack}(m-1,1), & \text{se } n = 0, m > 0 \\ f_{ack}(m-1, f_{ack}(m,n-1)), & \text{se } n > 0, m > 0 \end{cases}$$

Entrada

A única linha da entrada contém dois números naturais m e n separados por um único espaço em branco, nesta ordem, representando os parâmetros para a *Função de Ackermann*.

Saída

Seu programa deve imprimir uma única linha com o valor da f_{ack} para os dois parâmetros recebidos.

Exemplos

Entrada	Saída
0 7	8

Entrada	Saída
3 0	5

Entrada	Saída
3 2	29

Entrada	Saída
2 4	11

3 Reverso de um número natural

 (+)

Todo número natural estritamente positivo $n \in \mathbb{N}^*$ possui um *número reverso* correspondente. Por exemplo, considere que n seja escrito da seguinte maneira:

$$n = d_k d_{k-1} d_{k-2} \cdots d_2 d_1 d_0$$

onde $k \in \mathbb{N}^*$ corresponde ao número de dígitos significativos que formam n, ou seja, $d_k \in \{1,2,3,\ldots,9\}$ e $d_i \in \{0,1,2,\ldots,9\}$, com $0 \leq i < k$.
O *número reverso* de n é $n^r = d_\ell d_{\ell-1} d_{\ell-2} \cdots d_{k-2} d_{k-1} d_k$, sendo d_ℓ o primeiro dígito não nulo, tomados nesta ordem, dentre $d_k d_{k-1} d_{k-2} \ldots d_2 d_1 d_0$ do número original n.
Escreva uma função recursiva, em $\mathbb{C}$, que seja capaz de determinar o *número reverso* de um certo número natural estritamente positivo n fornecido como entrada.

Entrada

A única linha da entrada contém um único número natural estritamente positivo, n, $1 \leq n \leq 10^6$.

Saída

Seu programa deve imprimir uma única linha com o valor de n^r, o *número reverso* de n.

Exemplos

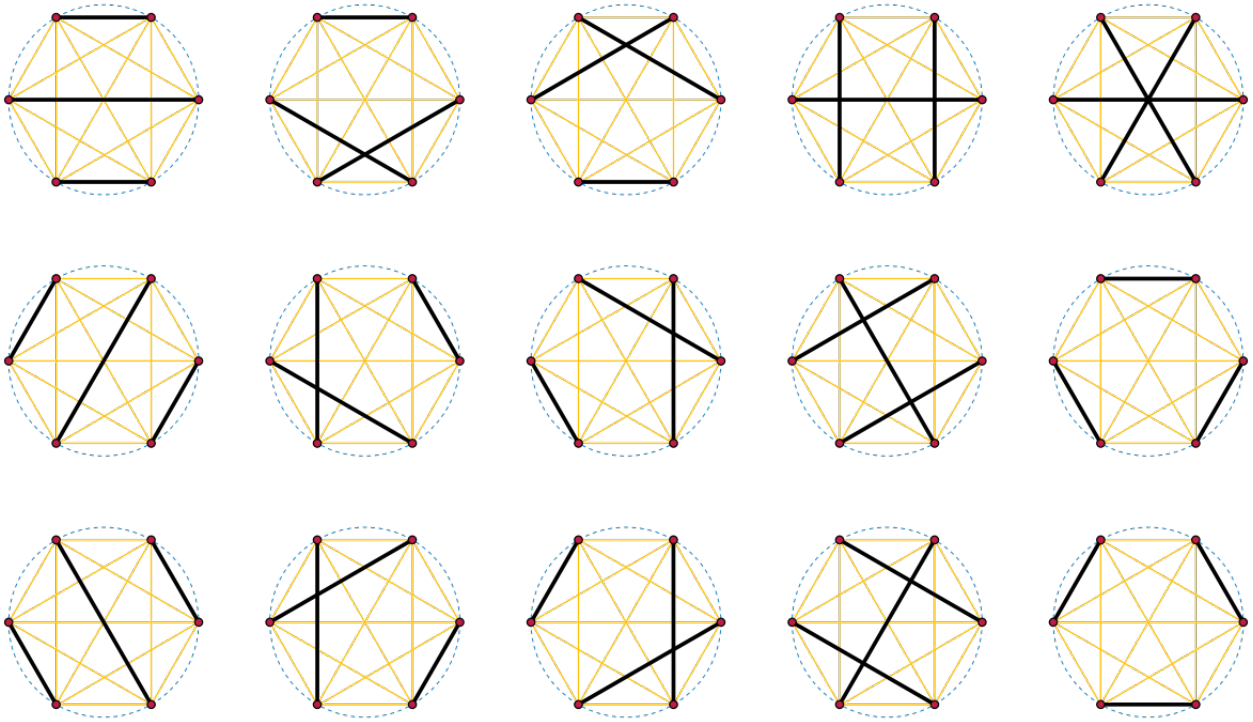
Entrada	Saída
411	114

Entrada	Saída
1230	321

Entrada	Saída
138000	831

4 Fatorial duplo

 (++)



Pode-se definir uma função $\ddot{f}(n)$, *fatorial duplo* de n, com $n \in \mathbb{N}$, como sendo o produto de todos os números naturais ímpares de 1 até n, inclusive este, quando ele é ímpar. Assim, por exemplo, tem-se que:

$$\ddot{f}(1) = 1$$
$$\ddot{f}(2) = 1$$
$$\ddot{f}(3) = 3$$
$$\ddot{f}(5) = 15$$

Você deve escrever uma função recursiva, em $\mathbb{C}$, que seja capaz de, recebendo n, imprimir o valor de $\ddot{f}(n)$.

Entrada

A única linha de entrada contém o valor de n, com $1 \leq n \leq 100$.

Saída

Imprima uma única linha de saída, com o valor de $\ddot{f}(n)$.

Exemplo

Entrada	Saída
1	1

Entrada	Saída
7	105

Entrada	Saída
10	945

5 O banco inteligente

 (+++)

Os atuais caixas automáticos dos bancos, ou ATMs – *Automated Teller Machines*, são uma ótima invenção mas, às vezes, precisamos de dinheiro *trocado* e a máquina nos entrega notas de $R\$100,00$. Noutras vezes, desejamos sacar um valor um pouco maior e, por questão de segurança, gostaríamos de receber todo o valor em notas de $R\$100,00$, mas a máquina nos entrega um *monte* de notas de $R\$20,00$.
Para conquistar clientes, o `Banco Inteligente` (`BI`) está tentando minimizar este problema dando aos clientes a possibilidade de escolher o valor das notas na hora do saque. Para isso, eles precisam da sua ajuda para saber a resposta para a seguinte questão: dado um determinado valor S de saque (em reais) e quantas notas de cada valor a máquina tem, qual é o número de maneiras distintas que há para entregar o valor S ao cliente?
Sabe-se que nas ATMs do `BI` há escaninhos para notas de 2, 5, 10, 20, 50 e de 100 reais.
Por exemplo, suponha que para certo cliente X tenha-se que $S = 22$ e que o número de notas de cada valor presente na ATM no momento da solicitação deste saque é:

$$\begin{aligned} N_2 &= 5 \\ N_5 &= 4 \\ N_{10} &= 3 \\ N_{20} &= 10 \\ N_{100} &= 10 \end{aligned} \tag{1}$$

Assim, há QUATRO maneiras distintas da máquina entregar o valor do saque solicitado:

1ª : uma nota de $R\$20,00$ e uma nota de $R\$2,00$ (duas notas);

2ª : duas notas de $R\$10,00$ e uma nota de $R\$2,00$ (três notas);

3ª : uma nota de $R\$10,00$, duas notas de $R\$5,00$ e uma nota de $R\$2,00$ (quatro notas);

4ª : quatro notas de $R\$5,00$ e uma nota de $R\$2,00$ (cinco notas).

Tarefa

Escrever, em $\mathbb{C}$, um programa que seja capaz de determinar o número de maneiras possíveis de atender à solicitação de saque do cliente.

Entrada

A primeira linha da entrada contém o número natural S expressando, em reais, o valor do saque desejado. A segunda linha contém seis inteiros N_2, N_5, N_{10}, N_{20}, N_{50} e N_{100}, respectivamente, indicando o número de notas de 2, 5, 10, 20, 50 e 100 reais disponíveis na ATM no momento do saque. Os números estão separados por um único espaço em branco entre eles.

Saída

Seu programa deve imprimir um único número natural: a quantidade de maneiras distintas da máquina atender ao saque solicitado.

Restrições

- $0 \leq S \leq 5000$ e $N_i \leq 500$, $\forall i \in \{2, 5, 10, 20, 50, 100\}$.

Exemplos

Entrada	Saída
22 5 4 3 10 0 10	4

Entrada	Saída
1000 20 20 20 20 20 20	34201

Entrada	Saída
50 1 1 1 1 0 1	0

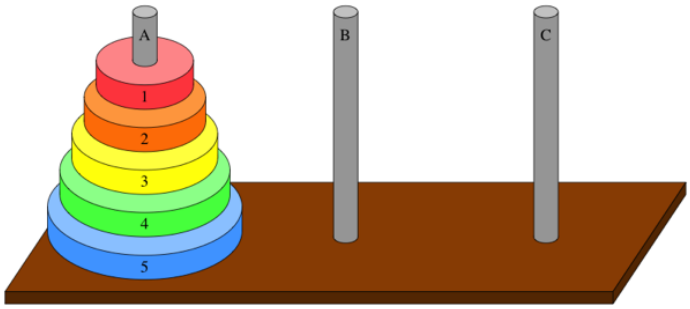
Observações

Pense como seria alterar este exercício para cada uma das seguintes variações:

1. Considerando o valor S solicitado, a ATM deverá entregar para o usuário o maior número de notas possível para a realização daquele saque. A saída deverá ser o número de notas entregues de cada tipo de cédula, na seguinte ordem: N_2, N_5, N_{10}, N_{20}, N_{50} e N_{100}.
   O exemplo nº 01, onde $S = 22$, teria como saída a sequência: 1 4 0 0 0 0. Ou seja, uma nota de R\$2,00 e quatro notas de R\$5,00.

2. Considerando o valor S solicitado, a ATM deverá entregar para o usuário o menor número de notas possível para a realização daquele saque. A saída deverá ser o número de notas entregues de cada tipo de cédula, na seguinte ordem: N_2, N_5, N_{10}, N_{20}, N_{50} e N_{100}.
   O exemplo nº 01, onde $S = 22$, teria como saída a sequência: 1 0 0 1 0 0. Ou seja, uma nota de R\$2,00 e uma nota de R\$20,00.

Entrada	Saída
50 2 2 2 2 2 2	4



6 Torre de Hanoi

 (+++)

Torre de Hanói é um "quebra-cabeça"que consiste em uma base contendo três pinos (ou hastes), em um dos quais são dispostos alguns discos, uns sobre os outros, em ordem crescente de diâmetro, de cima para baixo. O problema consiste em passar todos os discos de um pino para outro qualquer, usando um dos pinos como *auxiliar*, de maneira que um disco maior nunca fique em cima de outro menor, em nenhuma situação. O número de discos pode variar, sendo que o mais simples contém apenas três discos. (Fonte: Wikipédia)

Suponha que os pinos se chamam “O” (origem), “D” (destino), “A” (auxiliar), faça um programa recursivo que resolva a Torre de Hanói.

Entrada

O arquivo de entrada consiste de uma única linha contendo um número natural n, $n \in \mathbb{N}^* \mid 2 \leq n \leq 1000$, que indica a quantidade de discos contidos no pino de origem – pino “O”. Os discos são, sempre, numerados de 1 a n, indicando o diâmetro do disco (numa determinada unidade de medida qualquer).

Saída

A saída deve conter os movimentos a serem realizados para se resolver a *Torre de Hanói* com os n discos. Cada movimento deve estar em uma linha no formato de par ordenado na forma (<pino de origem>,<pino de destino>), onde <pino de origem> e <pino de destino> $\in \{O, D, A\}$.

Exemplos

Entrada	Saída
2	(O,A) (O,D) (A,D)

Observações

Note que não há “*espaço em branco*” entre as letras indicativas dos pinos nas respostas, como também para os parênteses. Além disso, todas as letras são grafadas em maiúsculas.

Entrada	Saída
3	(O,D) (O,A) (D,A) (O,D) (A,O) (A,D) (O,D)

7 Setas

 (++++)

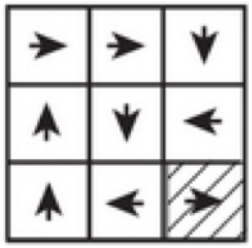
Gabriel é um garoto que gosta muito de um jogo eletrônico onde há várias letras num tabuleiro – que fica sobre o piso – e o jogador precisa, rapidamente, pisar nas letras corretas, de acordo com as instruções que aparecem na *tela de projeção* que está à sua frenta, seguindo uma música ao fundo.

Cansado de vencer o "jogo", Gabriel inventou um novo:

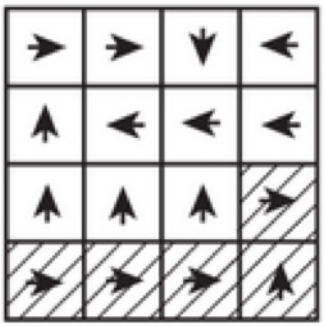
Agora temos um tabuleiro quadrado, com n células de cada lado, em que cada célula possui uma seta que aponta para uma das quatro posições vizinhas (►,◄,▲,▼). O jogador primeiro escolhe uma célula inicial para se posicionar e, quando a música começa, ele deve caminhar na direção para onde a seta em que ele está naquele momento apontar. Ganhará o jogo quem pisar em mais setas *corretas* durante um determinado período de tempo previamente fixado.

O problema é que Gabriel joga tão rápido que quando a seta atual *manda* ele "sair do tabuleiro", ele segue a orientação, muitas vezes quebrando alguns objetos próximos ao tabuleiro. Quando isso acontece, dizemos que a célula inicial deste jogo é uma célula *não segura*, pois leva a um caminho que termina fora do tabuleiro.

A figura a seguir mostra dois tabuleiros: um 3×3 e outro 4×4, respectivamente, com oito e onze células *seguras*:



Tabuleiro 3x3 com oito células seguras



Tabuleiro 4x4 com onze células seguras

As células seguras de cada tabuleiro são as seguintes:

3×3 – todas, exceto a (3, 3);

4×4 – (1,1); (1,2); (1,3); (1,4); (2,1); (2,2); (2,3); (2,4); (3,1); (3,2) e (3,3).

Sua tarefa é ajudar Gabriel: construa um programa $\mathbb{C}$ que indique, a partir de uma dada configuração do tabuleiro fornecida, quantas células são *seguras* para ele iniciar o jogo.

Entrada

A primeira linha da entrada contém o número natural n, o tamanho do tabuleiro, com $1 \leq n \leq 500$. Cada uma das n linhas seguintes contém n caracteres, com as direções das setas, sem nenhum espaço entre elas. As direções válidas são:

'V' (letra V, maiúscula) aponta para a célula da linha abaixo, na mesma coluna;

'<' (sinal menor-que) aponta para a célula à esquerda, na mesma linha;

'>' (sinal maior-que) aponta para a célula à direita, na mesma linha;

'A' (letra A, maiúscula) aponta para a célula da linha acima, na mesma coluna.

Saída

Seu programa deve produzir um único número natural k: o número de células seguras naquela configuração do tabuleiro.

Exemplos

Entrada	Saída
3 > > V A V < A < >	8

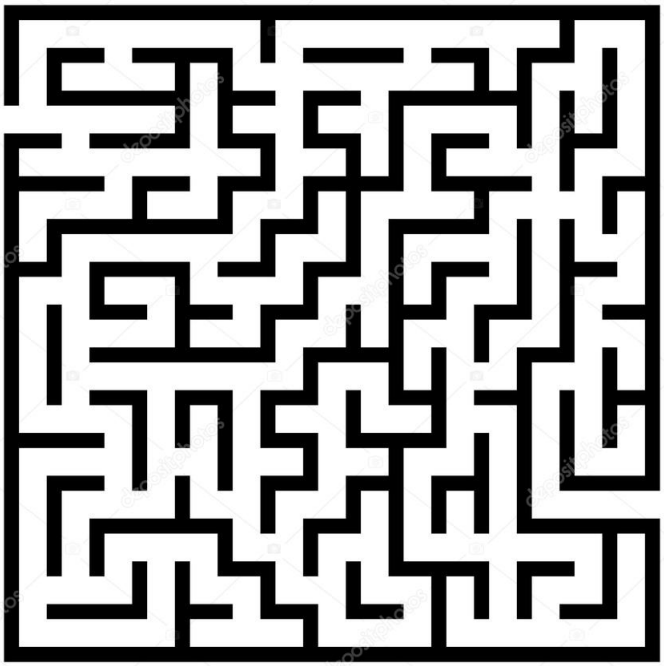
Entrada	Saída
4 > > V < A < < < A A A > > > > A	11

Entrada	Saída
4 V > > > V > V < > A > V < < V <	0

Entrada	Saída
5 > > V < < V > V V A V > > > A > > A A < > > A > A	25

8 Labirinto – 1

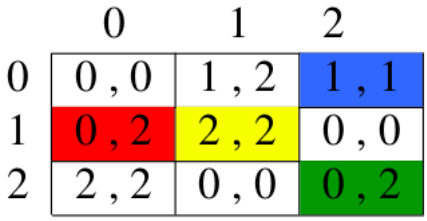
(++++)



Considere que você está jogando um intrigante "*Jogo de Labirinto*".
O labirinto é, em verdade, uma matriz $m \times n$, onde casa dessa matriz possui uma coordenada (x, y) da próxima casa onde você deverá ir, e a saída do labirinto sempre será a posição $(0, 0)$.
Por exemplo, observando a figura abaixo temos que o você está na casa vermelha, que é a posição $(0, 1)$ e dela você irá para a posição amarela $(1, 2)$ e da posição amarela você irá para a posição verde $(0, 0)$ – indicando que você conseguiu *sair* do labirinto e, portanto, venceu o jogo. Parabéns!

	0	1	2
0	0 , 0	1 , 2	1 , 1
1	0 , 2	2 , 2	0 , 0
2	2 , 2	0 , 0	0 , 2

Noutro exemplo, ao invés de iniciar na posição $(0, 1)$, você iniciaria na posição $(1, 0)$. Neste caso, você sairia da posição vermelha $(1, 0)$ e iria para a posição azul $(0, 2)$. De lá, você iria para a posição amarela $(1, 1)$ e então iria para a posição verde $(2, 2)$. Da posição verde, você voltaria para a posição azul $(0, 2)$, o que caracteriza que você entrou em "*looping*". Por isso, partindo da posição $(1, 0)$ não é possível chegar à saída do labirinto, ou seja, é impossível ganhar o jogo a partir dela. Lamento!



Você deverá escrever um programa $\mathbb{C}$ para *simular* este jogo de acordo com as especificações a seguir.

Entrada

A primeira linha da entrada contém as dimensões m e n da matriz, sendo o número de linhas e de colunas, respectivamente. Sabe-se que $m, n \in \mathbb{N}^*$ e que $1 \leq m, n \leq 100$.
As m linhas seguintes, contém, cada uma, os n pares de coordenadas de cada célula da matriz, com todos os números sendo separados por um único espaço em branco em relação seu anterior e posterior. Obviamente, o primeiro número da linha não tem anterior e o último número não tem posterior.
Por fim, a última linha contém as coordenadas da posição inicial, (x, y), a partir de onde o jogo começará, representa por meio dos números x e y, separados por um único espaço em branco, e na ordem especificada: x seguido de y.

Saída

A palavra VENCE (em letras maiúsculas), se for possível ganhar o jogo, ou seja, sair o labirinto a partir de uma certa posição (x, y) inicial.
A palavra PRESO (em letras maiúsculas), caso seja impossível ganhar o jogo.

Exemplos

Entrada	Saída
3 3 0 0 1 2 1 1 0 2 2 2 0 0 2 2 0 0 0 2 0 1	VENCE

Entrada	Saída
3 3 0 0 1 2 1 1 0 2 2 2 0 0 2 2 0 0 0 2 1 0	PRESO